SUNY OSWEGO CSC466 AI RESEARCH PROJECT

Artificial Neural Network For The Assessment Of Probability Of Winning From A Game State In Dobo

Jake Sauter

May 4, 2017

1 ABSTRACT

A Neural Net will be made to assess the probability of winning from a game state, with input being n neurons representing the partitions of the board and output being one neuron, with a value representing the probability of winning from the game state. First, a game state table will be made, with entries to represent a game state, and the probability of winning from that game state. After each game the sequence of states will be analyzed, the probability of a state in the winning sequence of moves will be incremented and the probability of winning from a game state in the losing sequence will be decremented. Once the game state table is made the Neural Net will be trained with the game table data and weights adjusted properly with a genetic algorithm using the absolute value of the difference of the output of the net and the table entry value, summed over all entries. Finally once the Neural Net is trained, a Neural Net Player will have the power of a Neural Net to assess its next possible choices of moves, and will pick the move with the assessed highest probability of winning

2 INTRODUCTION

Dobo is a two player turn-based game in which a valid move consists of removing a horizontal or vertical contiguous line of stones off of the rectangular shaped game board. The object of the game is to make the other player have no other option than to remove the last stone. This project is a study of a strategy for the game to increase the odds of winning for a player that uses a machine learning algorithm. First the probability value of winning from each state will be found by using one type of machine learning method, then that data will be used as the training data for a neural net to try and compress the data into a more memory and time efficient encapsulation. An explanation of implementation will be reviewed in regards to the Rote machine learning method used to generate the probability values, along with how the neural network was modeled and eventually trained via a genetic algorithm. There will also be some explanation of back-propagation as this was studied heavily as a training option for the neural network.

3 Background

A large amount of work has been done in recent decades in the area of neural networking. There has been research specifically pertaining to the use of neural nets in many areas from determining the best configuration for starting an energy production system [3] to game playing programs. A few main methods exist in training neural networks, however the use of backpropagation and the use of genetic algorithms, sometimes together are very accepted methods of training a neural network to fit a data set.

Research for this project lead to the research of several papers that included the use of neural networks to try and play tic-tac-toe [13], generation of a no loss strategy for tic-tac-toe [1], training neural networks with the use of genetic algorithms [10] [15] , backpropagation [7] and their combination [3].

In Training an artificial neural network to play Tic-Tac-Toe [13] Siegel reviewed a reinforcement learning technique for generating the probability values for each state in the game of tic-tac-toe. The method mentioned in this paper was adopted and slightly altered for use in this project. The differences come from how the values of each state are updated. In SiegelâĂŹs version, he uses the surrounding states in the sequence to adjust the value, while the approach of this project was purely empirical. More specifically, Siegel analyzed the game in reverse order of the states appearing, which is helpful, because it it known that the last state in the game is a losing/winning state depending on what player you are generating probabilities for. After this initial adjustment of the final game state, the program starts to iterate backwards over the game states, adjusting the probability value of each state away from the probability value of the latter state, by a learning factor constant.This method was adjusted slightly for use in this project, the differences mentioned in the Program Description section.

Genetic algorithms have become a useful tool in the training of artificial neural networks. In Genetic Algorithms and Neural Networks [15] Whitely mentions many ways in which the use of genetic algorithms can assist the training of neural networks. The three main methods that he mentions include using genetic algorithms for preprocessing data, genetic algorithms for training neural networks, and genetic algorithms for constructing neural networks. When used for preprocessing training data, genetic algorithms work to reduce the size of the set of the training data by finding a smaller, but still very representative set to allow for faster training.

When used for training neural networks, the most elementary idea is that an individual of the genetic algorithm is some weight configuration for the network, with the cost of an individual being the cost of the network, so the genetic algorithm is physically setting the weights of the network as the cost of the network goes down generation to generation, being the method adopted by this project. This method of training a neural network can sometime be more successful than backpropagation as seen in Training Feedforward Neural Networks Using Genetic Algorithms [10]. Genetic algorithms can also help optimize the learning rate, weight initialization and network topology of networks to find the quickest training time possible for neural nets being trained with other methods such as backpropagation. An individual of the genetic algorithm in this case would still be the weight values of the neural network, however the cost would be the total training time as the network, so as generations progressed there would be a visible trend in reducing training time of the network.

Backpropagation is a very accepted and powerful technique for training neural networks. The general idea in backpropagation is to calculate (with calculus) how responsible each individual weight is for the total error, so that the identified weights can be adjusted closer to the optimal value by the increment of the learning rate. There are three methods of calculating the gradient of the weights, being the individual method, batch method, and total method. In individual gradient descent, the gradient of the weights is calculated for a single point in the data set the neural net is being trained on, and updated, with the process being repeated for each data point in the set for a certain amount of epochs. In batch gradient descent, some number of data points are selected and the total error of those data points is used to calculate the gradient. Finally in total gradient descent, the error of the neural net over the total set of data points is used to calculate the gradient. These types of gradient descent change the behavior of training. The behavior of training gets less sporadic as the batch size increases, however if the batch size is too large training takes a really long time and the changes in the weights are less helpful. In Efficient Backprop [7] the authors walk through the derivation of the mathematical soundness of backpropagation and they mention the strengths and weaknesses of stochastic vs. batch gradient descent, similar to mentioned prior. In this paper there is also mention of momentum and other very interesting topics of which will not be described in in this paper, however a mention to the extraordinary content of the paper was due.

The combination of genetic algorithms and backpropagation, as slightly mentioned before is a very exciting combination of two already interesting concepts on their own. In Combining Back-Propagation and Genetic Algorithms [3] a case study of combining these two interesting approaches to train a neural network is described. The approach taken of training in this paper was to train a certain amount of neural networks using simple backpropagation. Then these pre-trained neural networks were used as the individuals in the initial population of the genetic algorithm that further adjusted the weights to an optimal initialization. A significant improvement was seen by combining the techniques over using just one of the techniques. This technique was the initial goal of the project, however complications lead to just the use of a genetic algorithm because with enough time a genetic algorithm will very likely find the optimal configuration. Finally in Danger of Setting All Initial Weights to Zero in

Backpropagation [5] an explanation is given as to why it is a better idea to set the initial weights of a neural network stochastically as opposed to all zero, or even all weights to 1.

4 Program Description

The general plan of this project is to model the game of Dobo, use a reinforcement learning method to find the probability values of each state, then use a neural network to try and compress the probability values of the game states into a more temporal and spatially efficient model. A simple genetic algorithm was used to train the neural network, however a lot of research was done on the optimal configuration of the parameters of the genetic algorithm.

Modeling the game of Dobo was a fairly trivial task, however a good representation is necessary for all of the computations that need to be done on the board. The initial approach was the implementation of the board using a list of lists representing each row of the board. This model proved to be difficult to manipulate later on in the project, so the representation was reinvented using CLISPs built in multidimensional arrays. This representation proved much easier to manipulate later on in the implementation process, so the time spent reinventing the representation was well worth it. The play of the game was represented in a generic-play method that had parameters consisting of two players. The generic-play method would create an initial game state, and act as a middle layer between each player‘s make-move method which took a board state as an input and returned the board state after the move was applied.

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.1}$$

Figure 4.1: A representation of an initial Dobo game board

The next task was to generate the probability values of winning from each state using a Rote/Reinforcement learning technique. The approach adopted by this project uses a similar idea to Siegels [13] that the last state was a bad game state, and the state before that was a good game state, and so on until the initial game state. The way that the values were updated in this project were on a ratio basis. When a state was encountered the amount of “hits” on the state increased and if it was in the winning sequence then the amount of “wins” on the state were also increased. The probability of winning from the state was purely the quotient of the wins of the state and the hits of the state. This became a little more complex when it was realized that when the knowledge base is small, wrong decisions could be made to inaccurately value the state, so as the total amount of hits on the final state increased (the final state is a subset of every analyzed game sequence) so did the adjustment amount on each state. To clarify, say that there is 10 hits on the final state, then every state in the winning sequence will have 10 wins and 10 hits added to its entry, while every state in its losing sequence will only have 10 hits added to its entry. Consequently now the final state will have 20 hits, so the adjustment rate increases rapidly as the knowledge base becomes more accurate.

Modeling the game state table was also a fairly trivial task. An entry in the game state table is a game state (board), the amount of wins of the state, the amount of hits of the state, and the quotient of the two being the probability of winning from the state. Searching the game state table is very computationally expensive to search. There are many transformations of a board that are the same due to symmetry (some helpful ideas on symmetry of boards from [1]), so when searching the game state table up to 7 transformations need to be applied to a game state to check for equality with the table entry. The project will be discussed in detail in the 3x3 version of Dobo as this is the version that was most successful do to its smaller size, although any m x n board size is possible for the program to work on with enough time.

Next, the goal was to use the generated probabilities for each state as training data for a neural network to make a more efficient model for the data. The topology of the neural network had an input layer of 9 nodes, consisting of one node for each partition of the game board. The hidden layers can be dynamically set at the start of the genetic algorithm and the output layer is of size 1, with an output representing the probability of the input-vectorized game state (seen here [4] that the output of a neural network can easily represent a posterior probability). When building my neural network I was able to appreciate the beauty of the data structure from the inside-out, which induced me to refresh myself on a little linear algebra from [12]. Making a forward propagating net proved not too difficult with a little help from [9] and [11] .

The genetic algorithm used to train the neural network was a simple one. An individual in the genetic algorithm is a list that represents the weights of a neural network. The initialization of the individuals in the genetic algorithm was one such that all elements in the list were randomized between -1 and 1. This was an easy representation as all that was necessary was a method to convert a neural network to a list of weights and a method that can take a list and neural network as its parameters and return the neural net with the weights in the list initialized as the weights of the network. The mutation operation selected some number of random positions based on the set mutation aggression, and incremented or decremented the weight by the learning rate (which decreases asymptotically over time). The crossover operator picks a random number from zero to the size of the weight list (of which we will call n), and selects the first n positions of the mother list, and the rest of the positions from the father list to make a new individual. As is typical, as the genetic algorithm progresses through generations a certain percentage of the population is copied (and possibly mutated) over to the new generation, and the rest of the needed new population comes from crossovers (possibly being mutated) from the old population. I feel as if it is important to mention here that for the training data for the neural network I âĂIJnormalizedâĂİ the probability values, making and probability in the table under .5 to 0 and over .5 to 1. This made the problem more of a binary classification problem, taking any output of the network over .5 as 1 and under .5 as 0, and if the results for this problem worked well I wanted to continue to the study of the continuous version of the problem, trying to model the actual probability value as opposed to the class of the probability.

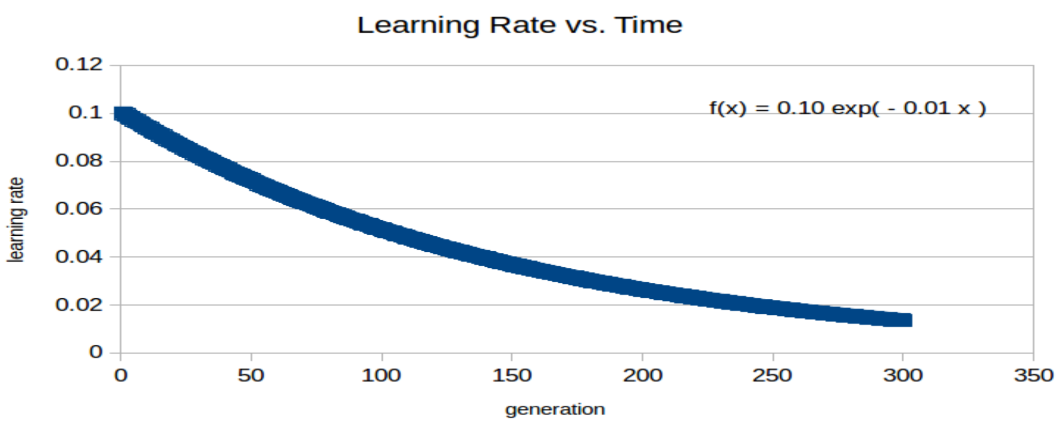


Figure 4.2: The asymptotic behavior of the learning rate.

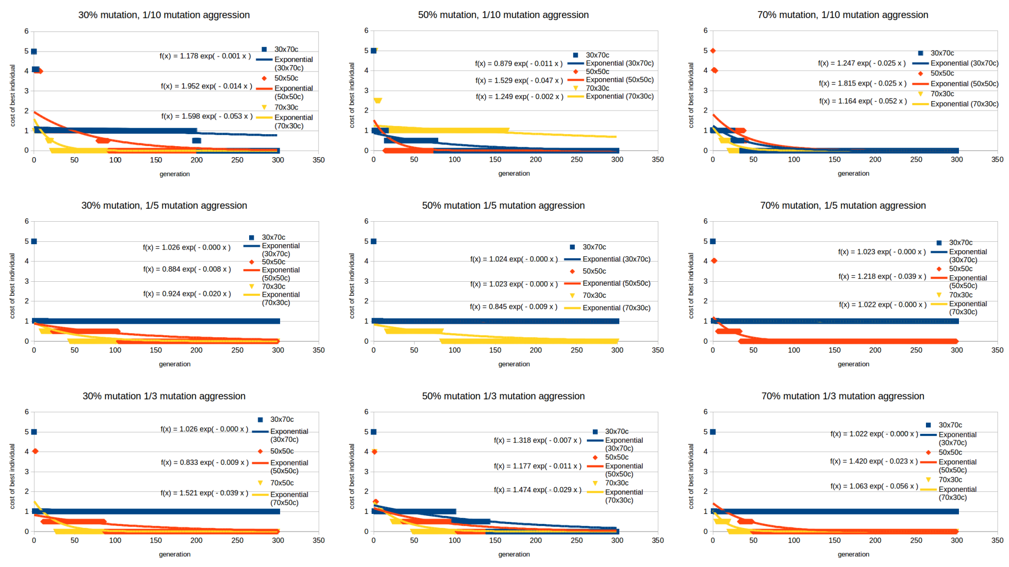
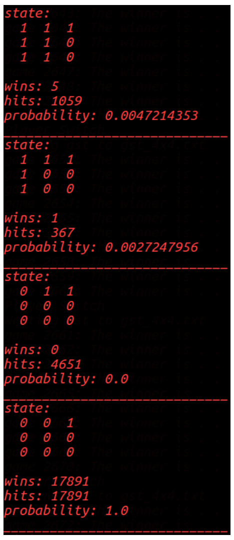


Figure 4.3: The graphical trends of changing meta variables of genetic algorithm.

At the end of the genetic algorithm, it is wished that the most fit individual be selected out of the population and that this individual fits the data modelled in the game state table well. Many different net configurations had to be tested to try and achieve the best fit possible, The maximum achieved retention rate will be discussed at a later point in the paper when it is time for reflections.

5 Demos



```
state:
  1  1  1
  1  1  0
  1  1  0

wins: 5
hits: 1059
probability: 0.0047214353
------------------------------
state:
  1  1  1
  1  0  0
  1  0  0

wins: 1
hits: 367
probability: 0.0027247956
------------------------------
state:
  0  1  1
  0  0  0
  0  0  0

wins: 0
hits: 4651
probability: 0.0
------------------------------
state:
  0  0  1
  0  0  0
  0  0  0

wins: 17891
hits: 17891
probability: 1.0
------------------------------
```

(a) A trained game state table

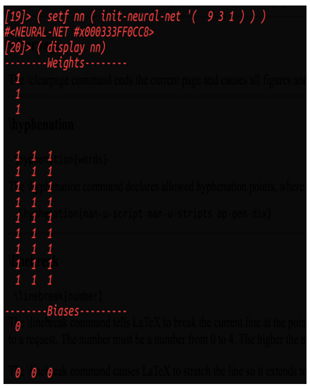
Making a table lookup player, Making an initial state
Name: TABLE-LOOKUP-PLAYER
Wins: 0
Losses: 0
Win Ratio: Undetermined

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Begin TABLE-LOOKUP-PLAYER move ... searching database for best move ... probability-list: (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) selecting ((0 2) (1 2) (2 2)) for my move max-p: 0, max: 1.1

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

(b) A demo of a table lookup player making a move

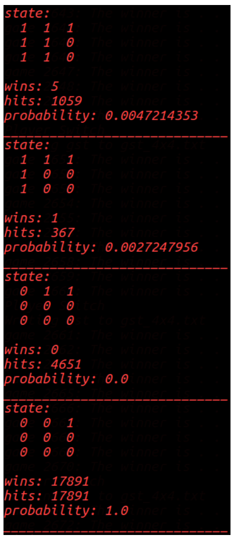


```
[19]> ( setf nn ( init-neural-net '(  9 3 1 ) ) )
#<NEURAL-NET #x000333FF0CC8>
[20]> ( display nn)
--------Weights--------
 1
 1
 1

 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
--------Biases---------
 0

 0  0  0
```

(a) A demo of a neural network.

initializing a neural net of size 9-3-1
input: $\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$
first layer activation: $\begin{bmatrix} 9 & 9 & 9 \end{bmatrix}$
activation of layer 1: 2.99963
net output: 0.9525574

(b) A forward propagation run of a neural network



```
state:
  1  1  1
  1  1  0
  1  1  0

wins: 5
hits: 1059
probability: 0.0047214353
------------------------------
state:
  1  1  1
  1  0  0
  1  0  0

wins: 1
hits: 367
probability: 0.0027247956
------------------------------
state:
  0  1  1
  0  0  0
  0  0  0

wins: 0
hits: 4651
probability: 0.0
------------------------------
state:
  0  0  1
  0  0  0
  0  0  0

wins: 17891
hits: 17891
probability: 1.0
------------------------------
```

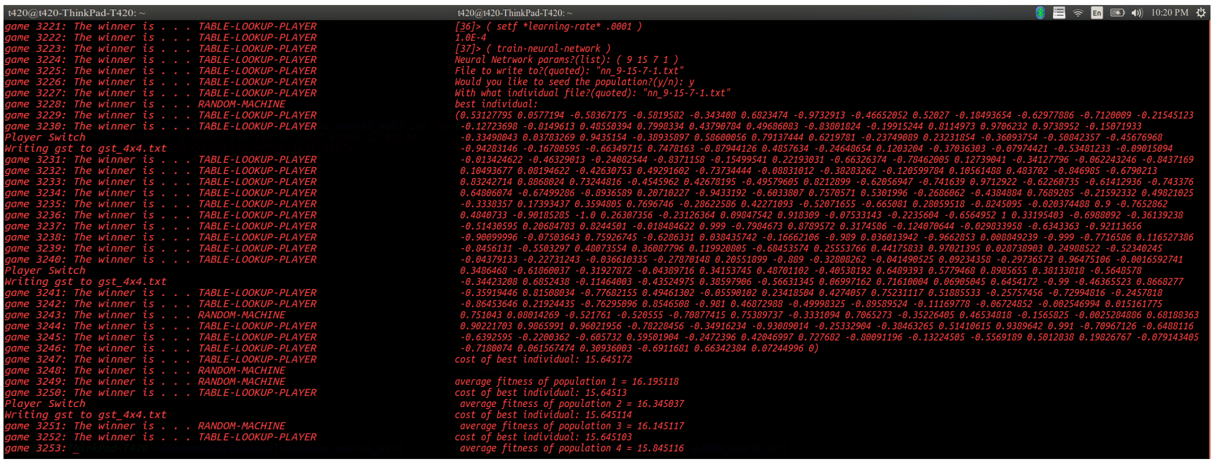
Making a table lookup player, Making an initial state
Name: TABLE-LOOKUP-PLAYER
Wins: 0
Losses: 0
Win Ratio: Undetermined

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Begin TABLE-LOOKUP-PLAYER move ... searching database for best move ... probability-list: (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) selecting ((0 2) (1 2) (2 2)) for my move max-p: 0, max: 1.1

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

(a) A trained game state table (b) A demo of a table lookup player making a move



```
t420@t420-ThinkPad-T420: ~
game 3221: The winner is . . . TABLE-LOOKUP-PLAYER
game 3222: The winner is . . . TABLE-LOOKUP-PLAYER
game 3223: The winner is . . . TABLE-LOOKUP-PLAYER
game 3224: The winner is . . . TABLE-LOOKUP-PLAYER
game 3225: The winner is . . . TABLE-LOOKUP-PLAYER
game 3226: The winner is . . . TABLE-LOOKUP-PLAYER
game 3227: The winner is . . . TABLE-LOOKUP-PLAYER
game 3228: The winner is . . . RANDOM-MACHINE
game 3229: The winner is . . . TABLE-LOOKUP-PLAYER
game 3230: The winner is . . . TABLE-LOOKUP-PLAYER
Player Switch
Writing gst to gst_4x4.txt
game 3231: The winner is . . . TABLE-LOOKUP-PLAYER
game 3232: The winner is . . . TABLE-LOOKUP-PLAYER
game 3233: The winner is . . . TABLE-LOOKUP-PLAYER
game 3234: The winner is . . . TABLE-LOOKUP-PLAYER
game 3235: The winner is . . . TABLE-LOOKUP-PLAYER
game 3236: The winner is . . . TABLE-LOOKUP-PLAYER
game 3237: The winner is . . . TABLE-LOOKUP-PLAYER
game 3238: The winner is . . . TABLE-LOOKUP-PLAYER
game 3239: The winner is . . . TABLE-LOOKUP-PLAYER
game 3240: The winner is . . . TABLE-LOOKUP-PLAYER
Player Switch
Writing gst to gst_4x4.txt
game 3241: The winner is . . . TABLE-LOOKUP-PLAYER
game 3242: The winner is . . . TABLE-LOOKUP-PLAYER
game 3243: The winner is . . . RANDOM-MACHINE
game 3244: The winner is . . . TABLE-LOOKUP-PLAYER
game 3245: The winner is . . . TABLE-LOOKUP-PLAYER
game 3246: The winner is . . . TABLE-LOOKUP-PLAYER
game 3247: The winner is . . . TABLE-LOOKUP-PLAYER
game 3248: The winner is . . . RANDOM-MACHINE
game 3249: The winner is . . . RANDOM-MACHINE
game 3250: The winner is . . . TABLE-LOOKUP-PLAYER
Player Switch
Writing gst to gst_4x4.txt
game 3251: The winner is . . . RANDOM-MACHINE
game 3252: The winner is . . . TABLE-LOOKUP-PLAYER
game 3253: _
```

```
t420@t420-ThinkPad-T420: ~                                    En  10:20 PM
[36]> ( setf *learning-rate* .0001 )
1.0E-4
[37]> ( train-neural-network )
Neural Netrwork params?(list): ( 9 15 7 1 )
File to write to?(quoted): "nn_9-15-7-1.txt"
Would you like to seed the population?(y/n): y
With what individual file?(quoted): "nn_9-15-7-1.txt"
best individual:
(0.53127795 0.0577194 -0.58367175 -0.5819582 -0.343408 0.6823474 -0.9732913 -0.46652052 0.52027 -0.18493654 -0.62977886 -0.7120009 -0.21545123
 -0.12723698 -0.8149613 0.48550394 0.7998334 0.43790784 0.49686083 -0.83801824 -0.19915244 0.8114973 0.9706232 0.9738952 -0.15071933
 -0.33498043 0.03783269 0.9435154 -0.38935897 0.58600056 0.79137444 0.6219781 -0.23749089 0.23231854 -0.36093754 -0.50842357 -0.45676968
 -0.94283146 -0.16780595 -0.66349715 0.7478163 -0.87944126 0.4857634 -0.24648654 0.1203204 -0.37036303 -0.07974421 -0.53481233 -0.09015094
 -0.013424622 -0.46329013 -0.24082544 -0.8371158 -0.15499541 0.22193031 -0.66326374 -0.78462005 0.12739041 -0.34127796 -0.062243246 -0.8437169
 0.10493677 0.08194622 -0.42630753 0.49291602 -0.73734444 -0.08831012 -0.38283262 -0.120599784 0.10561488 0.483702 -0.846985 -0.6790213
 0.83242714 0.8868024 0.73244816 -0.4545962 0.42678195 -0.49579605 0.8212899 -0.62056947 -0.741639 0.9712922 -0.62260735 -0.61412936 -0.743376
 0.64806074 -0.67499286 -0.8936589 0.20710227 -0.9433192 -0.6033807 0.7570571 0.5301996 -0.2686062 -0.4384884 0.7689285 -0.21592332 0.49821025
 -0.3338357 0.17393437 0.3594805 0.7696746 -0.28622586 0.42271093 -0.52071655 -0.665081 0.28059518 -0.8245095 -0.020374488 0.9 -0.7652862
 0.4840733 -0.90185285 -1.0 0.26307356 -0.23126364 0.09847542 0.918309 -0.07533143 -0.2235604 -0.6564952 1 0.33195403 -0.6988092 -0.36139238
 -0.51430595 0.20684783 0.8244501 -0.018484622 0.999 -0.7984673 0.8789572 0.3174586 -0.124070644 -0.029833958 -0.6343363 -0.92113656
 -0.90099996 -0.07503643 0.75926745 -0.6286331 0.038435742 -0.16662106 -0.989 0.036013942 -0.9662853 0.008849239 -0.999 -0.7716586 0.116527386
 -0.8456131 -0.5503297 0.48073554 0.36087796 0.119920805 -0.68453574 0.25553766 0.44175833 0.97021395 0.028738903 0.24988522 -0.52340245
 -0.04379133 -0.22731243 -0.036610335 -0.27870148 0.20551899 -0.889 -0.32808262 -0.041490525 0.09234358 -0.29736573 0.96475106 -0.0016592741
 0.3486468 -0.61860037 -0.31927872 -0.04389716 0.34153745 0.48701102 -0.40538192 0.6489393 0.5779468 0.8985655 0.38133818 -0.5648578
 -0.34423208 0.6852438 -0.11464003 -0.43524975 0.38597906 -0.56631345 0.06997162 0.71610004 0.06905045 0.6454172 -0.99 -0.46365523 0.8668277
 -0.35919446 0.81508034 -0.77682155 0.49461302 -0.05590102 0.23418504 0.4274057 0.75231117 0.51885533 -0.25757456 -0.72994816 -0.2457818
 -0.86453646 0.21924435 -0.76295096 0.8546508 -0.981 0.46872988 -0.49998325 -0.89589524 -0.11169778 -0.06724852 -0.002546994 0.015161775
 0.751043 0.08014269 -0.521761 -0.520555 -0.70877415 0.75389737 -0.3331094 0.7065273 -0.35226405 0.46534818 -0.1565825 -0.0025284886 0.68188363
 0.90221703 0.9865991 0.96021956 -0.78228456 -0.34916234 -0.93089014 -0.25332904 -0.38463265 0.51410615 0.9389642 0.991 -0.70967126 -0.6488116
 -0.6392595 -0.2200362 -0.605732 0.59501904 -0.2472396 0.42046997 0.727682 -0.80091196 -0.13224505 -0.5569189 0.5012838 0.19826767 -0.079143405
 -0.7180074 0.061567474 0.30936003 -0.6911681 0.66342384 0.07244996 0)
cost of best individual: 15.645172

average fitness of population 1 = 16.195118
cost of best individual: 15.64513
 average fitness of population 2 = 16.345037
cost of best individual: 15.645114
 average fitness of population 3 = 16.145117
cost of best individual: 15.645103
 average fitness of population 4 = 15.845116
```

(a) A training game state table. (b) A training neural network

6 Reflections and Conclusions

The initial plan for this project was to use the game state table to find the empirical values of winning from any game state in Dobo, then use the neural network as a way of compressing the encapsulation of this knowledge. I was surprised when I learned that the way I was generating the probability values was actually a method of machine learning called Rote learning, which can be classified as a reinforcement learning method. The initial plan also included the use of the BPGA approach for training, however after the problem was modeled using Tensor Flow [14] and a solution was not found using gradient descent, it was decided to not try and implement gradient descent in the use of my program because I do not believe that my method of gradient descent would come even close the power that Tensor Flow has built into its amazing, up-to-date libraries. The power of Tensor Flow can be seen on the Tensor Flow playground [2].

The final neural net (being the best individual in the last generation of the genetic algorithm) did not fit the data well enough to be a justifiably good player. The final neural net had a retention rate of around 80%, which is a pretty good retention rate however in order to be a good player the retention rate would have to be a lot closer to 100%, as the normalized data set player could only achieve around a 94% win rate against a random player as is.

I believe that I have accomplished quite a lot in this project. It may have worked out better and been a good idea to try different activation functions (some very easily accessible in CLISP [6]). I know that is must be possible in some way to model a network in such a way that it would learn all classifications (my believe comes from some convincing from [8]). Although I did not think I was totally successful I have learned a lot and got introduced to a myriad of machine learning techniques during my semester of study, and I cannot wait to continue my study throughout my undergraduate years and eventually in graduate school.

7 WORKS CITED

[1] Bhatt, Anurag, Pratul Varshney, and Kalyanmoy Deb. Evolution of No-loss Strategies for the Game of Tic-Tac-Toe. Rep. no. 2007002. N.p., Jan. 2007. Web. 12 Feb. 2017.

[2] Carter, Daniel Smilkov and Shan. "Tensorflow - Neural Network Playground." A Neural Network Playground. TensorFlow, n.d. Web. 28 Feb. 2017.

[3] Ceravolo F., De Felice M., Pizzuti S. (2009) Combining Back-Propagation and Genetic Algorithms to Train Neural Networks for Ambient Temperature Modeling in Italy. In: Giacobini M. et al. (eds) Applications of Evolutionary Computing. EvoWorkshops 2009. Lecture Notes in Computer Science, vol 5484. Springer, Berlin, Heidelberg

[4] Dikran Marsupial. "Can a Neural Network Output Represent a Posterior Probability?"Likelihood - Can a Neural Network Output Represent a Posterior Probability? - Cross Validated. Stack Exchange, 25 Oct. 2012. Web. 30 Apr. 2017.

[5] Harris, David J. "Danger of Setting All Initial Weights to Zero in Backpropagation." Neural Networks - Danger of Setting All Initial Weights to Zero in Backpropagation - Cross Validated. Stack Exchange, 26 Apr. 2012. Web. 30 Apr. 2017 . [6] "Lisp Activation Funcitons." Function SINH, COSH, TANH, ASINH, ACOSH, ATANH. LispWorks, n.d. Web. 20 Mar. 2017.

[7] LeCun, Yann, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Muller. Efficient BackPop. Tech. N.p.: n.p., n.d. Print.

[8] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

[9] Miller, Steven. "Steven Miller." Mind: How to Build a Neural Network (Part One). N.p., 10 Aug. 2015. Web. 30 Apr. 2017.

[10] Montana, David J., and Lawrence Davis. Training Feedforward Neural Networks Using Genetic Algorithms. Tech. N.p.: n.p., n.d. Print . [11] Rohrer, Brandon. "How Deep Neural Networks Work." YouTube. YouTube, 02 Mar. 2017. Web. 5 Apr. 2017.

[12] Shilov, Georgi E. (1977), Silverman, Richard A., ed., Linear Algebra, Dover, ISBN 0-486-63518

[13] Siegel, Sebastian. Training an Artificial Neural Network to Play Tic-tac-toe. Rep. N.p., 20 Dec. 2001. Web. 9 Feb. 2017.

[14] "TensorFlow API." TensorFlow API. Google, 26 Apr. 2017. Web. 15 Apr. 2017.

[15] Whitely, D. Genetic Algorithms and Neural Networks. Tech. N.p.: n.p., n.d. Print