Using Artificial Neural Networks to Play the Game of Strategic Tic Tac Toe

New Mexico Supercomputing Challenge

Final Report

Portales High School Team 1

Nick Brown and Christopher Leap

Table of Contents

Executive Summary **2**

1. Introduction **3**

    1.1. Artificial Neural Networks **3**

    1.2. Strategic Tic Tac Toe **4**

2. Methods **6**

    2.1. Making the Tic Tac Toe Game **6**

    2.2. Structure of the Artificial Neural Network **7**

    2.3. Getting the Network to Play a Game **8**

    2.4. Training the Artificial Neural Network **8**

3. Results **12**

4. Conclusions **13**

Executive Summary

Artificial neural networks are at the cutting edge of artificial intelligence. However, these networks require training. So, to effectively utilize artificial neural networks, we must learn how to quickly and effectively train these networks. Through our experimentation, we discovered that certain training methods work better than others throughout different stages of training and development. We plan to continue developing and refining the training process of artificial neural networks to allow them to further solve the world's problems.

1. Introduction

Artificial neural networks (ANNs) are at the forefront of computing due to their flexibility in problem solving complex problems through pattern recognition. Because ANNs have earned importance in the world of computing, the methods required to create neural networks and train them to perform specific tasks is consequently important. We have experimented with training ANNs to play a game, Strategic Tic Tac Toe, due to its simple rules, strategically deep gameplay, and lack of hidden information or chance.

1.1 Artificial Neural Networks

Artificial neural networks are modeled off of the brain and its naturally occurring biological neural networks.[1] They are a collection of nodes connected with varying strengths to each other. Each node can send signals to nodes connected to them, but due to the varying strength of connections, some nodes are more effective at influencing others. ANN act in the same way as a human brain, allowing them to recognise patterns and robustly make decisions.[2] Instead of needing to explicitly tell the computer to give which output when given which input, ANNs can learn to recognize when a pattern emerges by training against data sets or through other various training methods.

Deepmind's AlphaGo team utilized an ANN to build a computer to play against a professional Go player. Go used to be infamous for being the ultimate challenge for artificial intelligence. The game allowed for billions of possible moves, making recursive tree searches near impossible. That is, the amount of possible moves that could be made, the amount of possible moves that could be made after that, and so on added up to too large of a number for a modern supercomputer to be able to sift through.

So, using minimax or other brute force methods would not work. The team could not feasibly program specific strategies into the computer either, though. Doing so would require the team to develop superior strategies to whichever opponent they planned to beat (eventually the Go master

[1] Chris Stergiou, "What is a Neural Network?", https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/cs11/Article1.html (March 28, 2017)

[2] Chris Woodford, "How Neural Networks Work - And Introduction" (2017), http://www.explainthatstuff.com/Introduction-to-neural-networks.html (March 28, 2017)

Lee Sodol), which is unlikely. In addition, strategies shift and change throughout time, so any static strategy would not work for long until the team would need to reprogram another strategy.

However, by using ANNs, the AlphaGo team trained their computer to recognize the patterns of expert players, mimicking the patterns, and later using that knowledge to beat Lee Sedol, the world's top Go player.[3] The ANN was able to learn and shift in its strategies alongside the expert players, allowing it to later beat them.

Being able to train ANNs is relevant because they can learn to automate tasks that could seem difficult to solve through programming with other methods.. Knowing the best training techniques for ANNs is therefore important.

1.2 Strategic Tic Tac Toe

Strategic Tic Tac Toe is a variation on the game of, as one may guess, Tic Tac Toe. We originally found this game on the website Coolmath-Games.[4] The game consists of nine Tic Tac Toe boards, each placed throughout the nine spaces in a larger Tic Tac Toe board. (See Figure 1.2a)

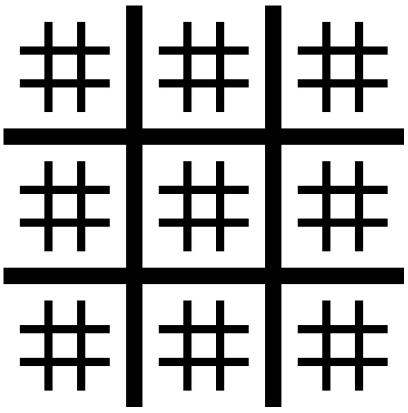

Figure 1.2a An empty board for Ultimate Tic Tac Toe

The game is played, with both players taking turns, by placing pieces in the smaller boards. The space on which each player places their piece determines which smaller board their opponent will have to place on in the subsequent turn. For example, if a player places in the middle space of one of the smaller boards, their opponent will be forced to place their piece on the middle board. (See Figure 1.2b)

[3] DeepMind, "AlphaGo" https://deepmind.com/research/alphago/ (March 28, 2017)

[4] Cool Math Games, http://www.coolmath-games.com/0-strategic-tic-tac-toe

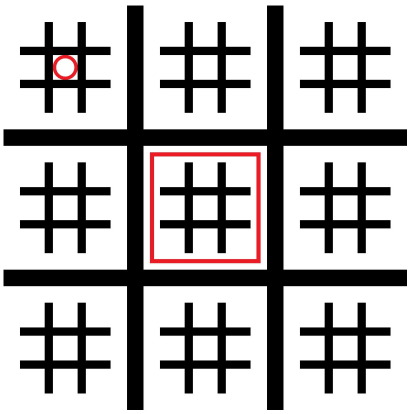

Figure 1.2b The first player made their move in the middle of the upper-left board, so their opponent must place on the middle board

The only exception to this rule occurs on the first turn, since no player has made any moves yet, or when the board corresponding to the previous player's move has already been won or tied. In these situations, the player will be able to choose which board they want to place their piece on. Normal Tic Tac Toe rules apply; to win a board, either a diagonal or straight line of length three must be occupied by only one player. To win a game of Strategic Tic Tac Toe, one player must win three small boards in a diagonal or straight line.

We chose this game for two reasons: it is strategically complex while still being relatively simple, and it is easy for new players to learn. Many people are familiar with Tic Tac Toe and have played it multiple times. However, after a certain number of games, Tic Tac Toe becomes too easy to play; most games end in ties when two moderately competent players go head-to-head. Strategic Tic Tac Toe adds complexity to the game and widens the skill gap. While being complex, Strategic Tic Tac Toe remains accessible to new players and has simple rules. Since many people are familiar with Tic Tac Toe, they can easily pick up the few additions to the rules and start playing quickly.

This is useful when testing the strength of the ANN against normal people. The simple rules are useful when programming the actual game and prevents the ANN from having to pick between different types of moves, different objects to move, and the implications of those actions. Taking these types of choices away from the ANN further simplifies the game provided for the ANN.

We wish not to use the ANN to play a complex game. We plan to study the methods to train the ANN to play a game. So, we plan to develop the methods on a smaller scale so that we can later scale up the methods so they will work on larger scale, more complex games. As Abraham

Lincoln once said, “given five minutes to chop down the biggest tree… ...I would spend three of those five minutes sharpening the axe.”[5] So, we plan to sharpen the axe with which one may chop down larger trees.

2. Methods

2.1 Making the Tic Tac Toe Game

The game of Strategic Tic Tac Toe mainly consists of a board and the rules and restrictions imposed upon moves that players make.

A board has to have spaces where pieces are placed, allow players to place those pieces, and allow players to view information about the board, such as the moves that can be made, the player currently controlling the board, and the pieces placed on each tile of the board. With all of these properties the board is able to properly function. All of this allows for a small three-by-three board to be formed, but a larger board is unable to be formed, since each board can only contain tiles, not other boards. To create a larger board that contains other boards, ten boards are created. Nine of these boards are contained in a list and when a move is made in the game, that move is made on the corresponding board in the game. This allows the boards to be stored and moves made on them. To handle the properties of the overall board, whenever a smaller board is won in the list of boards, the same move is made on the larger board not in the list. This allows both the individual boards’ qualities to be preserved but to also have use of the larger board.

To enforce the rules of the game, most of the restrictions are imposed at the time that the player tries to make moves. This allows the game to prevent the player from placing their piece in a non-empty space or on a board that is either already won or cannot be placed upon due to the opponent's previous move. Other rules are imposed after moves have been made to check, for example, if the game has been won and if so prevent the game from continuing, but overwhelmingly most of the rules are imposed to prevent players from making illegal moves.

These two parts combine to create a game that can be played without players having to enforce rules. This allows the game to deny players from trying to cheat and generally

[5] Retrieved from http://quoteinvestigator.com/2014/03/29/sharp-axe/

preventing ANNs that make illegal moves from working.

2.2 Structure of the Artificial Neural Network

The Artificial Network can be broken down into two main components: nodes and connections. A node needs to be able to do three tasks: collect input, process the input, and output the result. The connection is the pathway by which the nodes send input and output. These two components form the structure of the network (see Figure 2.2a), allow for data to pass through it, and make the network trainable.

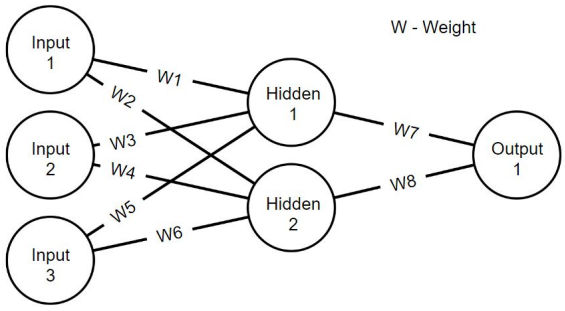

Figure 2.2a An example of a neural network with three input nodes, one hidden layer with two nodes, and one output node

The networks used in our program arrange nodes into layers, and the connections only link nodes in adjacent layers. These layers are categorized as either input layers, hidden layers, or output layers.

The input layers are the first layer that receives the input data for the network. After collecting and processing the data, each node in the input layer sends the processed data to each node in the next layer.

Next, the first hidden layer collects the data from the input layer, process it, and either sends the data to the next hidden layer, if there is one, or sends the data to the output layer. The number of hidden layers depends on the network, and the data keeps getting passed on from one to the next until each one has processed the data. After this, the data gets sent to the output layer.

The output layer collects input from the previous layer, processes it, and then displays the processed data.

Each connection between nodes carries a weight. That is, the output from one node to another may not have as much impact on the input as the output from another node. If the weights are adjusted, the way in which the network processes inputs is changed. By taking control of the weight adjustment, the

network can be trained to handle specific problems.

2.3 Getting the Network to Play a Game

ANN are designed to take inputs and produce outputs. So, to make an ANN play Strategic Tic Tac Toe, the inputs and outputs have to be tailored to work with Strategic Tic Tac Toe.

The ANNs' input is the state of each tile on the Tic Tac Toe boards after one of the possible moves is made; the ANN processes its inputs and outputs the score of that board, using the trained weights of each connection to calculate it. The score of the move is then compared to the score of the next move, until all moves have been scored, compared, and the ANN has found the best move that it thinks it should make, given its training.

2.4 Training the Artificial Neural Network

Two components of the training of ANNs are important: the opponents that the ANNs train against and the way that the ANNs' performances are assessed. We have determined multiple types of opponents at different stages in an ANN's development that may be the most effective at training the ANN. These opponents may include humans, artificial intelligence developed to play the game, minimax, artificial intelligence that randomly chooses moves, and other ANNs.

Humans are able to easily adapt their play styles and competitiveness to better train the ANNs. However, humans are very variable in their skill level in any given game; do not quickly process board states at the speed required and make moves; and unless extremely talented at playing the particular game, cannot adjust their skill level to almost any level. Using humans to train the ANNs also presents the problem of creating a scoring system for the ANNs. Since humans cannot play multiple games in mere seconds, more weight is given to each game that the human plays. A smaller sample size of games coupled with human's high variability makes scoring for ANNs less objective and more variable. This variability can cause training ANNs to take longer and yield worse results, since not every point of data collected is as accurate and there are less points overall. However, humans periodically playing against the ANNs can allow the people trying to train the ANNs to

record results of training and tailor the gene pool of the ANNs.

Using an artificial intelligence that uses minimax to train the ANNs would be a good way to train ANNs since minimax is a strategy that relies upon perfect information, each player knows exactly what each other player can do and what each of those actions will result in. That is, there is no hidden information or chance involved in the game. Minimax works by looking at the moves that its opponent can take, then the moves it can take, then the moves that its opponent can take ad nauseum, or until told to stop, then determines which move will have the least detrimental or most positive effect on its chance of winning, by scoring the board's state found at the end of the process. Strategic Tic Tac Toe is a game of perfect information. This allows a minimax AI to play almost perfectly or play at a low skill level. This controlled variance of skill levels allows a smaller number of games to be played between the AI and ANNs. This allows results to be quickly generated. However, minimax AIs plateau in skill at a certain point since, once the AI looks into the future a couple of turns, the number of moves seen is restrictively large, slowing down the AI greatly. In addition, if the game which the ANN is being trained to play has a game tree as large as Go, or even half of it, a minimax-oriented AI will not be of as much use to the ANN.

An algorithm that randomly chooses moves has high variance in its skill level. Because of this, it tends to be very bad at games, as it has no discernible strategy. A random AI, however, is very fast at choosing moves; it only has to perform a random calculation. This may make the AI very good for training the early iterations of ANNs, when the ANNs are very bad at the game, but need to have a large number of iterations before they can advance to a higher skill level.

Training ANNs against each other allows the ANNs' opponents to have a similar skill level as them, quickly play a game, and even doubles up on games played by ANNs, instead of one ANN playing in a game, two play in one game. However, the ANNs can deviate, from randomness introduced in the creation of new ANNs, from the task at hand, partially or completely destroying the advances they have made. Since there is no standard that the ANNs have to base their performance on, except for one another,

these deviations from the course can be extensive. Pitting ANNs against each other also creates a much less black and white rating system, where a ANN either wins against their opponent, ties with their opponent, or loses to their opponent. Instead, since each ANN plays against a large number of ANNs, a relative ranking system can be created.

If ANNs play against each other, a system or tournament structure must be created so that the ANNs can obtain a relative ranking system to each other. Tournament structures that are slower and in which more games are played will have the best player be victorious more often than faster tournament structures with fewer games. Several tournament structures exist: swiss, elimination, round-robin, and several other more complex structures with few merits. In elimination tournaments, after a player loses a certain number of times, they are kicked from the tournament. In single elimination tournaments, players are allowed only one loss; in double elimination players are allowed two losses; and so on and so forth.[6] This tournament structure has high variability, since a good player could cluster their losses at the beginning of a tournament, while a bad player could cluster all of their losses to the end of the tournament, but the bad player would seem to be a better player than the good player and rank higher. Elimination is however very fast since the brackets quickly shrink, allowing the tournament to quickly finish.

Figure 2.4a An example of a single elimination tournament structure.[7]

Round-robin tournaments, on the other hand, see each player playing each other player, then determines a ranking for each player by the number of wins and losses they have accrued. This tournament structure is slower than elimination

[6] Print Your Brackets http://www.printyourbrackets.com/singleelimination.html

[7] Retrieved from https://www.printyourbrackets.com/single-elimination-tournament-brackets.html

tournaments, and since round-robin pits each player against each other player[8], the number of games played grows exponentially with the number of players. This tournament structure is much less variable than the elimination structure. However, with just 8 players the number of games played is 28, climbing to 120 games with just 16 players.

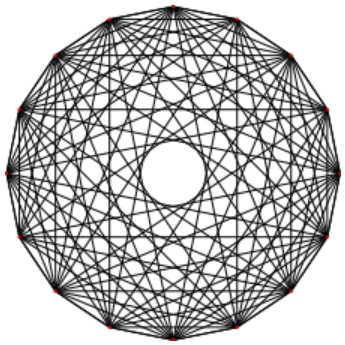

Figure 2.4b An example of a round robin tournament structure with dots representing players and connections representing games.[9]

The swiss tournament structure plays less games than the round-robin structure while having less variability than the elimination structure. Swiss works by playing all opponents with the same record against each other, in each round. If the number of people with a certain record does not allow for perfect matchmaking, the players with the two closest records will play against each other.[10] This tournament structure prefers to have its number of players be a power of two, since the tournament ends when a sufficient number of rounds has been played to determine who is the best player, the power that two is raised to to get closest to the number of players participating in the tournament. This allows for one player to have no losses, assuming no one ties, which determines a clear winner, then each player is ranked by how many wins they have, then how well they did against their opponents, then the skill level of the opponents that they faced. All of this generates a tournament structure where each match played is tailored so that the a large amount of possible information is gained, and so that the tournament is not extremely long. Swiss gives the best of both the elimination structure and the round robin structure, and so it is what we have decided to use.

These training methods are utilized in a generation-ranking system. Multiple ANNs are randomly generated in what is called a generation. The members of this generation

[8] Print Your Brackets. Retrieved from http://www.printyourbrackets.com/roundrobin.html

[9] Retrieved from https://commowikimedia.org/wiki/File:Cross_graph_8_Nodes_highlighted.svgns.

[10] The Spruce https://www.thespruce.com/the-swiss-system-611537

compete in the training methods mentioned above, such as a head-to-head tournament, playing matches against a random-move-selecting opponent, or playing against a developed AI. After training the generation, the members of the generation are ranked. A new generation is then generated from the members of the previous, ranked generation. The top quartile of the previous generation are put into the next generation, and the remaining slots are filled by "breeding" the higher ranked members of the previous generation. Breeding two ANNs involves combining the weights of their connections with some level of randomness and variability. For each weight in the "child" network, there is a 40% chance that the child will inherit the weight from one parent, 40% chance that the child will inherit the weight from the other parent, and 20% chance the child will mutate and gain a completely random weight. In this way, the new generation will include the best members of the previous generation and some new, "evolved" members that are possibly better than the previously best. By doing this, each generation seeks to improve on the previous generation.

3. Results

In the early stages of training the ANNs against each other, the ANNs were unable to win while playing if they played first. This invalidates the training, since the ANN that is selected as the best and who subsequent generations are modeled after is picked semi-randomly. The games that the ANNs play in this situation are identical. The ANN takes the first move suggested. This is all caused by an algorithm for choosing the best move The first move suggested is first set as the best move, then the subsequent moves are compared against it, and only when a move is better than the best move is that move called the new best. This introduces a minimal preference to the first move suggested to the computer. However, if chance was introduced to the decision when the scores were equal, this preference to the first move suggested would be eliminated. If a coin is flipped to determine whether or not a new move, with equivalent score to the old move, will replace the old move, the chosen move will tend to be the last suggested move, since the last move has to only win one coin flip, while the first has to win a coin flip for every suggested move. This makes the chosen moves tend to be the last

suggested move, but since probability is introduced, the degree by which the last move is favored is much less than that of the previous method.

Moves: Second → First ↓	ANN VS. ANN	ANN VS. Random	Half-and-Half
ANN VS. ANN		TIE	Half-and-Half
ANN VS. Random	ANN VS. ANN		ANN VS. Random
Half-and-Half	Half-and-Half	ANN VS. Random	

Table 3a. A representation of how well each training procedure compares to the others when trained for six generations.

The different training methods were compared by training ANNs with a certain method, then training other ANNs with a different training method, until all training methods had been used, then playing the resulting ANNs against each other. This should, in theory provide a clear training method that trains the ANNs most quickly and effectively. Instead of a clear winner emerging, the ANNs trained with different training methods beat each other in a rock-paper-scissors fashion, as seen in table 3a.

4. Conclusions and Future Plans

The slight bias produced by the way that the program selected its moves suggests that the ANN is very weak at playing the game in the beginning, and each network plays with the same strategy. We hypothesize that the weights are not yet tuned to each other at this stage, so each one reaches the same strategy, similar to many people learning to do a backflip; each person may be doing completely different things with their muscles, but until they learn to use their muscles in harmony, each person ends up on the ground.

The negative effect of each player utilizing the exact same strategy, is that each game is identical. This prevents generations from advancing because no number of generations can get the ANN out of the early stages of development. This is obviously negative, since the ANN will never learn and training it is therefore useless.

By making the choosing technique more random, the ANNs can overcome the second

player advantage and add diversity to each player's strategy.

We will also improve upon the program's handling of moves with similar scores. If for instance, the list of moves that we are giving to the ANN were first shuffled, this would eliminate the issue of favoring moves that either come first or last in the list of suggested moves.

The rock-paper-scissors matchups of the different training techniques may either be luck, this may be due to the slightly random nature of the ANNs, or these matchups may be caused by a deeper more telling reason. However, the consistency with which the same matchups were won by the same people, except when second player advantage caused the two ANNs to tie, suggests that the matchups between the ANNs trained in the various ways are fairly consistent. Sufficient investigation into the reasons behind the way that the matchups play out has not been performed, but will be carried out in the future.

In the future we will optimize the program so that it runs more quickly. This will allow us to add more layers of nodes in the ANNs, making their decisions more complex, play more tournaments to train the ANNs, and create more sophisticated AIs for the ANNs to play against. All of this will allow us to further explore how to best train the ANNs.