

University of Arkansas, Fayetteville
ScholarWorks@UARK

Computer Science and Computer Engineering
Undergraduate Honors Theses

Computer Science and Computer Engineering

5-2015

Using Genetic Learning in Weight-Based Game AI

Dylan Anthony Kordsmeier
University of Arkansas, Fayetteville

Follow this and additional works at: <http://scholarworks.uark.edu/csceuh>

 Part of the [Artificial Intelligence and Robotics Commons](#)

Recommended Citation

Kordsmeier, Dylan Anthony, "Using Genetic Learning in Weight-Based Game AI" (2015). *Computer Science and Computer Engineering Undergraduate Honors Theses*. 32.
<http://scholarworks.uark.edu/csceuh/32>

This Thesis is brought to you for free and open access by the Computer Science and Computer Engineering at ScholarWorks@UARK. It has been accepted for inclusion in Computer Science and Computer Engineering Undergraduate Honors Theses by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu.

Using Genetic Learning with Weight-Based Game AI

An Undergraduate Honors College Thesis

in the

Department of Computer Science
College of Engineering
University of Arkansas
Fayetteville, AR

by

Dylan A. Kordsmeier

Table of Contents

1. Introduction.....	3
1.1 The Evolution of Computer Game Artificial Intelligence.....	4
1.2 Problem Statement and Research Question.....	5
2. Background and Related Work.....	6
2.1 Weight-Based Artificial Intelligence in Games.....	6
2.2 Machine Learning in Games.....	7
3. System Design and Implementation.....	8
3.1 The Connect 4 Game Implementation.....	8
3.2 Genetic Algorithm for Learning.....	11
3.3 Evaluation Framework.....	12
4. Evaluation.....	13
4.1 Brute Force Experiments.....	13
4.2 Single Population Training.....	15
4.3 Dual Population Training.....	18
5. Conclusions and Future Work.....	20

Abstract

Human beings have been playing games for centuries, and over time, mankind has learned how to excel at these fun competitions. With the ever-growing interest in the field of Machine Learning and Artificial Intelligence (AI), developers have been finding ways to let the game compete against the player much like another human would. While there are many approaches to humanlike learning in machines, this article will focus on using Evolutionary Optimization as a method to develop an AI to effectively play the Connect Four game.

1. Introduction

In the modern era, games are becoming a pervasive part of society at large. As games are studied by many more institutions, research has proven that better artificial intelligence agents in a game will allow for more re-playability (Du, 2009). The solution that some researchers have found is to have a Machine Learning algorithm create a dynamic intelligence. One algorithm in the Machine Learning field is Evolutionary Optimization, or Genetic Learning. This consists of initializing a population within the confines of the problem, using a selection technique to advance the population in a positive trend, and promoting diversity in the population to cover a wider range of potential solutions (Gashler, 2014). However, does a complex algorithm really beat a simple brute-force algorithm, such as depth-first search? For one, these algorithms are expensive in runtime. A depth-first search can take much too long for the average game player to wait between moves. What happens when predicting its opponent's next move is too unrealistic for an algorithm? This paper will attempt to prove a game program using Machine Learning can find the most competitive game strategy using a set of weights that control the artificial intelligence where an exhaustive breadth-first search is not applicable. I will test this hypothesis

by comparing the results of several different tests from a Machine Learning standpoint and different sets of weights that have proven themselves against competitors.

1.1 The Evolution of Computer Game Artificial Intelligence

While games have been around for as long as man can remember, they were only for human beings to play. It would have been hard for the people to imagine having some other entity that can pretend to be a human to play against. Though the idea of some level of artificial intelligence had been around from Greek mythology and into the Middle Ages (McCorduck, 2004), it was not deemed a reality until technology advanced quite a bit further. It was at the Dartmouth conference of 1956, that the idea of an "Artificial Intelligence" was born (Crevier, 1993). This was after Alan Turing's paper that speculated about the possibility of machines that think (McCorduck, 2004). This paper set the groundwork for the conference that brought together the main players in the field of Artificial Intelligence's future. In as little as fifteen years, this led to the development of programs that could speak English, solve mathematical word problems, or prove geometric theorems (Russell, 2003). This excited many investors who would pour large amounts of money into the booming field. However, results started to slow as the new terrain had become ventured. This led to a sort of funding drought in the AI field, because investors needed new, sufficient answers. Because of limited computer power, frame and qualification problems, and the inability to reflect commonsense knowledge and reasoning, AI was mostly shut off commercially (NRC, 1999). Then, the rise of expert systems came about in 1980, and started another Artificial Intelligence boom. With it, came the first attempt to conquer the commonsense knowledge problem directly. Douglas Lenat, the head of the project called Cyc, states, "The only way for machines to know the meaning of human concepts is to teach them, one concept at a time, by hand" (Lenat, 1989). This meant he would create an

expansive database that covers all information a person knows. A version of this database called ResearchCyc was released in 2006 for AI researchers (Ramachandran, 2005). This dedication is what kept the field of Artificial Intelligence alive.

The specific type of learning I am focusing on is called genetic learning. This field's genetic algorithms were conceived in the 1960s by John Holland. His intent was to study adaptation in nature and figure out how to implement that same phenomenon in computer systems. He had many chromosomes that were made up of multiple genes which were made up of multiple alleles. These chromosomes were pitted against each other with a method that determines fitness, and then they would reproduce. The chromosomes with the highest fitness score would reproduce more offspring than the worse chromosomes. The end goal was to use a natural selection style learning to create the most fit chromosome. This main idea is still carried out today along with his methods of altering the population. He first implemented the ideas of crossover, inversion, and mutation into a learning algorithm (Mitchell, 1999).

Today, a large amount of research for artificial intelligence and genetic learning is done for games of all kinds. As games are studied by many more institutions, research has proven that better artificial intelligence agents in a game will allow for more re-playability and an overall better gaming experience (Du, 2009).

1.2 Problem Statement and Research Question

There are many different ways to approach artificial intelligence in a game. A designer can use brute force look-ahead to pick the best move out of all possible combinations, or they can simply write some rules that can mimic humanlike behavior. Say for chess, the first rule would be never move into check. These different approaches allow for certain flexibility in learning style. Each approach will offer better results to some problem and worse results that are

not necessarily bad to others. The brute force look-ahead will give the most optimal results, but all calculations will be performed during the actual run of the game. This adds a high-cost to the run and can slow the entire system down. A rule-based algorithm takes a knowledge database that gives a system “real human knowledge” to play a game. This will make the game more lifelike but predictable after several plays. The weight-based game will allow for an easy to write and very computationally fast system, but it will also become predictable.

I picked the weight-based artificial intelligence for my research, because play selection in Connect Four can be controlled by seven weights. These weights are used to determine the relative value of different moves based on the number of tokens the player and the computer have placed in a sequence of four board positions. This approach will hopefully prove whether or not there is a weight-based artificial intelligence system that can effectively play a game of Connect Four with a user.

2. Background and Related Work

2.1 Weight-Based Artificial Intelligence in Games

Weight-based artificial intelligence is similar to rule-based, except it focuses on scoring moves instead of replicating human actions. For example, in a game of Checkers the implementation could have a weight for jumping the opponent’s piece, avoiding a jump, setting up for a jump, etc. In Checkers it is perfectly acceptable to base a move on the score alone because the objective is more straightforward. Take the game Reversi as a counterexample. The object of the game is to have more tokens of your color than your opponents; however, the best strategy involves having fewer tokens until the middle of the game. Because weights would cause a general trend of gaining tokens, a weight-based artificial intelligence would not be as

effective (Millington, 2009). Checkers on the other hand can be implemented very successfully with weight-based gaming. While a majority of the time the weights will be set by hand for a certain competition, these weights can also be automatically set. This is the approach the team behind Anaconda Checkers used as well. They also used a genetic learning algorithm to update a set of weights, and they were successful in the commercial checkers world. “The result was a Checkers player that beat a commercially available Checkers program 6-0” (Jones, 2009).

2.2 Machine Learning in Games

This idea of having an artificial intelligence agent figure out how a game should be played by itself was not always the case. It has been more of a recent understanding that this can greatly enhance the game. 1986 was the year that the *Machine Learning* journal saw its first issues, after the field started to grow outside of artificial intelligence (Langley, 2011). Machine Learning works in many ways now that significant research has been completed. It works from teaching a chess game every nuance of strategy to generating a strategy catered directly to a player. The ability to recognize the strategy of an opposing player is called opponent modeling. The strategy is usually the general play style of the opponent. This categorization allows the machine to figure out the type of strategy being played and its counter (Schadd, 2007). A more mainstream version of this kind of learning is in development. This game called *Nevermind* uses biofeedback from the user to manipulate the game environment. That is, it learns what makes the user biologically afraid or stressed and pits it against them on the screen (www.nevermindgame.com). This kind of technology is much more involved than the utilization of weights.

Relying on weights to understand strategy has already been explained, but how does the developer know what weights to manipulate? The checkers’ artificial intelligence called

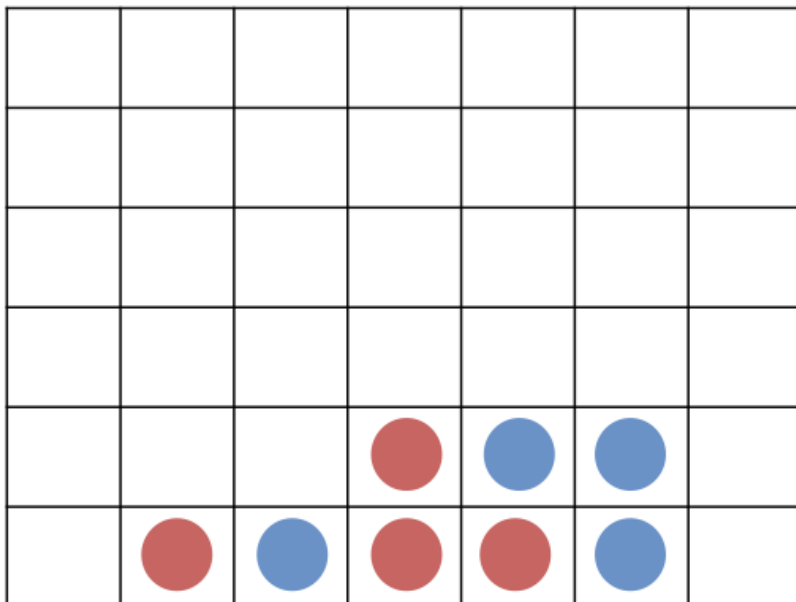
Anaconda chose to manipulate the weights and biases of a neural net. A neural network is a lot more involved than just a standard weighting system, but it has the same general idea. It will use these weights to essentially choose a strategy. In Anaconda's case, the strategy was an eight move look-ahead. However, this was not the only factor the neural net produced, because when tested against a program that chooses moves based only on the number of checkers each player possesses also with an eight move look-ahead, Anaconda won eight out of ten times. This means the results for training the weights provided an extra burst to give Anaconda an expert-level ranking (Chellapilla, 2001).

3. System Design and Implementation

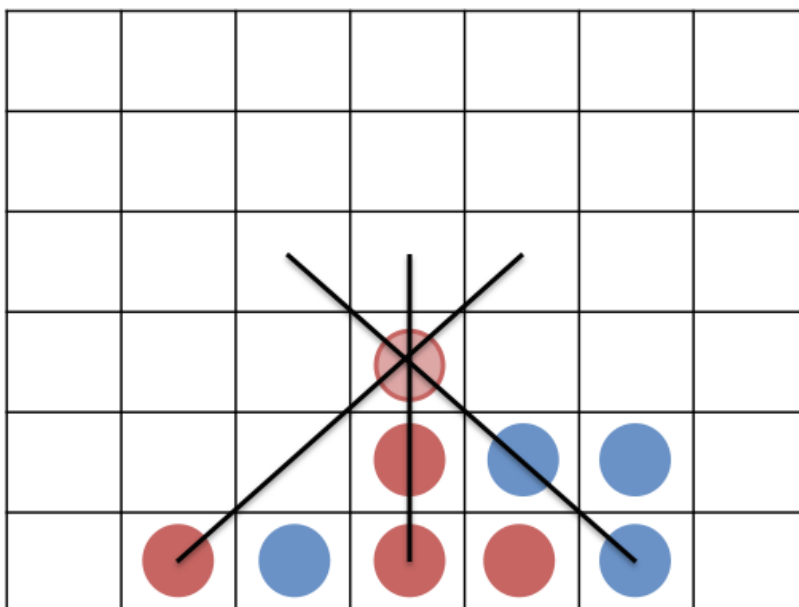
3.1 The Connect 4 Game Implementation

The system we created for this study was designed around a Connect Four game coded by Dr. John Gauch. The goal of Connect Four is to place four pieces of the same color in a single row, column or diagonal on the board before your opponent completes four in a row. The board has seven columns and six rows, and players are only allowed to place their pieces in the first open location at the top of each column. Hence, at each move a player must choose which of the seven columns to add their piece to attempt to complete four in a row.

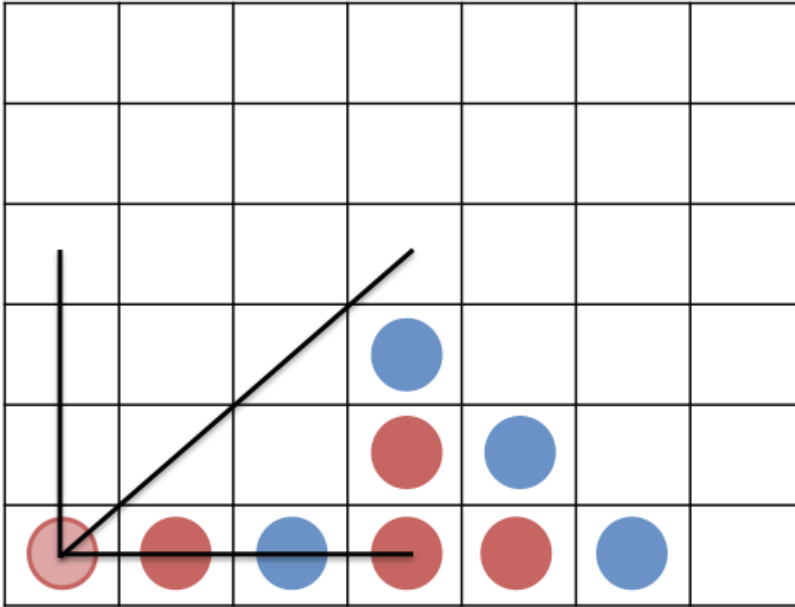
Consider the game in progress illustrated below. Each player has placed four pieces, and it is now time for the computer to decide where to place their next red piece. In this case, the computer must look at all possible sequence of four board positions that include the seven potential move positions. For each potential move position we count the number of red pieces, and the number of blue pieces. Then we use pre-programmed weights to determine the value of placing the red piece in that location.



For example, if the computer places their red piece in column four in the board below, it would block a sequence of two blue pieces on one diagonal, create a sequence of two red pieces on the other diagonal, and create a sequence of three red pieces in a column. Intuitively, this looks like a good move because it creates several opportunities for red to win, while also blocking a sequence where blue could potentially win.



On the other hand, if the computer places a red piece in column one in the board below, it would create a sequence of one red piece in a column, and another sequence of one red piece on a diagonal. This move also adds one red piece to a row that is already “blocked” because it already has one blue piece and two red pieces. Intuitively, this move has less value than the column four move above.



To quantify the relative values of different moves, we use a two-dimensional array of weights. The row index is used to indicate the number of red pieces in the sequence. The column index is used to indicate the number of blue pieces in the sequence. Thus array location $\text{weight}[\text{row}][\text{column}]$ contains the value of placing the computer piece in a row that already has “row” red pieces, and “column” blue pieces. Looking at the full four by four weight array, it was determined that only seven weights have any value to the AI agent because placing a red piece in a sequence that is already “blocked” does not directly increase your chances of winning.

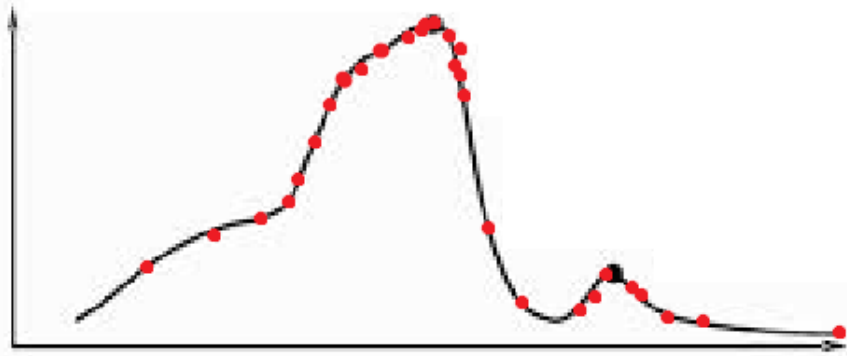
The initial version of our Connect Four game would play against itself using those weights and print out the board after every play. The AI agent would run through every possible sequence of four board locations and count the number of pieces the player currently had in that

possible winning slot and the pieces the computer had. It would then use the fetch a value from the weights matrix associated with those numbers to give that potential move location a score. To make the AI slightly less predictable, a small random weight is added to each potential move. After scoring every possible slot, the AI would pick the best possible option in which a token could be played.

3.2 Genetic Algorithm for Learning

The algorithm I used to aid the artificial intelligence was a version of genetic learning. While I did not use inversion for my genetic algorithm, my implementation of this evolutionary optimization held close to the original design, and because the implementation of the Connect Four game I am researching was made to be weight-based, it was clear what would be my “chromosomes.” I used a population of weights to be optimized through crossover or interpolation and mutation. Interpolation is just finding a number between the two parents. This allows the population to explore new territory that crossover may not reach.

The general idea of optimization is displayed on Graph 1 below. Each red dot represents a member of the population, and the line represents how fit a certain position is. The dots should climb upward to the optimum of each curve, and then shift slightly or produce other members at the top until the optimum is converged upon. By interpolating new members or mutating existing points, we can avoid falling into the trap of local optima by shifting results off a hill or creating a new point between hills. This will allow the population to continue bettering itself until it finds the global optimum.



Graph 1: Population climbing hill to optima.

3.3 Evaluation Framework

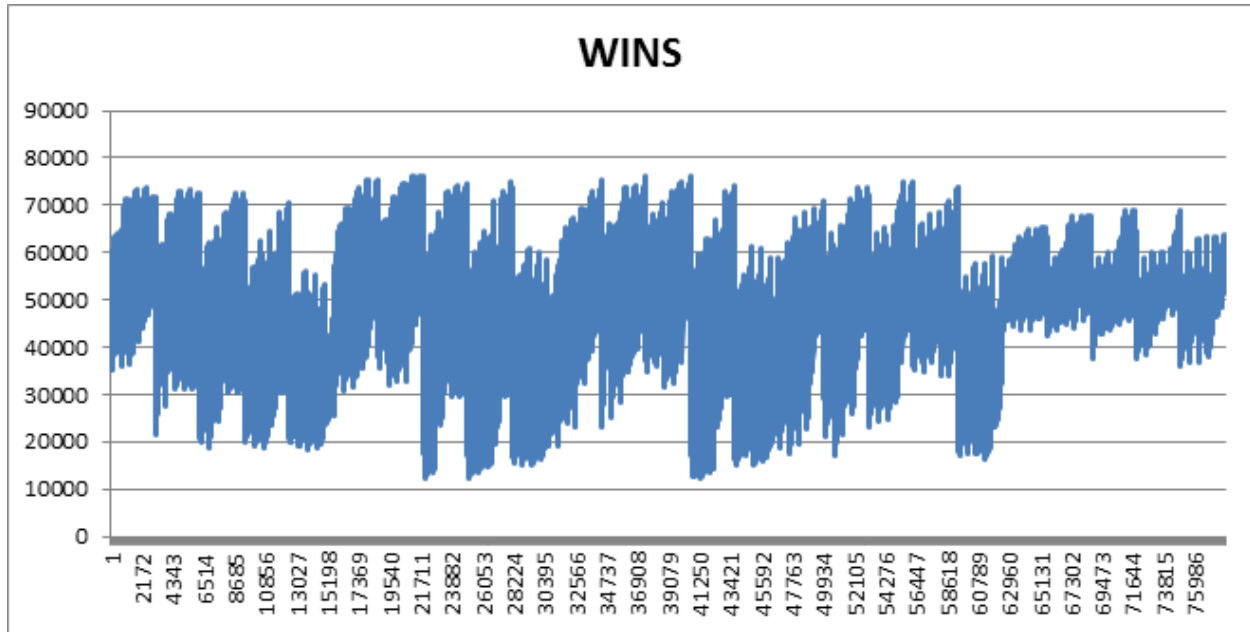
The question becomes how do we know what the global optimum is? In our game AI we are using seven floating point values for the weights to control move selection. Hence it is not possible to perform a brute-force search to locate the global optimum. So, our task is to find a reasonable answer that achieves good numbers but mainly makes the user experience challenging and enjoyable. To do this, I created a plan for evaluation that could cover a wide range within the space. The idea was to start with the brute force evaluations over a certain space to find a good starting position for the genetic algorithm. Then, I play sequences of games to rank each collection of weights based on their win/loss ratio playing all other sets of weights. Then, I will apply genetic algorithm to “learn” new weights. I hope this will allow the weights to greatly surpass the benchmark tests found by picking numbers in a small space. My final experiment will be designed to create the most formidable opponents. I will have two populations play each other, while both are learning the optimal weights for going first and the optimal weights for going second. This way, the offense increasing will increase the defense and so forth.

4. Evaluation

4.1 Brute Force Experiments

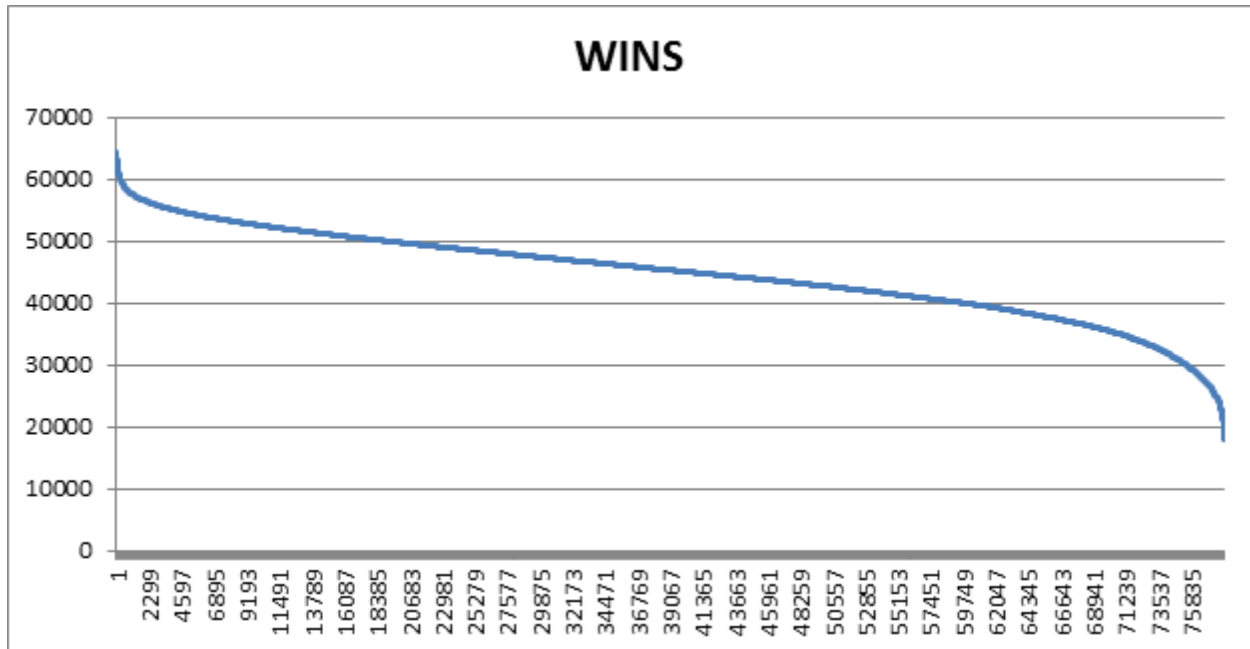
To start the evaluation of the system, I started by creating a baseline or benchmark of weights determined to be the best by a brute force testing algorithm. I had four different sets of predetermined weights that worked on a linear scale (0,1,2,3,4), a ten times scale (0,10,20,30,40), an exponential base 2 scale (1,2,4,8,16), and an exponential base 10 scale(1,10,100,1000,10000). These weights were each separated into their own four pools, which led to there being 78,125 (5^7) different combinations of weights in each pool, and tested with every possible combination inside each of them. This way, I could find the best of each pool to use as benchmarks for my genetic learning algorithms.

Each dataset had the same general pattern with a different range of values. The pattern would increase slightly as the iterations increase defense. Then, it would drop when the defense went back down to zero, but increase faster as offense increased. The middle section of Graph 2 shows the trend that the middle is better. This means a better defense and a more mild offense, such as only attacking when there are three in a row. Graph 2 shows an average graph for the four different sets with the exponential base 10 data set. The graph shows the most wins in the set as each weight changes. The data pool that produced the overall best results was the times 10 scale.



Graph 2: Exponential Base 10 results

The overall trend for the data was a bit better than expected. Because the test weights were only playing a set of contained weights, I expected Graph 3 to have around 50% of the data points above the middle value and 50% below. That is to have weights win and lose to the same set of weights. This was proven false; however, because there are 78,125 different combinations of weights, which would leave the middle point to be 39,062.5. The average number of wins in the linear data pool, which scored the worst overall, was 44,724. Because all of these weights were only being scored by playing first, this led me to hypothesize that having the first move is a very large advantage.



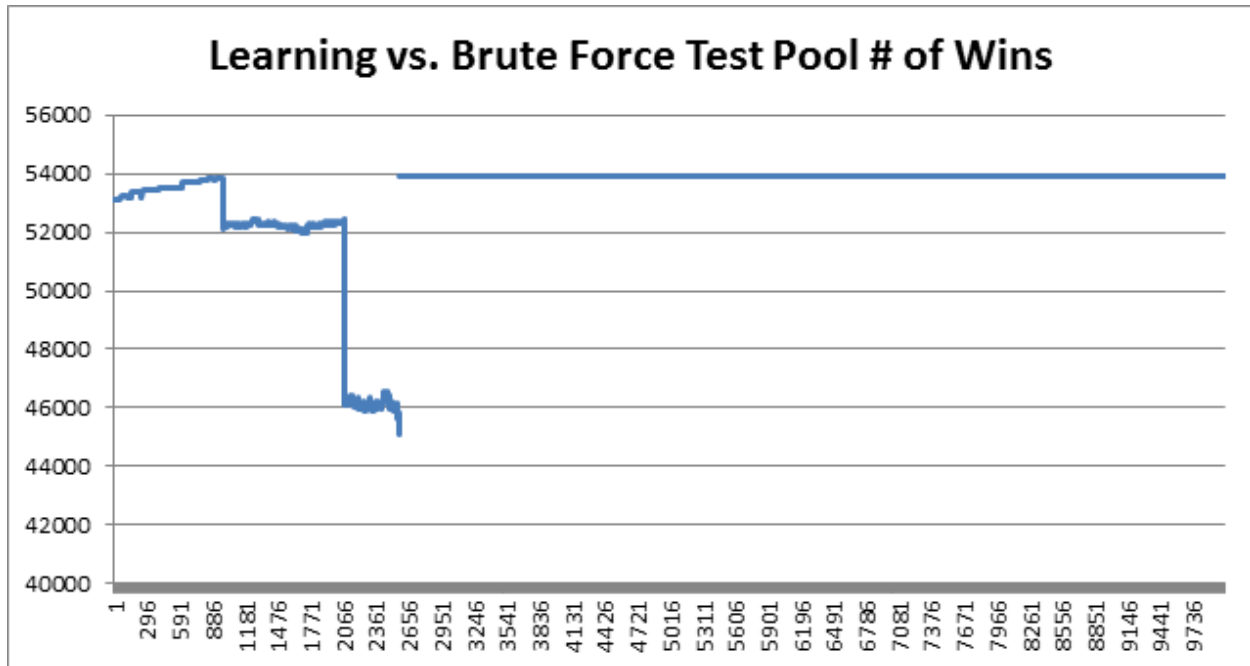
Graph 3: Linear Data Pool, sorted in decreasing order of wins

4.2) Single Population Training

Now that I realized the game would need a different strategy for having the first or second move, the experiment was changed to include a new genetic algorithm. I split this part of the experiment into two sections. The first I will discuss is the original idea of genetic learning. The setup starts with a population of one hundred sets of random weights. These weights were then tested against all of the best weights produced by the brute force tests and given a preliminary score. Then randomness led the rest of the experiment. The algorithm would randomly pick two contestants, check their scores to determine a winner, remove the loser, replace it with a new set of weights, and then recalculate wins of each population member. To replace the weights, the algorithm had a couple of options. It would perform either crossover between the winner and a random member of the population, which picks a random point in the middle of the weights and retrieves the first numbers from the winner and the last numbers from the random member, or interpolation, which loops through all of the member and winner's

values and randomly picks a point between them. Then, in addition to the new member, the algorithm also randomly chooses between either mutating a single member of the population and the entire population slightly. This will help the algorithm avoid any local optima to find the global maximum. These different alterations to the population would happen with a predetermined set of percentages. There is a 50% chance it interpolates or crosses over. Then, there was a 95% chance a single member was mutated and a 5% chance it would mutate all. These numbers were set for consistency in experimentation, but other numbers may have produced better results.

The weights actually learned in an odd way, but I believe it to prove that the genetic algorithm will remove itself from local optima. The results at the beginning of the experiment started off strong with random weights winning 53,110 matches out of a possible 54,959. This definitely makes a point that having numbers that can vary and include floating point numbers makes a large difference over a smaller sample space; however, the test data was only proven when tasked with making the first move. Due to a lot of power being given to the first mover, it could have been moving second that punished the best results of the brute force search over a sample space. Nevertheless, as the experiment continued the number of wins of the best member of the learning population actually decreased. There are two drastic drops (as evidenced in Graph 4) that make this experiment seem like a flop, until a very sudden massive jump puts the best member at 53,894. This means the best of this population beat 98.06% of the benchmark results in wins. Even at the worst point in this test, however, the best member could still beat 45,072 (82.01%) of the test subjects with the advantage of playing first. The overall best member appeared 9,030 iterations into the learning and lasted 280 iterations in first until it was mutated slightly. This member won 53,928 (98.12%) games.



Graph 4: Wins over iterations for the learning vs. benchmark tests

The main question these results produced is why would this learning algorithm settle for worse options and hover there? It started with the best population member being mutated away from its high winning percentage which led to another member being in charge. I believe this member was stuck in a local optimum. This is because of the jitter found after the first major drop. If the line was horizontal, then it would just be one population member. This member was being mutated and bringing new members into the local optimum. But just as before, sets of weights were moving slightly closer to another, even worse, local optimum. After all of the points had successfully been removed from the old due to a mass mutation, they found themselves onto a hill with a much lower win percentage. It only took one interpolation to move a point back to the high standards seen at the beginning of the experiment, which were then maintained throughout the duration of learning. The graph does not do the best job of illustrating the very slight jitter around the winning results towards the end. This was not just the doing of

one population point; there were at least a few different sets of weights. The best set of weights overall was:

108.243 23.7753 105.331 119.609 14.6388 52.6764 83.0098

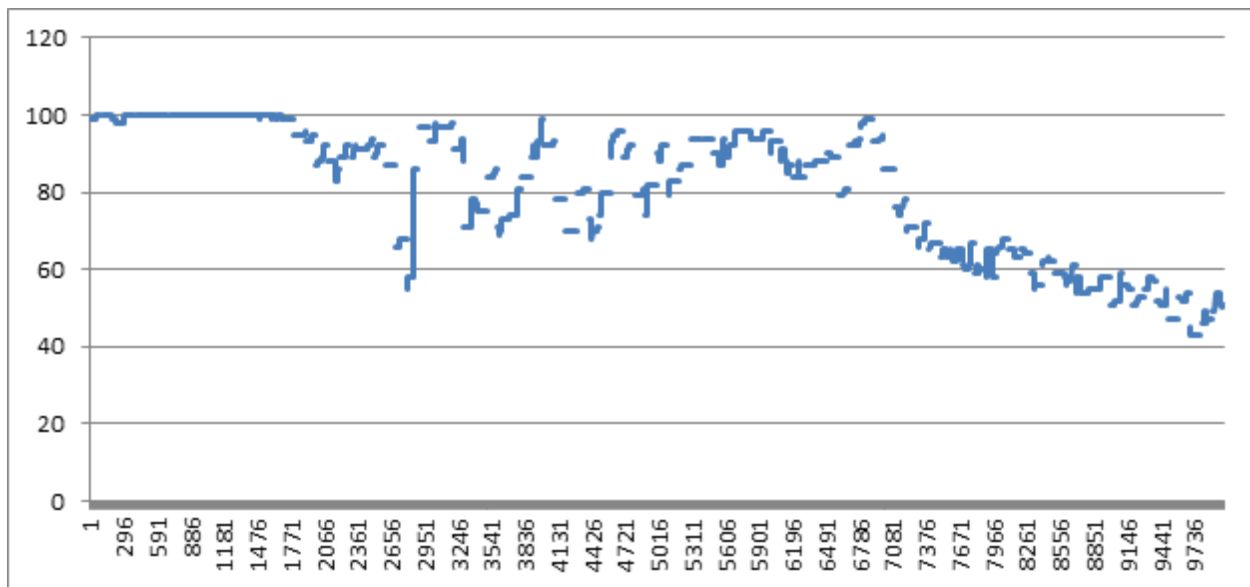
The high numbers happen to fall on attacking first your own lines of 2 or 3, then attack a spot that is open to all pieces, and finally defend on the opponent's lines of 3 and 2. Having 1 token in a slot does not mean as much, and the artificial intelligence treats it as such. As for these numbers, the overall theme is very clear: offense wins. This caused me to believe that in combination moving first and playing a more aggressive game work together as moving second and catering to a defensive strategy. This will be highlighted more in my next experiment.

4.3) Dual Population Training

The second experiment for the genetic algorithm was an attempt to fix this problem that certain weights perform better when going first but worse when moving second. This implementation utilizes two different populations that will both be optimized by the same genetic learning algorithm as the single learning population. That is, there will be one population that plays against a second population to get their scores while the second population keeps track of any wins they accumulate. One pass through each combination will score the entire system. Then, the genetic learning algorithm will go through the "move-first" population one hundred times while accurately keeping track of their wins. Every hundred iterations the algorithm will switch between training the "move-first" and "move-second" populations. This should advance both populations further together and give the user the strongest opponents whether the user goes first or second.

The results of the experiment must be separated into two different sections: the "going first" section, which I will call the fast approach, and the "going second" which I will call the

slow approach. The fast approach is so called, because it has a tendency to attack its own sequences with no regard for the user's. Therefore, the games tend to end quickly one way or the other. Graph 5 shows the wins of the best fast approach sets throughout each generation. The chart starts off very strong (100 is the max) but tapers off towards the end.

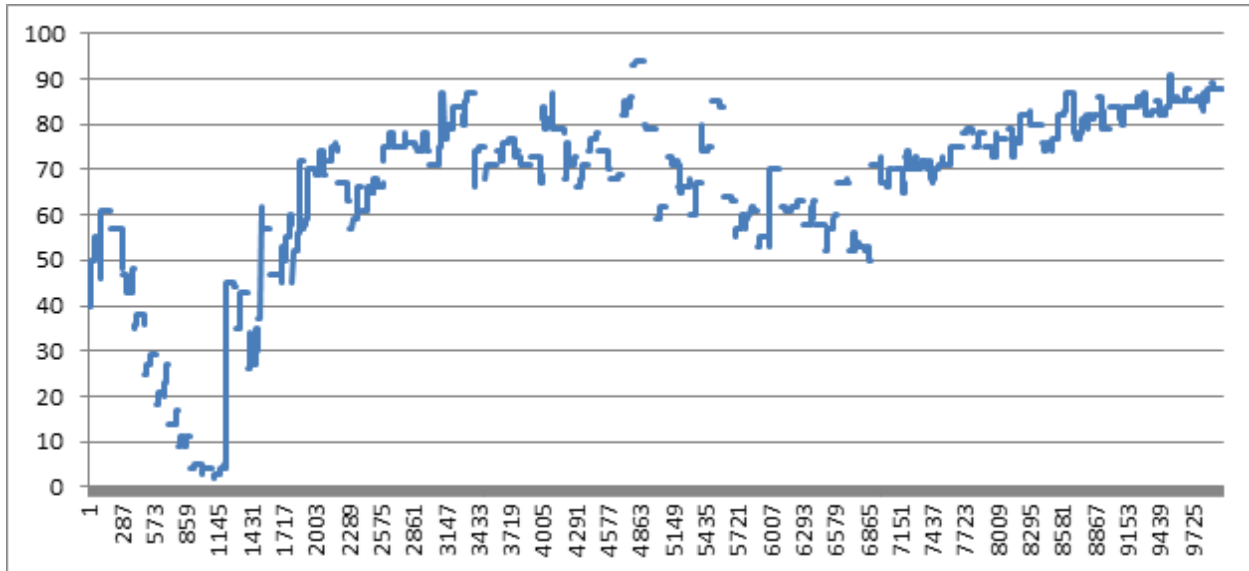


Graph 5: First-mover's wins in the dual learning experiment

The slow approach offers the inverse results as expected, because as the defense grows stronger the offense will slow down, and this is evidenced by Graph 6. The overall results for this slow approach did favor a more defensive strategy. The set of weights produced at ten thousand generations was:

3.48917 17.4376 46.0233 66.7117 10.6567 90.4317 87.6678

This set of weights won 88 games going second, which was determined to be a difficult task.



Graph 6: Second movers wins in the dual learning experiment

While the results were unexpected because the second-movers somehow overtook the first-movers, there was a more shocking reality behind it all. The game was still dumb compared to even a novice player. While the weights were proving themselves against each other, they did not have real-life human instinct to see more than just simplistic patterns. By thinking further ahead than the machine, a user can easily beat the system at play. The results in the world of academia were decent. The idea behind having two systems learn with each other is a good one, but it leads the result to only compliment the one type of strategy the other population deemed most fit. The learning algorithm needs more variety to help it account for more subtleties of the game.

5. Conclusions and Future Work

Can a weight-based artificial intelligence system effectively play against a user in Connect Four? I believe the answer is yes. The most simplistic version that I have presented here has a slight knowledge of the game. It will play in the middle of the board a majority of the time,

which is the best strategy for getting the pieces in the position that the system needs; however, it will not sense any tactics the user uses to open up moves. This can be solved by adding some weights for look-ahead (as shown by the training of weights and biases on the neural net in the Anaconda Checkers system). Allowing the artificial intelligence to have a glimpse into the future would create more opportunity for humanlike behavior. This kind of behavior is definitely needed to succeed in the artificial intelligence world.

There are many more experiments needed to make this system the best it could possibly be other than the look-ahead. One experiment would be having weights tuned by a human to match a strategy in real life as a test instead of computer generated searches. This would allow a slight human aspect and more randomness to be inserted into the experimentation. Another experiment should have a larger number of sets of weights in each population of the dual learning test. I believe this would scatter out some weights to incorporate more strategies that both populations would need to face. It seems as though the good weights come in different clusters, and I believe each cluster could be seen as a different strategy. With more time and work, this system can be improved to be a novice Connect Four Player. There are more fancy tricks to be used to create a grandmaster level nemesis.

Works Cited

- Chellapilla, Kumar and David Fogel. "Evolving an expert checkers playing program without using human expertise." *IEEE Transactions on Evolutionary Computation*, vol.5, no.4, p.422-428 (August 2001).
- Crevier, Daniel (1993), *AI: The Tumultuous Search for Artificial Intelligence*, New York, NY: BasicBooks, ISBN 0-465-02997-3
- Du, Yanzhu et al. (2009). *Applying Machine Learning in Game AI Design Method 1 : MDP*.
- Gashler, Michael S. (2014). *Machine Learning: Keeping it Simple*. University of Arkansas
- Jones, M. Tim (2009). *Artificial Intelligence: A Systems Approach*. Sudbury, Massachusetts: Jones and Bartlett Publishers. ISBN 978-0-7637-7337-3.
- Langley, Pat (2011). "The Changing Science of Machine Learning". *Machine Learning* 82 (3): 275–279.
- Lenat, Douglas; Guha, R. V. (1989), *Building Large Knowledge-Based Systems*, Addison-Wesley, ISBN 0-201-51752-3, OCLC 19981533.
- McCorduck, Pamela (2004), *Machines Who Think* (2nd ed.), Natick, MA: A. K. Peters, Ltd., ISBN 1-56881-205-1, OCLC 52197627
- Millington, Ian; Funge, John (2009). *Artificial Intelligence for Games*. Boca Raton, Florida: CRC Press. ISBN 978-0-08-088503-2.
- Mitchell, Melanie (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press. ISBN 9780585030944.
- NRC (1999), "Developments in Artificial Intelligence", *Funding a Revolution: Government Support for Computing Research*, National Academy Press, ISBN 0-309-06278-0, OCLC 246584055.

Popperipopp, 2008, http://en.wikipedia.org/wiki/Connect_Four.

Ramachandran, Deepak, 2005. "First-Orderized Research Cyc: Expressiveness and Efficiency in a Common Sense Knowledge Base". Retrieved 02 March 2015.


Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2.

Schadd, F.; Bakkes, S.; and Spronck, P. 2007. Opponent modeling in real-time strategy games. In GAMEON, 61–70.

www.nevermindgame.com/biofeedback/

This thesis is approved.

Thesis Advisor:



Dr. John Gauch

Thesis Committee:



Dr. Matthew Patitz



Dr. Christophe Bobda