Tony Morelli

05/06/2005

Using Co-Evolution and Neural Networks to Generate a Tic Tac Toe Player

Abstract

This paper describes a method of learning to play Tic Tac Toe. The strategy for Tic Tac Toe is derived from a neural network with parameters evolved by a genetic algorithm. The neural network consisted of the board positions as inputs and a score as an output with two hidden layers. The score was then used to evaluate potential board positions with a minimax algorithm making the actual move decisions. Two different experiments were performed: the first experiment involved evolving the neural network parameters against a known good opponent, the second experiment used co-evolution of neural network parameters with 2 separate populations. Each member of one population played games against the top 5 members of the other population. Fitness was recorded, and cross over and mutation took place as necessary. To actually evaluate how well the two populations were evolving against each other, after each generation, all members of the population were run against the known good opponent to determine how good the player actually was. Although the average player was never able to beat or tie the known good opponent, the results are promising that with a few tweaks that can happen.

Introduction

The game of Tic Tac Toe is played on a 3 by 3 grid. One player marks his spots with an X, and the other marks with an O. The players alternate placing marks on the grid with the hopes of winning the game. The winner of the game is the first player to place 3 of his marks in a row. If the entire grid is filled with marks, and there is no winner, the game is determined to be a draw. This project created a player to play the game without any existing knowledge of Tic Tac Toe strategy. The only rule that was known to the evolving player was that the players alternated turns, and when it was the evolving player's turn, it could only place a mark on an empty spot in the grid. To create an artificially intelligent player, three AI (Artificial Intelligence) techniques were used. They were the minimax algorithm, artificial neural networks, and genetic algorithms.

The MiniMax Algorithm is a search method applicable in 2 player turn based games. These games have all future

moves readily available to the opponent. Games such as Tic Tac Toe, Checkers, and Chess are examples of games in which the MiniMax algorithm is useful. The way the algorithm works, is that 1 player is the maximizer, and the other player is the minimizer. The object of the maxizer is to maximize his potential score by anticipating the minimizer is going to minimize his score. In this method, the maximizer will place a token in every available space, then play all of the newly created boards as the minimizer, and so on. Each look ahead is considered a ply. The more ply, the more accurate the player will be as the player can 'see' more into the future of possible board positions. In order to use the MiniMax Algorithm, each board state must be able to be assigned a value. For example, if X is the maximizer a board position that shows 3 x's in a row would receive a very high score. If O is the minimizer a board position that shows 3 o's in a row would receive a very low score.

Artificial Neural Networks are a simulation of real neural networks located in our nervous systems. The neurons in the nervous system contain 3 parts, the cell body, the dendrites (input), and the axon (output). The axons of one neuron are connected to any number of dendrites of other neurons. The neuron will take all of its inputs, and analyze them, and if they meet certain requirements, it will send pulses out its output. An Artificial Neural Network works on the same idea. The nodes in a neural network contain any number of inputs and outputs. Each node has a threshold, which means if the sum of all its inputs goes beyond the threshold, it will turn on its outputs. Each output of a node has an associated weight with it. This weight is multiplied by the output value before it is added in the summation of the inputs of the next node.

Genetic Algorithms are a method of evolving data to find the best solution to a problem. They are inspired by Darwin's theory of evolution. The solution to a problem is evolved by keeping the best parts of one solution, and combining them with the best parts of another solution. Just like in nature, a genetic algorithm has generations. Inside of a generation is a population of potential solutions to a problem. The population of the next generation is created by mating two members of the current population. Just like in nature, an offspring has characteristics of both of

its parents. If a member of the population is a good solution to the problem (has a high fitness) there is a higher chance it will be chosen to be a parent. Through this process, new potential solutions are created. After a certain number of generations, the average fitness starts to approach the maximum fitness, and at this point, the maximum fitness is selected as the solution to the problem.

There has been a lot of work involving neural networks and genetic algorithms. This project is most similar to a method described in David Fogel's Blondie24. In this book he uses evolved neural networks to co-evolve a checkers player that utilizes the minimax search algorithm to make move decisions. A similar method is used here to co-evolve a tic tac toe player.

The results of these experiments are promising. The neural net players were evaluated against the perfect player, and after some time, they had a player that could tie the best player. The average evolved player was not playing the perfect player to a draw all the time, but there were members of the population who were. Letting the GA run for more generations could have changed this.

The rest of this paper will be setup as follows. The first part will go over how the experiment was setup. How the genetic algorithm was used, how the neural net was created, how the perfect player was utilized, and what data was used will be described in detail. The second part will display the results generated from the experiments. And finally, some conclusions will be made as well as a brief discussion on future work.

Methodology

This project utilized the MiniMax Algorithm, Neural Networks, and Genetic Algorithms to create a Tic Tac Toe player. How each of the three were individually used is described below, followed by how they all interacted with each other.

The MiniMax Algorithm is a search algorithm used to determine the best move in a 2 player turn based game where each player knows all of his opponents potential moves. The more moves (ply) a player can look into the future, the better prepared the player will be to make a more accurate decision. In this experiment, the number of ply for each player to look ahead was configurable in a configuration file. I tested ply from 1 to 4, with 4 being the

optimum for a tic tac toe game. To evolve players, I used a static evaluator which is known to be 'unbeatable'. This evaluator returns a score of a board given the location of the x marks and the o marks. It was also used to determine the quality of the 2 populations that were evolving against each other in my second experiment. The static evaluator looked at all the pieces on the grid. It counted wins as +30 (if x won) or -30 (if o won), almost wins (2 in a row with a potential for a win) as +20 or -20, and blocks (2 in a row with the other token blocking the win) as +10 or -10. so if the ply was set to 4, the x player would generate all combinations 4 ply into the future, evaluate all boards at that level using the static evaluator, and then use the minimax algorithm to ultimately make the correct decision as to which way to go. The evolving populations of neural networks were used as a neural network evaluator as opposed to the static evaluator. The specifics of the neural net are laid out below, but basically the neural net was used to return a value for each potential board layout, returning a high score for board that x would like (the maximizer) and a lower score for a board that o would like (minimizer).

The neural network used in these experiments is used to evaluate board layouts. Each board has an associated score for it generated by the neural net. This value is then used by the minimax algorithm to make the correct decisions. The neural net has 9 inputs, which correspond to the nine spaces on a tic tac toe board. The inputs are a 1 if there is an X in that location, a -1 if there is an O in that location and a 0 if that location is empty. The neural net used here has 2 hidden layers, with 1 output. The output is the score of that particular board. The hidden layers consist of 18 nodes in the first layer followed by 5 nodes in the second layer. All 9 inputs are wired to all 18 nodes in the first hidden layer, and all 18 nodes in the first hidden layer are wired to the 5 nodes in the second layer. The weights and thresholds varied from -8 to 8, and were determined by the genetic algorithm. The overall structure of the net remained constant, but the values of the thresholds and weights were evolved.

The genetic algorithm was used to evolve the weights and thresholds inside of the neural network. The GA used a crossover rate of 0.667 with roulette wheel selection and a mutation rate of 0.001. The GA was run for 100

generations and the data was averaged over 5 trials. The population size was set to 100. Each member of the population had a chromosome length of 590 bits. This was for the 95 weights and 23 thresholds in the neural net. Each value was allotted 5 bits. The value for each of the parameters was created as follows. The 5 bits were converted to an integer, then divided by 2. That number was then added to -8 to come up with that particular value.

All three of these AI components were interconnected to create a tic tac toe player. Two experiments were performed. One involving 1 neural net evolving against a known good player, and the other involving 2 neural networks evolving against each other with the static evaluator being used as a benchmark. In the first experiment, the GA would chose settings for the neural network and then create the network. That network would be passed into a tic tac toe game, where a game would be played against the known good static evaluator. Once a winner was determined, the tic tac toe game would evaluate the final board positions and return a value of that board. The GA would then use that value to perform roulette wheel crossover and mutation, if needed, on the entire population. In the second experiment, the GA maintained 2 sets of populations and put them head to head in a tic tac toe game. Each member of a population would play the top five of the other population and the total score would be totaled and used for the roulette wheel crossover. In order to determine how the population was evolving, each member of each population played the known good static evaluator. These runs were only used to evaluate the members in the population against a known point. All decisions for crossover were based on their scores in head to head competition.

Results

The initial results from these experiments are promising. In the first experiment (1 evolving neural net vs a known good opponent), two different configurations of the program were run. The first configuration set the ply for both players at 4. The GA was run for 100 generations, and the results were averaged over 5 trials with different initiating seeds. The fitness range was expected to be between 0 and 480, with 480 meaning the neural net won every game, and 0 meaning the static evaluator won every game. A score of 240 meant every game was a draw. Prior to

running the experiment at 4 ply, I expected the GA to produce a player that would achieve a score of 240. Since there was no way to beat the static evaluator at 4 ply, the best I could expect the neural net to do was to draw. And thats exactly what happened. After 100 generations, there was always someone who consistently scored a fitness of 240. The average fitness of everyone in the population was 181.3. That means the average neural net would lose some time against the static evaluator.

A second experiment was performed which was set up the same was as the previous, except for one small change. The static evaluator, which was known to be perfect at 4 play, was set to only look ahead 1 move (1 ply). That was the only change that was made, and the GA was once again run to create and evaluate neural networks. I expected the neural networks to have a much higher fitness. The results were at first somewhat surprising. After 100 generations, the optima had flattened out at 190. This means that no neural net had been evolved that could draw or beat the static evaluator at 1 ply, even when the neural net was configured for 4 ply. Thinking about why this happened, I have come up with the following theory. You are only as good as your opponent. And since the network was attempting to learn to play tic tac toe against someone who basically did not know what he was doing, the neural net learned to make the same bad decisions.

In the second part of this project, the GA maintained 2 separate populations of neural network parameters, and played the top performers from one population against all the other members of the other population. The quality of the neural networks being created was monitored by having each member of each population play the static evaluator to a ply of 4 to see how good they really were. I expected to see results very similar to the first experiment where members of the population would start to achieve a score of 240, meaning they were playing the perfect player to a draw. After 100 generations, the optima had flattened out at 240 for each of the populations, which showed that they both had evolved players who could play the known perfect player to a draw every time. The average player for each population was at 179.1. This again means that the average player in the population was not as good as the known perfect player, but each population did have members who could play the

perfect player to a draw. The similarity between the co-evolved results and the single evolving neural net show that the co-evolution was working, although it did not produce results that were any better than evolving against a known good player. The benefit is that if a known good player is not available, or not definable, the co-evolution process will produce members who are good players.

Conclusions/Future Work

The results show that co evolution can produce as accomplished of a tic tac toe player as evolving a single population against a known perfect player. I was a little surprised that the average player in the population ended up being as bad as they did, but I think changing the GA to use rank selection, as well as utilizing elitism could have bumped up the average.

To continue this project, I would like to take the knowledge obtained here and use it in non-turn based games. I think using a GA with a neural net should yield good results if the GA and the neural net are structured correctly. The other interesting part which needs further investigation, is the creation of the neural net itself. My neural net layout was based on quite a bit of different neural net layouts that have been created in the past. There was no designing this neural net to be specific for a tic tac toe game. It would be interesting to use a genetic algorithm to create neural network layouts and then to use another GA to populate the weights and thresholds. This would be a GA within a GA. Besides taking up a lot of CPU power, this should result in the perfect neural net overall layout, as well as the perfect weights and thresholds for that neural network.

References

1. David E. Goldberg, *Genetic algorithms in search, optimization and machine learning* 1989, Addison-Wesley.
2. Patrick Henry Winston, *Artificial Intelligence* 1992, Addison-Wesley
3. Dacid Fogel, *Blondie24* 2001, Morgan Kaufmann