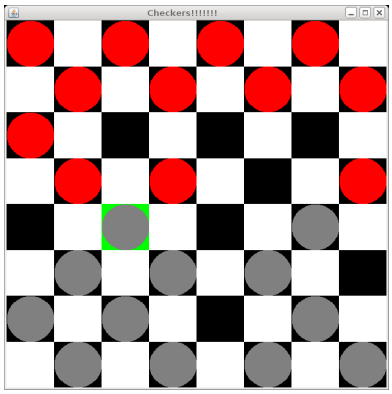
The Implementation of Machine Learning in Checkers



Abstract

Most games have a set algorithm that does not change. This means that these programs cannot adapt to a situation or learn from mistakes that it makes. However if a machine could learn, then it could adapt to new situations and would have a nearly boundless skill level. Machine Learning programs can be beaten once, but against an opponent that did not change, it eventually be able to beat it. The project that I am writing will learn how to play the game of checkers as it plays, by modifying itself after ever game played. It will review its play and if it played well it will play that way more often, if it did not it will avoid that way of playing.

Background

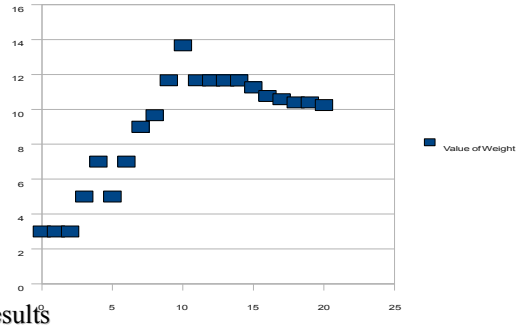
One method of creating a learning program is to have the program learn by rote. An experience base stored all of the boards encountered and the moves that had been tried off of these boards. While this is similar to creating all of the possible board combinations and simply searching them exhaustively, this does not require as much memory and is used in conjunction with a static evaluation function. A different method of creating a learning program is described by Olsen (2), generalization learning. The program modifies the heuristic function each time it plays based on the outcome of the play. The program starts with a heuristic function that has all of the features weighted equally. After the program plays a game, it will change the values of the weights of the heuristic function based on the outcome of the game. This type of learning when implemented in a program, generally played very well during the middle game play, but less well in beginning and end game play, while the rote learning process played very well in end game play.

Development

For the learning by generalization algorithm I will use a temporal difference learner. Temporal difference learning adjusts the evaluation function during play based on the difference between the evaluation function at one point and the evaluation function at a different time. This will bring the evaluation function into a state of equilibrium toward the ideal evaluation function. Every time another move is made, the heuristic is changed based on the previous heuristic. The equation for this is $U(s) <-- U(s) + \alpha(R(s) + \gamma U(s') - U(s))$. s is the current state of a system, U is the evaluation function of state s, α is the learning coefficient and decreases as the number of times state s has been encountered increases and R is the reward that you will get for state s.

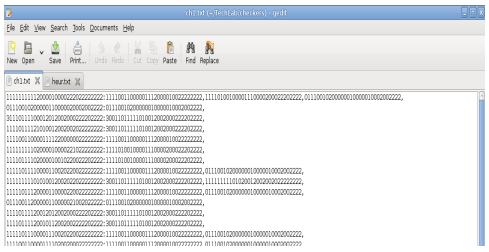
The learning by rote system that I am using also uses the minimax algorithm. But instead of learning a better and better heuristic function, it stores a set of boards that it has seen that are most probable to be played. The program then uses this to increase the depth of the minimax search because the time it takes to access a stored board is much less than the time it takes to find all of the moves that can be made. This increases the skill of the AI by increasing the number of turns the AI can look into the future.

My program stores each board that was visited and the boards that can be reached from that board. It does this by converting each board state into a number. The number is a 32 digit number of base 5. There are 32 digits for each space on the board that a piece can be in, and base 5 for each piece that can be there. Then the program maintains a hashmap of the boards where a board is the key and each board that can be reached from that board is the value. At the end of play, the program prints the data out to a file for the next iteration of the program.



Results

This graph shows the value of one of the terms of the heuristic over the course of a game. This weight is the weight attached to the value of having more pieces than the other player. In the beginning the AI had a large underestimate of the value of the weight and increased it at an increasing rate and then the weight was overshot in the course of the game, so the program then begins to lower the heurisitc by progressivelly less amounts. The porject ed “optimal” value of this wieght is somewhere around 10.



This is a copy of the output of the learning by rote file. On the left is a board represented by the string of numbers. On the right of the colon is the list of boards that can be reached from the original board.

This graph shows the number of boards stored in the game database. You can see at the beginning the database acrues a large amount of boards, but as the game is played, the number of boards levels off. This is because as you play more games, the probablility that you will see knew games decreases. This decreases the returns of the learning by rote process. This also requires large amounts of memory.