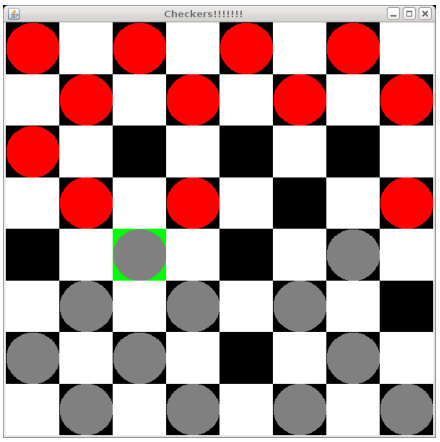
The Implementation of Machine Learning in Checkers



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

Most programs that play games today use a search through a tree of moves and boards. Each different possible move and the board that would result from that move is evaluated and when all of the boards have been searched an evaluated; the program chooses the move that results in the best outcome quickest. However this can become very costly very quickly. There is not enough time to search through the entire tree for all but the simplest games. To combat this, you must make choices as to which moves you search. Heuristics were developed for this reason. The heuristic determines which boards are good and which ones are bad, without being able to perfectly predict the results of the game. The heuristic for each game is different and a common form of a heuristic is

$H(s) = c0F0(s) + c1F1(s) + \ldots + cnFn(s)$

Where H(s) is the value of the board, Fi is a feature of the particular board for example a feature of a checkers game would be how many checkers you have compared to your opponent, and ci is the weight given to that particular feature (2 Olsen). How accurately this function predicts the outcome is how effective the program will play. Creating a heuristic for a game generally requires an expert and specially tailoring the heuristic to its application. This is why a learning program would be useful. The program could learn its own evaluation function that would be more accurate and with less work than having a static function.

One method of creating a learning program is to have the program learn by rote. One program designed by De Jong and Schultz (1) used an experience based that the program would reference when it made moves. The 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

I will use a combination of the learning by rote algorithms and the generalization algorithms. I will create a file and store all of board combinations that have been seen and the moves taken from that particular board. I will then also use a changing heuristic to evaluate each board. The heuristic will also learn based on the outcome of the game. Since the ability of these methods changes based on where in the game you are, I will attach more weight onto the predictions of the learning by rote method in the beginning and end game and more weight to the generalization algorithm during the middle game phase.