The Implementation of Machine Learning in the Game of Checkers

William Melicher
Computer Systems Lab
Thomas Jefferson

March 12, 2009

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 does not change, it eventually will 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.

1 Introduction

The game of checkers has been weakly solved by a computer program. The program generated every possible board combination and simply uses a brute force method of searching through these at every move. However it would be nice to not require a super computer to play the game of checkers. A Machine learning program would be able to play at a high level of play without requiring huge data bases or large amounts of processing power for each move. The game of checkers has lower complexity level compared to other games like othello, chess, go, and others, but the use of machine learning in this program can also be extended to those games and other situations that require a learning program.

The basic principle of machine Learning is that the program can use past examples of a situation to predict how an action will end up in the future. So a machine learning program would be able to continually adapt to the circumstances that it is in. This type of program would also be able to adjust its play when it is playing an opponent that does not change. The program needs to play games to learn how to play, and more games would give it more experience to learn how to play. The program needs to play against itself to get a large enough experience base to be able to play well.

This research could be applied to any situation that requires quickly solving many similar problems in succession. Any problem where previous data can be used to predict a future situation would benefit from a machine learning algorithm. The program would take the data from previous problems and the outcome from those problems and then use it to predict the outcome of a particular situation.

Checkers is a game that has many tactical aspects. The object is to remove all of the opponents pieces by jumping them, and avoiding loosing all of your own pieces. It is necessary to block your opponents moves and create situations where you can jump the opponents checkers. You may only move forward, until you reach the back of the board. These pieces have much more power because they can move in four directions and not just two. Also the edge of the board is more defensible than the middle of the board, because pieces cannot jump over you. After making a jump of a particular piece, if you can make an additional jump, you may.

2 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. This is called the minimax algorithm. Each player wants to act in their own interest, to either maximize or minimize an evaluation function called a heuristic. The heuristic determines which boards are good and which ones are bad, without being able to perfectly predict the results of the game. The minimax algorithm looks at all of the possible boards that can be reached

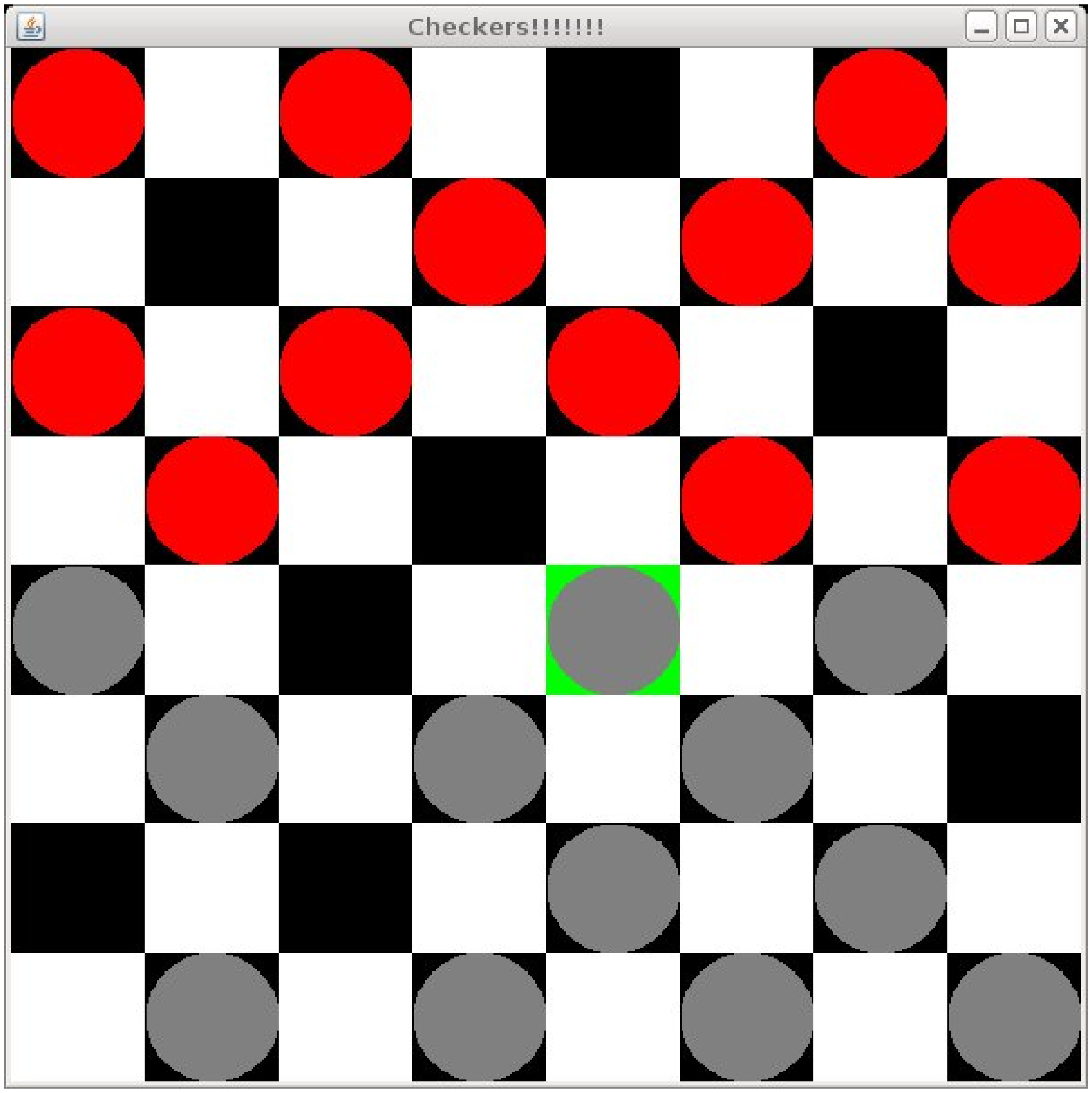

Figure 1: Here the selected gray piece can double jump the two red pieces. After the jump it will be on the back row which means it will become a king.

from one state and then looks at all the boards that can be reached from another state, and so on and so on. At some depth the algorithm stops and then applies the evaluation function to determine the fitness of a board. At each level, the player whose turn it is will make the move that results in the best outcome for them. This algorithm is limited by two things, which are the depth of the decision tree that it searches, and the quality of the heuristic function. However this can become very costly in terms of time very quickly. There is not enough time to search through the entire decision tree for all but the simplest games. The heuristic for each game is different and a common form of a heuristic is

$$H(s) = c_0 * F_0(s) + c_1 * F_1(s) + \; + c_n * F_n(s) \tag{1}$$

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). The play of the AI increases in skill as you can more perfectly predict the future states of the board. This means having an AI that accurately describes the position of the board and the minimax algorithm can go to a large depth. 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.

3 Developement

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 capability 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.

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

$$U(s) < - - U(s) + a(R(s) + yU(s') - U(s)) \tag{2}$$

s is the current state of a system, U is the evaluation function of state s, a 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.

References

[1] De Jong, Kenneth A. and Alan Schultz, Using Experience-Based Learning in Game Playing., 1998.

[2] Olsen, Daniel, Learning to Play Games From Experience: An Application of Articial Neural Networks and Temporal DifferenceLearning. 1993.

[3] Norvig, Peter and Stuart Russel. Artificial Intelligence, a Modern Approach. New Jersey Pearson Education, 2003.

[4] Rich, Elaine and Kevin Knight. Artificial Intelligence. New York Mc-Graw1991.