The Implementation of Artificial Intelligence and Machine Learning in a Computerized Chess Program

by James Mannion
Computer Systems, 2008-2009
Period 3

Abstract

Computers have developed to the point where searching through a large set of data to find an optimum value can be done in a matter of seconds. However, there are still many domains (primarily in the realm of game theory) that are too complex to search through with brute force in a reasonable amount of time, and so heuristic searches have been developed to reduce the run time of such searches. That being said, some domains require very complex heuristics in order to be effective. The purpose of this study was to see if a computer could improve its heuristic as it runs more searches. The domain used was the game of chess, which has a very high complexity. The heuristic, or evaluation function, of a chess program needs to be able to accurately quantify the strength of a player's position for any instance of the board. Creating such an evaluation function would be very difficult because there are so many factors that go into determining the strength of a position: the relative value of pieces, the importance of controlling the center, the ability to attack the enemy's stronger pieces – something that chess masters spend entire lives trying to figure out. This study looked to see if it was possible for a computer program to "learn" an effective evaluation function by playing many games, analyzing the results, and modifying its evaluation function accordingly.

Background

In 1950, Claude Shannon wrote a groundbreaking paper called "Programming a Computer for Playing Chess." It dealt with various issues, such as how one might go about writing an artificial intelligence program that could play chess well, and what the evaluation of such a program might look like. It discussed the problems involved with brute-force chess algorithms and suggested an AI algorithm that a computer chess program could use. It suggested using a 2-ply algorithm (an algorithm that looks two turns into the future before applying the evaluation function), however if the program finds that a move could lead to a check or checkmate, then it would investigate that move and subsequent moves out to as many as 10 turns. At the end of the paper, Shannon speculated about the possibility of computer programs that can "learn," but noted that such developments were probably man years down the road.

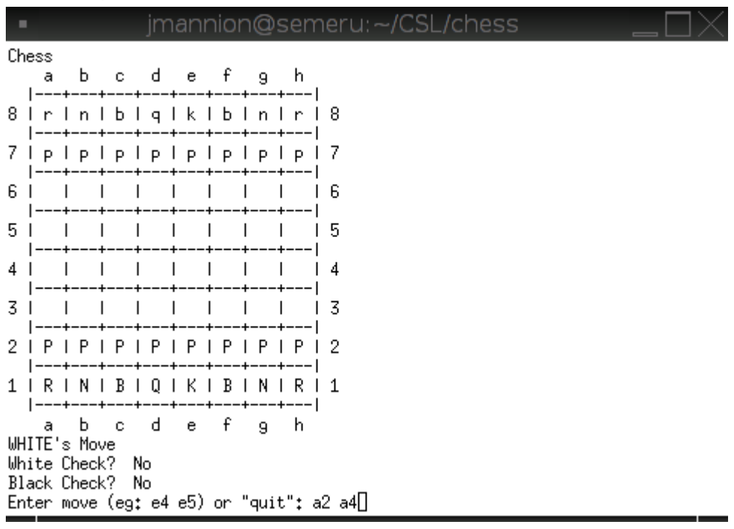
About 50 years later in 1999, D.F. Beal and M.C. Smith published a paper called "Temporal difference learning for heuristic search and game playing." At this point, programs that could play chess had been well-developed, but research in machine learning was really just beginning. The paper investigated the use of Temporal Difference learning as a way of making a computer chess program modify and improve its evaluation function each time it plays. The researchers made their chess program learn for 20,000 games, and then ran their program against a program that had not learned its evaluation function. The learned program performeddecisively better over 2000 games than the unlearned program, showing that it is indeed possible for computers to improve their evaluation functions.

References

Shannon, Claude E. "Programming a Computer for Playing Chess." 1950.
Beal, D.F. and Smith, M.C. "Temporal Difference Learning for Heuristic Search and Game Playing." 1999
Moriarty, David E. and Miikkulainen, Risto. "Discovering Complex Othello Strategies Through Evolutionary Neural Networks."
Huang, Shiu-li and Lin, Fu-ren. "Using Temporal-Difference Learning for Multi-Agent Bargaining." 2007
Russell, Stuart and Norvig, Peter. Artificial Intelligence: A Modern Approach. Second Edition. 2003.

Development

The first stage of the program simply involved a console-based chess game where two humans could input their moves into the command line, and the board would be re-printed with the new move, as shown



Stage two of the project involved writing simple AI players. The first AI player written was one that simply made a random legal move. The next AI player written was simply a three-ply minimax search with alpha-beta pruning. Its heuristic function was such that it would chose moves based on a simple piece differential, with each piece being weighted the same as every other piece, regardless of position on the board. These two programs were created for the purpose of comparing the progress of a player with TD learning against a constant player over time.

Stage three introduced temporal difference learning. TD learning compares predictions of future board states to actual values seen later on in the game and adjusts the weights of the heuristic function in order to make predictions made in later games closer to observed values. As more and more games are played, the weights of the heuristic function will improve and move toward equilibrium values. The more in-game values that are observed, the more accurate and more effective these equilibrium values will become.

$$h = c1(p1) + c2(p2) + c3(p3) + c4(p4) + c5(p5)$$

$$pi = sum(wi) – sum(bi)$$

Testing/Analysis

The program with TD learning was run against a static program. All weight values in the heuristic were initialized as 1. The win-loss differential of the learner was tracked over 2,000 games to see if the learner would start winning more and more as it underwent the learning process.

The altered piece-square tables were analyzed after the learning session was complete to see if the program developed generally accepted strategies such as maintaining control of the center. Higher numbers toward the center of the boards would indicate such a strategy.