The Implementation of Artificial Intelligence and Temporal Difference Learning Algorithms in a Computerized Chess Program

James Patrick Mannion
Computer Systems Laboratory
Thomas Jefferson High School for Science and Technology
Alexandria, Virginia
2008-2009

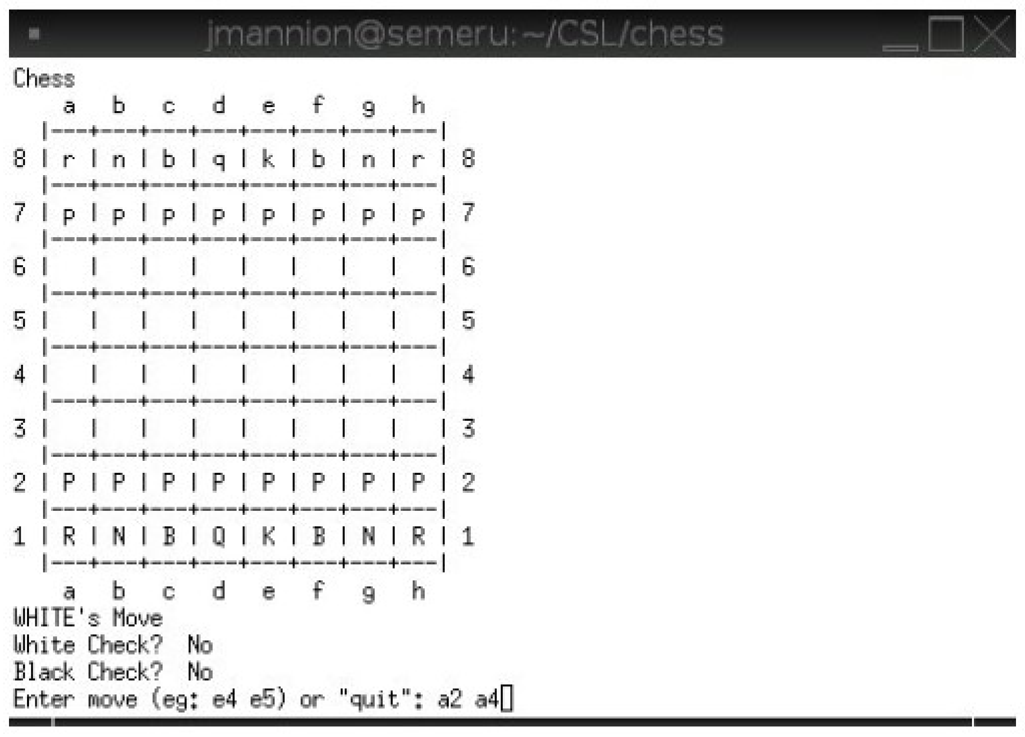
Abstract

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 (or learn) 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 enemys 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 and modifying its evaluation function to achieve better results. The process by which the program improved its evaluation function is called Temporal Difference learning, which does not require the program to know anything about chess strategies before learning begins and which looks at how changes in the heuristic function affect the predicted strength of the outcomes of the current boardstate in order to adjust the evaluation function appropriately. [1]

1 Development

Python was used to code this chess program. The early stages 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 in Figure 1.

This program underwent numerous debugging sessions to make sure that it would correctly handle illegal moves, checks, checkmates, castling, the obscure en passant pawn capture, and other game play mechanics. For the sake of simplicity, stalemates were called after 200 moves (100 turns) without checkmate.



```
Chess
    a   b   c   d   e   f   g   h
  |---+---+---+---+---+---+---+---|
8 | r | n | b | q | k | b | n | r | 8
  |---+---+---+---+---+---+---+---|
7 | p | p | p | p | p | p | p | p | 7
  |---+---+---+---+---+---+---+---|
6 |   |   |   |   |   |   |   |   | 6
  |---+---+---+---+---+---+---+---|
5 |   |   |   |   |   |   |   |   | 5
  |---+---+---+---+---+---+---+---|
4 |   |   |   |   |   |   |   |   | 4
  |---+---+---+---+---+---+---+---|
3 |   |   |   |   |   |   |   |   | 3
  |---+---+---+---+---+---+---+---|
2 | P | P | P | P | P | P | P | P | 2
  |---+---+---+---+---+---+---+---|
1 | R | N | B | Q | K | B | N | R | 1
  |---+---+---+---+---+---+---+---|
    a   b   c   d   e   f   g   h
WHITE's Move
White Check?  No
Black Check?  No
Enter move (eg: e4 e5) or "quit": a2 a4
```

Figure 1: The starting board, with white about to move a pawn.

A program was then written that could do a minimax search to play chess. It was then given temporal difference learning. TD learning compares predictions of future board states to predictions 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 w_i will be modified in the following manner (similar to [3]):

$$w_i \longleftarrow w_i + a(P_t - P_{t-1})\partial_{w_i} P_{t-1} \tag{1}$$

$$a = \frac{200}{199 + t} \tag{2}$$

$$P = \frac{1}{1 + e^{-h}} \tag{3}$$

$$h = \sum_{i=1}^{5} w_i x_i \tag{4}$$

$$x_i = j_i - k_i \tag{5}$$

Here, j_i and k_i are respectively the number of black and white pieces of type i, a is a coefficient that decreases as more and more board states are seen by the program, $\partial_{w_i} P_{t-1}$ is the partial derivative of the predicted success rate (which has been squished to a number between 0 and 1 from the evaluation function h) with respect to the weight, t denotes a current timestep and $t-1$ denotes a previous value. If the current value is favored, the weight will increase. If the previous value is favored, the weight will decrease. The purpose of a is to make the weight change less and less as more games are played so that the weight closes in on an equilibrium value over time.

2 Testing

The program with TD learning was run against the computer player described above that used a simple piece-differential heuristic. All weight values in the heuristic were initialized as 1 and both players' minimax searches were one-ply for speed. Figure 2 shows the progress of one of the weights in the heuristic function. The win-loss differential of the learner was tracked over 25 games to see if the learner would start winning more and more as it underwent the learning process. Figure 3 shows the progress of the learner's win-loss differential.

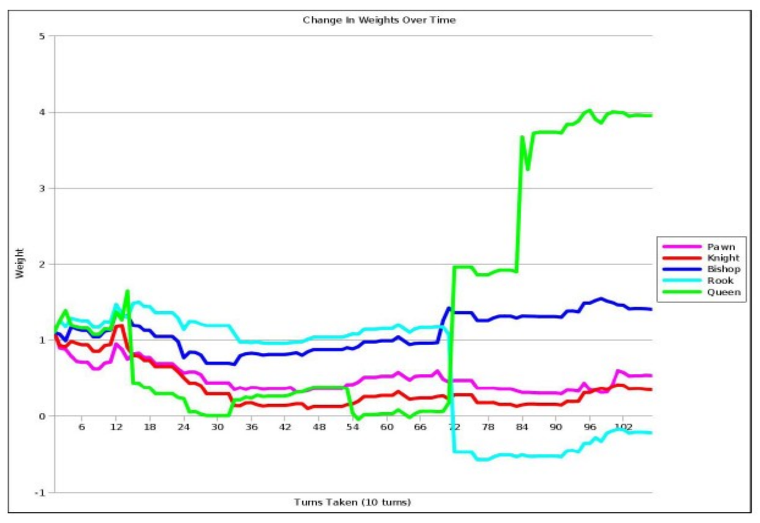


Figure 2: The changing weights in the evaluation function.

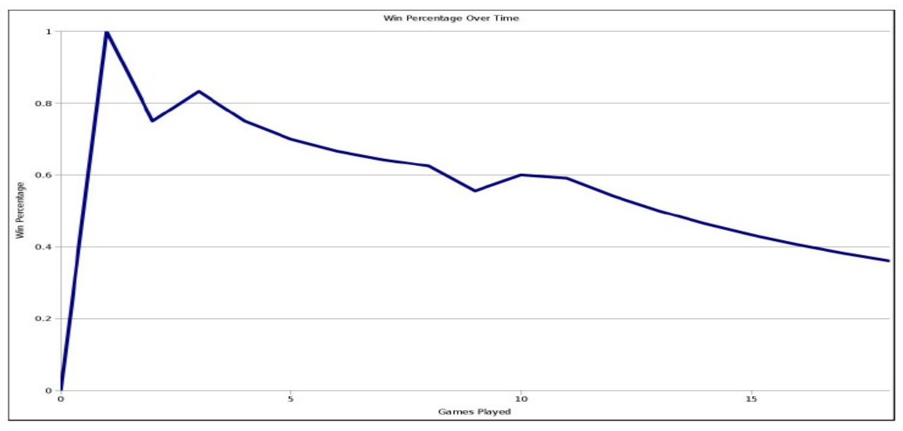


Figure 3: The win-loss differential of the machine learner over time.

3 Conclusion

From the testing results it is clear that Temporal Difference learning is an effective way to reach an evaluation function with equilibrium weights relatively quickly and without the program requiring prior knowledge about chess strategies. However, the accuracy of these weights is clearly questionable, as Figure 3 shows a decline in performance as the weights become adjusted. It is most likely that the bad weights were due to human error in the learning or minimax algorithms. Beal and Smith used a very similar learning algorithm which learned weights that performed very well [3], so the discrepancy between our results would be best explained by coding errors on my part. Because of this, I have deemed this study's efforts to have a program learn how to play chess a failure. Future research would include attempts to improve upon the learning algorithm used in this study to achieve better weight results and to apply the Temporal Difference learning algorithm to different domains.

References

[1] Russell, Stuart and Norvig, Peter. *Artificial Intelligence: A Modern Approach.* Second Edition. 2003.

[2] Shannon, Claude E. "Programming a Computer for Playing Chess". 1950.

[3] Beal, D.F. and Smith, M.C. "Temporal Difference Learning for Heuristic Search and Game Playing". 1999.

[4] Huang, Shiu-li and Lin, Fu-ren. "Using Temporal-Difference Learning for Multi-Agent Bargaining". 2007.

[5] Asgharbeygi, Nima, Stracuzzi, David and Langley, Pat. "Relational Temporal Difference Learning".

[6] Moriarty, David E. and Miikkulainen, Risto. "Discovering Complex Othello Strategies Through Evolutionary Neural Networks".