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

June 9, 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 players 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 Introduction

Heuristic searches (such as the A*) in general can be applied to practically any domain that somebody would want to search through. The actual heuristic functions used, however, are extremely domain-specific. Some problems require very simple heuristics (or estimations that allow a program to make good choices that will lead to the best outcome without actually knowing the entire search tree), such as a problem whose domain involves points on a flat surface that are connected by paths, in which the shortest path from point A to point B is desired. Such a problems heuristic would simply be the distance of a hypothetical straight path from the current point on the path to point B. By adding this estimation to the total distance traveled along a path thus far and comparing it to the total distances and estimated remaining distances of other possible paths, a search program can find the optimum solution to this problem quite easily. This is an example of a domain with a very simple heuristic to calculate. However, many problems that researchers are interested in nowadays have much more complex domains, and therefore much more complex heuristics. The age-old game of chess is one such problem.

It is possible to write a computer chess program that is effective at playing chess without having to search through the whole game tree (also known as a “brute-force” algorithm). By looking only, say, two or three moves into the future rather than 10 moves or more, a program can still make educated decisions about good moves to make if it has some way of estimating how strong a projected position would be. This is where a heuristic, or evaluation function as it is more commonly called in the context of board games like chess, comes in handy. By looking a few moves into the future, applying the evaluation function to each possible board, and choosing a move based on which projected board has the highest strength of position, a computer can still be an effective chess player, while at the same time dramatically cutting down the number of required calculations.

It sounds simple enough, but once you actually try to create such an evaluation function, it becomes much, much harder. How do you effectively evaluate a position? Do you just look at the number of your pieces versus the number of their pieces? Do you look at whether or not your pieces control the center? Do you look at the possibility of sacrificing one of your own pieces in order to capture a more important piece? An evaluation function can be very complicated to formulate, especially in a game such as chess where there are so many strategic factors to take into account. Chess masters spend entire lifetimes figuring out the best way to evaluate which moves they should make, so at first creating such a function could seem very daunting, especially for someone who has not devoted years to learning the game of chess.

But if chess masters learn how to decide which moves to make by playing a lot of games and learning from their mistakes, why couldnt a computer do the same thing? In fact, computers, although they lack human intuition, can play full games of chess in a matter of seconds, so it is conceivable that they could learn how to effectively evaluate moves much faster than a human could. Therein lies the purpose of this study: to see whether or not a computer can be programmed to not only play chess effectively and efficiently, but also learn from its mistakes, like humans do, and get better from game to game. The chess program I have developed uses Temporal Difference learning to modify its evaluation function every turn by looking at how modifying a giving weight will improve the predicted chance of winning the game. This means that theoretically, the program could start out by making random moves, but then modify its evaluation function so that it progresses from simply choosing a random move to actually making an educated choice.

2 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. [2]

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 performed decisively better over 2000 games than the unlearned program, showing that it is indeed possible for computers to improve their evaluation functions. [3]

Similar studies have been done in other areas. In 2007, Shiu-li Huang and Fu-ren Lin used Temporal Difference learning to teach two agents how to effectively negotiate prices with each other so that agreements that maximize payoff and settlement rates for both parties. TD learning was shown to be very effective for this purpose [4]. One of the reasons TD learning was chosen from the many different types of learning for this project was its usefulness in different domains. If it proves to be effective in chess, in addition to bargaining, then it could very well be effective for any problem in the very general and vast field of artificial intelligence. In another study called "Relational Temporal Difference Learning", researchers Nima Asgharbeygi, David Stracuzzi, and Pat Langley developed a type of learning called Relational Temporal Difference learning, which fuses the adaptability and effectiveness of Temporal Difference learning with the speed of Relational Reinforcement learning (a type of learning that generalizes the learning rather than making it move-specific). They tested their learning fusion on a series of games with well-defined rules including Tic-Tac-Toe and MiniChess and showed that the learning worked well for every game they threw at it. Like the bargaining study, this demonstrated that TD learning is a very versatile type of learning that could have widespread implications in the realm of artificial intelligence. [5]

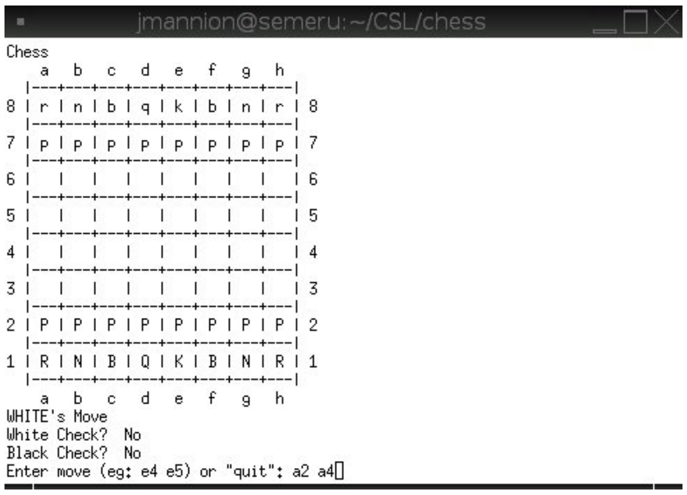
Researchers David E. Moriarty and Riso Miikkulainen from the University of Texas at Austin used a different type of machine learning called evolutionary neural networks to create programs that could play Othello. They found that even though their programs were given no information about the game, they eventually learned very complex Othello strategies that have only been mastered by high-level human Othello masters. [6] It is very interesting that a computer would naturally learn strategies in a matter of days that took humans years and years to conceive. In this project I will analyze the fruits of TD learning to see if well-known chess strategies naturally emerge over the course of testing.

3 Development

Python was used to code this chess program. 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 in Figures 1-3.

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.

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. A minimax search is a type of adversarial search which uses a heuristic function to not only choose which move the program thinks is best, but to guess the move that the other player will make during their turn. A three-ply search is one in which the program looks three moves into the future



```
jmannion@semeru:~/CSL/chess
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.

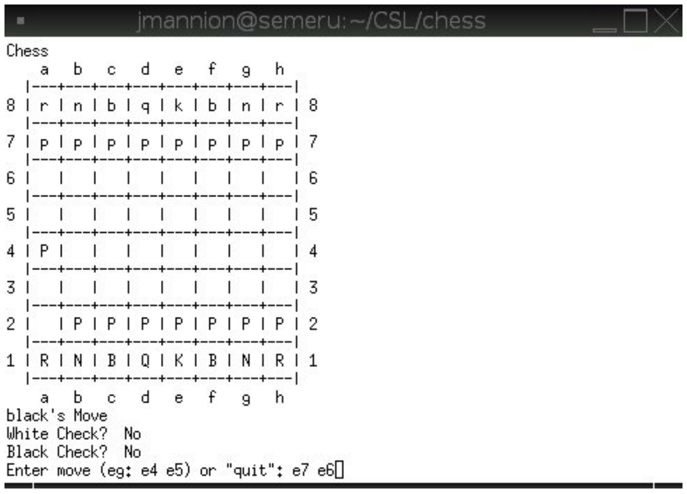
before making it's movement choices. Its heuristic function was such that it would chose moves based on a simple piece differential, with each piece being weighted the same as every other piece, regardless of position on the board. These two programs were created for the purpose of comparing the progress 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 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 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. The TD learning algorithm that was developed in this study worked like so: given a term in a heuristic function $w_i x_i$ where w_i is the weight of the term and x_i is a measured value that the heuristic is dependent on, the weight will be modified in the following manner (similar to [3]):

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

$$a = \frac{200}{199 + t} \quad (2)$$

$$P = \frac{1}{1 + e^{-h}} \quad (3)$$



```
jmannion@semeru:~/CSL/chess
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 | P |   |   |   |   |   |   |   | 4
  |---+---+---+---+---+---+---+---|
3 |   |   |   |   |   |   |   |   | 3
  |---+---+---+---+---+---+---+---|
2 |   | 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
black's Move
White Check?  No
Black Check?  No
Enter move (eg: e4 e5) or "quit": e7 e6
```

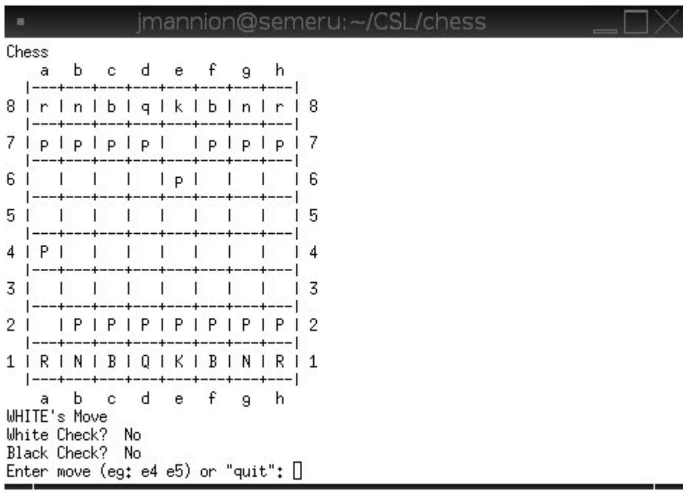
Figure 2: White pawn moved, black about to move a pawn.

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

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

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.

As the program progresses each timestep (or each turn for the learning player), the weights w_1 through w_5 will be adjusted, hopefully for the better. This means that board states will be evaluated based on the relative numbers of pieces each player controls and the value of each type of piece with respect to the other types of pieces.



```
jmannion@semeru:~/CSL/chess
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 | 7
  |---+---+---+---+---+---+---+---|
6 |   |   |   |   | p |   |   |   | 6
  |---+---+---+---+---+---+---+---|
5 |   |   |   |   |   |   |   |   | 5
  |---+---+---+---+---+---+---+---|
4 | P |   |   |   |   |   |   |   | 4
  |---+---+---+---+---+---+---+---|
3 |   |   |   |   |   |   |   |   | 3
  |---+---+---+---+---+---+---+---|
2 |   | 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": 
```

Figure 3: Black pawn moved.

4 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 4 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 5 shows the progress of the learner's win-loss differential.

5 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 5 shows a decline in performance as the weights become adjusted. It is possible that given a longer run period, the weights would have become better, however this is doubtful since Figure 4 shows that they were reaching equilibrium status. It is also possible that had the program been run

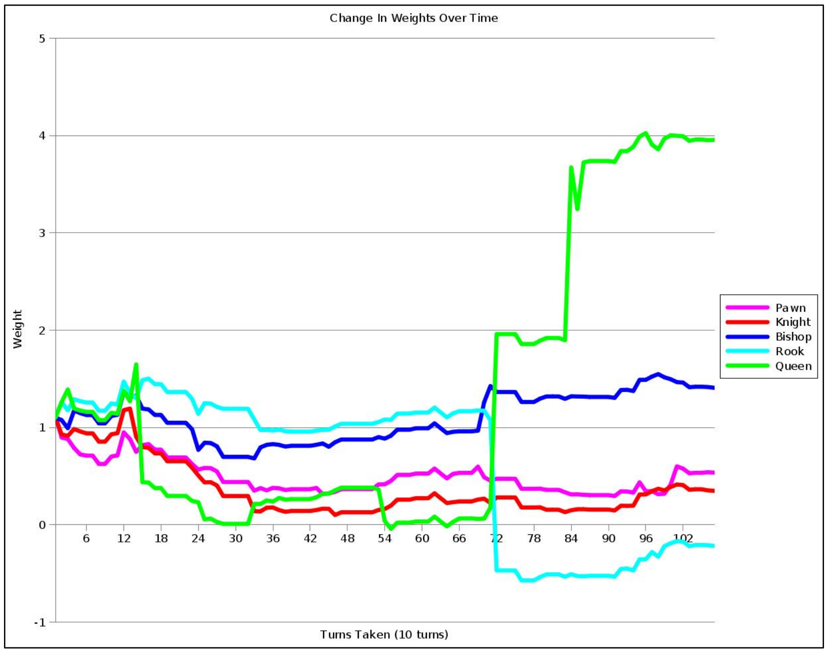


Figure 4: The changing weights in the evaluation function.

at three-ply, the extra information of future boardstates would have yielded better results. However, 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 excellent weights that performed better than the generally accepted weights of 1, 3, 3, 5 and 9 (for pawns, knights, bishops, rooks and queens respectively) [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, such as other games like Othello and Go, or other problems that require searching through large sets of data. Temporal Difference learning has a lot of potential as a solution to many problems in the realm of Artificial Intelligence because it is relatively fast and is not very memory-intensive. Further research with TD learning would be very beneficial to the advancement of game-playing and search techniques.

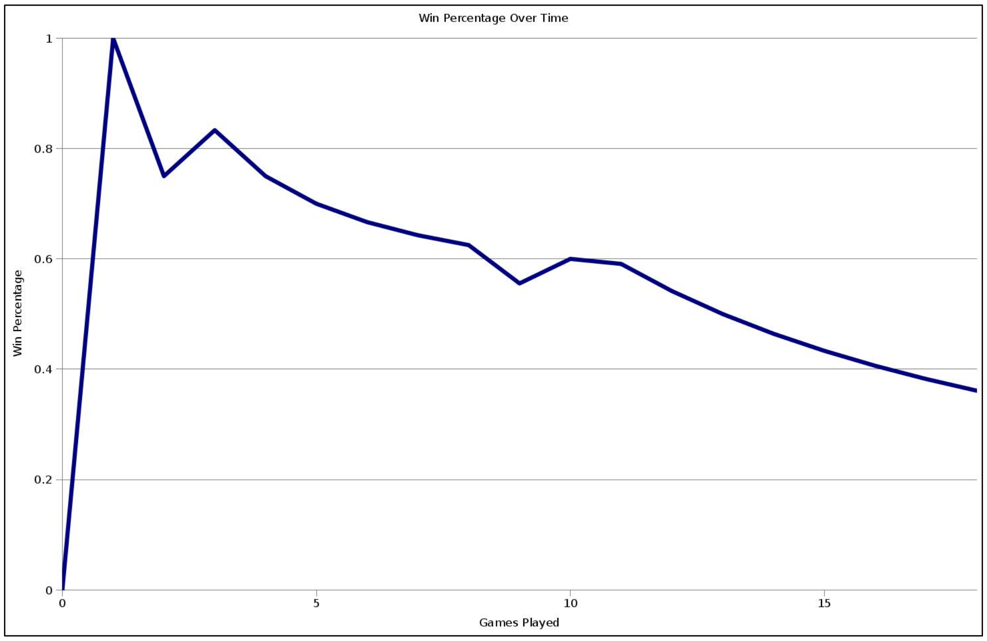


Figure 5: The win-loss differential of the machine learner over time.

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".