MASTERING THE GAME OF CHESS THROUGH A RE-INFORCEMENT LEARNING ALGORITHM

The game of chess is the most widely-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques and hand-crafted evaluation functions the have been refined by human experts over several decades. In contrast, the AlphaZero program, by DeepMind technologies, defeated stockfish, one of the world's strongest programs, through reinforcement learning from self-play. Again, AlphaZero's sister, lLela Chess Zero, defeated stockfish through reinforcement learning and stands as current reigning champion. AlphaZero, starting from random play to achieve superhuman performance within 24 hours and convincingly defeated stockfish, the then world chess champion.

However, these programs can only run on very expensive computers that have expensive graphical processing units, making themselves out of reach of many poor chess players. We therefore present a chess program that uses a reinforcement learning algorithm that runs on inexpensive computers and phones, like any ordinary chess program and guarantee to challenge the two programs that have never seen defeat: AlphaZero and leela Chess Zero.

In this paper we present a reinforcement learning algorithm, that can be plugged in to any game playing program. We also extend the work by, Weinstein Littman Goschin, who implemented Forward Sparse Sampling Search (FSSS) Algorithm in the minimax framework. We address the practical issues of his implementation and show that our implementation outperforms the original program. In the rest of text we will refer to our FSSS algorithm implementation as a reinforcement learning algorithm.

In our setting, we added to stockfish, the currently strongest alphabeta chess program, a reinforcement learning algorithm. The problem with Weinstein Littman Goschin's implementation is that it requires a lot of computation and many samples of same state for several number of moves. Our implementation, does not sample the states. We collects the samples differently: as our program is playing, its collects the samples. The information in the sample is simply

- The board state represented by board signature
- The move made
- Minimax score
- The depth the move was calculated

The samples are store in the hard disk and main for reuse. In the next game, or any later, if a position is in the samples, we can use the information to make pruning decisions. In this way, we eliminate the computational requirements for FSSS to collect the samples. As the number of games increases also, the samples increase. The pseudocode is shown below:

Move think(){

- When time is up
- Store the following:
    - The board state represented by board signature
    - The move made
    - The depth the move was calculated
    - Minimax score
- Return move

}

During search:

int alphabeta(){

// code to probe the samples table

- Check if the position is in the samples
- If the current depth <= sample depth
    - Return the score of the score of the sample
- Else
    - Try the sample move as the first move

This is only a few lines of and that's it:

Experimental Results

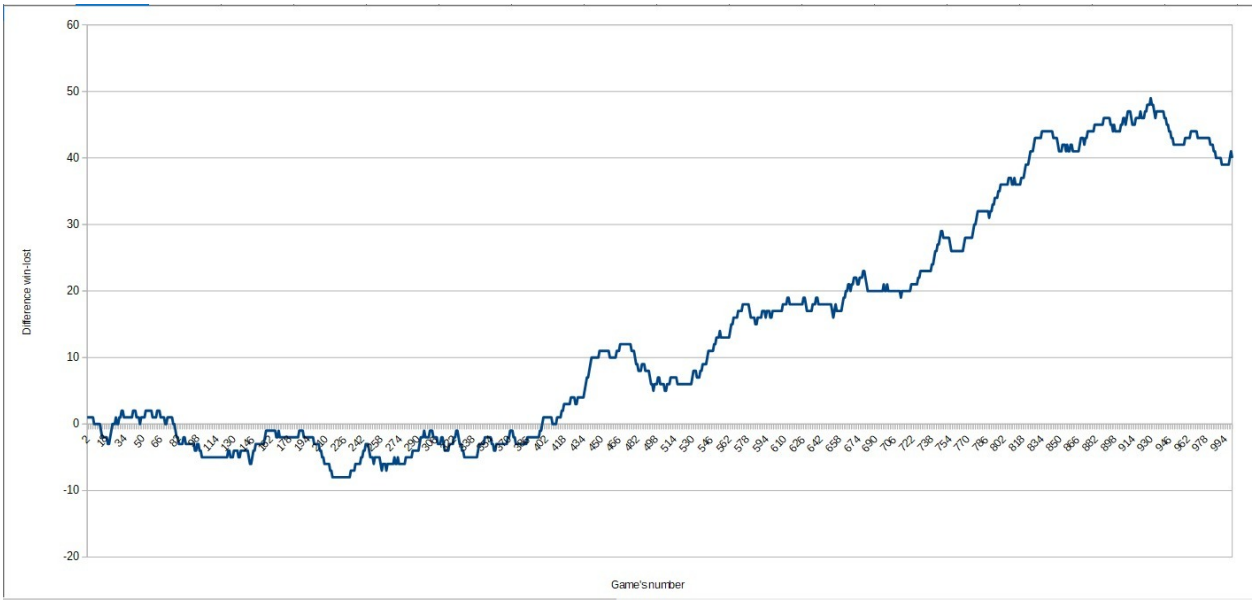
We conducted a 1000- bullet game match against stockfish with our program called brainlearn on one minute time controls.

The results are:

- Brainlearn: 163 wins, 123 losses, 714 draws
- Stockfish: 123 wins, 163 losses, 714 draws

This is an elo gain of 14 points in just one thousand games.

Here is the graph for the win rate vs. number of games for brainlearn:



The graph shows the win rate increases as the number of games are increasing. After 700 games the win rate continues to increase.

Improved Algorithm

The algorithm was further modified to capture the moves made for both players during the game. The captured game moves were used as priors during sub-sequent games to improve search efficiency of the alphabeta algorithm. We then run a 1,000 game match between the two algorithms. To our surprise, the new algorithm beat the first version by 503 points to 497!

Conclusion

- We have shown that our FSSS implementation is practical.
- We have shown that our implementation is a reinforcement learning algorithm.
- We have that the FSSS algorithm needs only a single sample in order to work.
- We have shown that this algorithm can challenge any program like alphaZero.

Further Research

The new algorithm can capture positions from any pgn files in an offline setting to capture the priors for use during game playing. There are thousands of pgn files around the internet. The idea is to feed the program with thousands of high quality pgn games to see if we can produce stronger chess-playing agents, thereby evolving the battle for supremacy in computer chess!

References

1. Stockfish chess: https://stockfishchess.org
2. FSSS algorithm by Weinstein Littman Goschin (2012) https://pdfs.semanticscholar.org/7f6f/a81467f098101aa410128728aa1ea608011f.pdf?_ga=2.139390835.1186788769.1568111602-1271337885.1568111602