Machine Learning in Computer Go

Dept. of CIS - Senior Design 2014-2015*

Jonathan P. Chen
jonchen@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

ABSTRACT

The focus for our senior project machine learning techniques in computer Go. Due to the strategic complexity and the sheer number of possibilities, Go has posed to be an area of scientific interest for both artificial intelligence research and machine learning research. For our thesis, we will focus on improving specific approaches to the game. Our objective is to implement existing algorithms then improve upon these existing techniques in measurable ways.

1. INTRODUCTION

Since the conquest of Deep Blue over Chess Grandmaster Gary Kasparov in 1996, the field of computer games, that is, using computers to play board games, has grown dramatically. In the last decade, many traditional games have been solved, including, Chess, Reversi (Othello), Checkers, and Backgammon. Computer Go remains the last game in which the top computers remain inferior to the top humans.

Go's origin goes back more than 3000 years, when wars were occasionally settled with a game of Go. Over time, it has grown to be an international mind sport, with hundreds of millions of players [6] Go is a zero sum game where the object of the game is to conquer the most territory on the game board. The game is played sequentially on a 19 x 19 board where players alternate placing stones in order to claim territory. A territory is "captured" when there is a closed string of pieces surrounding the contour of the space. The game ends when the board is filled or both players "pass" consecutively. There are various rulesets including the Tromp-Taylor, Japanese, and Chinese variations. For our purposes, we will stick with the traditional rules, as outlined above.

Go has been the great challenge of the twenty-first century for a variety of reasons. At first glance, Go possesses many of the same characteristics as Chess and other popular games, including being zero sum, deterministic, and perfect information. On the other hand, the complexity of Go far surpasses that of other games. Whereas in Chess has a space complexity of 10^{50}, a full Go board has roughly 10^{160}. This corresponds to a game tree size of roughly 10^{400} compared to the 10^{123} of Chess.

The key aspect is that Go is exceedingly difficult in strategizing in the early game and the midgame. Whereas in Chess, the endgame is more complex since the board has more free space as more pieces are captured, Go works in the opposite way in that the number of available locations gets smaller as the game progresses. Also, global versus local interactions are unique in Go. What happens in one area of the board is completely irrelevant to what goes on in other areas until the mid to late game, which makes it very difficult to optimize strategies in the midgame. Another impediment is the lack of a clean way to assign board evaluations. In Chess and other board games, game scenarios are ranked by how strategically effective they are. In Go, this proves to be a borderline impossible task since the significance of the placement of a single stone could not be apparent until the endgame.

Game	State Space	Game Tree
Checkers	18	54
Chinese Chess	48	150
Chess	50	123
Go	160	400

Table 1: Complexity Table: Note all the numbers are exponents of base 10

Irving John Good, a contemporary of Alan Turing at Bletchley Park, famously wrote in 1965, "In order to programme a computer to play a reasonable game of Go, rather than merely a legal game - it is necessary to formalise the principles of good strategy, or to design a learning programme. The principles are more qualitative and mysterious than in chess, and depend more on judgement." [5] Perhaps Go is a game where computers are just inherently slower than humans because the foresight relies on pattern recognition rather than optimizing steps locally. A human may never rival a computer in terms of computing power, but a computer likewise has a hard time competing with humans on creativity and pattern recognition.

2. RELATED WORK

Historically, creating a competitive Go player has been viewed as a really challenging problem in the field of artificial intelligence. The first attempt Go program was written by Albert Zobrist in 1968 which relied on pattern recognition by using an influence function to estimate territory by segmenting the board into black and white territories by assigning a numeric value for every point on the board, and his eponymous hashing technique (Zobrist hashing) to detect 'ko' situations on the board.[10] While this was a good initial foray into creating a Go AI, it did not achieve any noted successes against human players.

Ryder's work followed Zobrist's work in 1971 and he added the use of lookahead strategy to make decisions. He viewed

*Advisor: Mitch Marcus

Go as a cumulation of short term strategies which prevent the loss of critical stones to the opponent and long term strategies which result in the acquisition of territory. The only known game result from Ryder's player is a loss to a novice who did not have much experience. [1]

Even in 1998 professional Go players were able to beat Go AIs in spite of playing with handicaps of over 25 stones, which are significant handicaps for even strong human players. Finally, in 2008, MoGo won one game against professional Go player Catalin Taranu, a 5th dan player, on a 9x9 board. MoGo used a Monte-Carlo tree search algorithm which was optimized by using UCT (Upper bound Confidence for Trees) [8]. Till this point most AIs were designed using a minimax tree search using an evaluation function which is used to estimate the value of the potential move for the player. However, this traditional approach was not effective because of the difficulty in creating a good evaluation function for a Go board and also due to the high branching factor of a Go game tree due to the nature of the game and the large board size. However, techniques like alpha-beta pruning and Principal Variation Search can be used to optimize the naive minimax tree search.

Monte-Carlo tree search is one of the best strategies for creating a professional level Go AI. It's a type of game tree search that depends on a Monte-Carlo method based heuristic evaluation function. The Monte-Carlo method is used to simulate a large number of games, usually on the order thousands of random games, which begin at the different nodes of the game tree and are played out to their end by selecting moves at random and then the outcome of this 'playout' is used to assign a weight to the game tree node. In a pure Monte-Carlo tree search, we would not need to develop an effective evaluation function, a problem that we noticed with minimax implementations. More importantly, the Monte-Carlo tree search, due to its design, over time, focuses on the more promising parts of the tree, thereby reducing to some extent the problem caused by the high branching factor of Go[7]. The major disadvantage with Monte-Carlo tree search, however, is that due to its random nature it can potentially miss out on game-changing moves, especially those in the later stages of the game. Inherent in the nature of the game of Go is the choice between exploration and exploitation, in a move or a sequence of moves, you can either explore newer parts of the board or you can attack or defend your existing areas. This choice is also apparent in Monte-Carlo tree search. This is where we can use Upper Confidence bound for Trees of UCT to guide our search in a manner that balances exploration and exploitation. Using UCT we choose in each node of the game tree the move, for which the expression: $\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$ has the maximum value[2]. The different variables in this equation represent the following:

- w_i stands for number of wins after the ith move
- n_i stands for number of simulations after the ith move
- c is the exploration parameter, chosen empirically
- t stands for the total number of simulations in a given tree node, equal to the sum of all n_i

In 2008, Crazy Stone, a Go software AI, beat a 4 dan professional, receiving a handicap of eight stones. Crazy Stone combined tree search with Monte-Carlo evaluation that does not separate between a min-max phase and a Monte-Carlo phase [4]. Crazy Stone, like MoGo, also uses the UCT optimization technique on top of a Monte-Carlo tree search.

There has been a recent trend towards using neural networks to study a wide range of games and optimizing strategies based on that observation. Neural networks essentially establish a based on tactical patterns, and integrating that pattern across the entire board [9]. Though this has been proved promising in a couple simulations on a reduced board size, this has yet to be tested against professionals on a standard 19x19 board.

Furthermore, in March 2013, Crazy Stone beat a 9-dan player in a 19x19 game with just four handicap stones. Crazy Stone has been improved in a lot of ways over the years, one of which is learning move generation patterns from from a large data set of past games in order to compute a probability distribution over legal moves[3]. This approach is the closest related work to our approach.

3. PROJECT PROPOSAL

Now is the time to introduce your proposed project in all of its glory. Admittedly, this is not the easiest since you probably have not done much actual research yet. Even so, setting and realizing realistic research goals is an important skill. Begin by summarizing what you are going to do and the expected benefit it will bring.

3.1 Anticipated Approach

The first step for us is to assemble a training database. This database would take the form of a list of files that detail every single move that was made in a game of Go, as well as the board state at each point. This database is necessary for any algorithms we might implement that rely on machine learning techniques, as well as the algorithm outlined here.

Following this, we will need to generate a Markov model [3] representing conditional frequency distributions of moves. The exact method of generating the model is something we have yet to decide on; however, techniques that we plan to explore include weighting the probability of moves based on whether the player that made that move won the game and identifying moves that won games based on moves made by the enemy.

The next step would be using the Markov model to develop an algorithm that would, given a board state, determine the optimal next move. This algorithm may also use more specific details of the game such as the opponent's most recent move(s). Exactly what information the algorithm uses depends on the exact approach we take when building the Markov model.The algorithm would obviously be probabilistic in nature, which is important for the next part of the overall Go algorithm.

After this, we will need to develop a heuristic that can be used to score different board states. The exact nature of this heuristic is something that we will have to determine when we have a more sophisticated understanding of Go strategy. The heuristic should be determined both strategically

The final step of our anticipated Go algorithm is using the algorithm above to determine several series of optimal next moves, score each final board state using our heuristic, and determine which initial move is best (based on its final board state's score). This will be the next move chosen by our Go player.

3.2 Technical Challenges

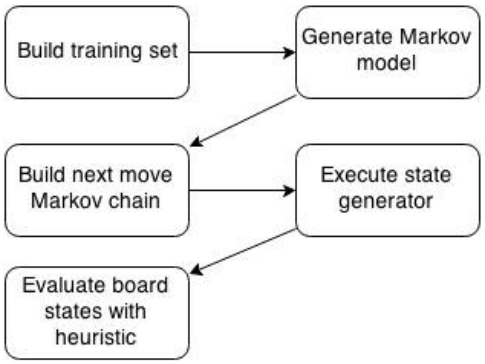

Figure 1: Block Diagram of Projected Plan

There are inherent technical challenges involved in machine learning and in the nature of the game Go itself. First, the large board size is currently the greatest impediment to solving the game, as there are over a googol possible game choices. We will focus our optimization efforts on 9x9 and 13x13 Go board sizes. Also, machine learning techniques require an ample set of training cases to work off of. The effectiveness of any given algorithm is limited by the level of the games in the training cases. Using a machine learning algorithm to improve upon an existing AI method, such as Monte Carlo methods in a significant way is also nontrivial because existing algorithms are effective to varying degrees and improvements will not easily result in noticeable global change. To use machine learning algorithms such as Markov Models or neural networks, we would need to come up with an heuristic would have to be in place to determine which moves are more effective than others.

3.3 Evaluation Criteria

In order to evaluate the performance of our Go player, we will use a number of different metrics.

First, we will measure the percentage of games, on average, it wins against our implementation of the existing Go strategy that we choose. We will compare this with the percentage of games the existing Go player wins against itself for a sense of how well our player does. We will run tests on both the 9x9 and 13x13 versions of the Go board in order to determine which algorithms perform better on which boards.

Also, we will measure the time it takes to compute a move on average. Obviously, this is a secondary metric, but still an important one, as many existing algorithms take astronomical amounts of time to run even on supercomputers.

4. RESEARCH TIMELINE

- ALREADY COMPLETED: Familiarized ourselves with the main strategies used by computer scientists to solve Computer Go.
- BY MID-NOVEMBER: The next step will be to determine which of the strategies we want to implement. We will choose one or more strategies and implement them, along with a functional Computer Go simulator. We will use these strategies to develop improvements and optimizations to those strategies that result in a better Go player, and they will also serve as benchmarks against which we can compare our new strategies. By this point we will have learned enough about the currently used algorithms to be able to analyze and refine our anticipated approach.
- PRIOR-TO CHRISTMAS : Our goal is to have finished building a basic Go player based on existing strategies, as well as a functional Computer Go simulator. We should also have a refined version of our anticipated approach, as well as the details of what heuristics we plan to use for it, formulated to the point where it is ready to be implemented.
- COMPLETION TASKS : Fully functional implementation of an algorithm to play Go.
- IF THERE'S TIME : Since we want to explore machine learning techniques that we can use to improve our Go player, we will assemble some sort of training database for the Go player. This would consist of move histories of played out games. We also might investigate the implications of using neural networks to concoct new strategies by studying a large database of existing games and strategize based on successful strategies.

5. REFERENCES

[1] L. E. Baum. Statistical inference for probabilistic functions of finite state markove chains. *The Annals of Mathematical Statistics*, pages 1554–1563, 1966.

[2] Guillaume M. J-B. Chaslot, Mark H. M. Winands, H. Jaap van den Herik, Jos W. H. M. Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4:343–357, 2008.

[3] Remi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. `http://remi.coulom.free.fr/CG2006/CG2006.pdf`, 2006.

[4] Yizao Gelly, Sylvain; Wang. Modification of uct with patterns in monte-carlo go. Technical report, INRIA, 2006.

[5] I Good. They mystery of go. *New Scientist*, 1965.

[6] Brett Harrison. Move prediction in the game of go. `http://www.eecs.harvard.edu/econcs/pubs/Harrisonthesis.pdf`, April 2010.

[7] Levente Kocsis and Csaba SzepesvÃąri. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[8] J Ryder. Heuristic analysis of large trees as generated in the game of go. Technical report, Stanford University, 1971.

[9] et al Wu, Lin. Learning to play go using recursive neural networks. Technical report, University of California, Irvine, 9 August 2007.

[10] Albert Zobrist. *Feature Extraction and Representation for Pattern Recognition and the Game of Go*. PhD thesis, 1990.