Artificial Intelligence In Go

Kevin Yang
Chaparral High School, California,CA 92591

Abstract: In the recent decade, artificial intelligence of all sorts have revolutionized the highly complex combinatorial game of Go. The once thought impossible task of a computer beating human experts in Go was accomplished by Google DeepMind's AlphaGo, the first successful computer in Go. This study explains conceptually how AlphaGo works, as well as demonstrates the effect of the machine's internal features and settings on the final results. After discussing the internal networks and algorithms of AlphaGo, the goal is to observe the playout value at which decision stabilization is reached. To achieve this, experiments were done based on the three stages of the game: opening, midgame, and endgame. Decision stabilization will be measured as parallel to win rate stabilization. Three complete games (neither side resigns) are analyzed over a course of 500,000 playouts and then 1,000,000 playouts. These playout capacities correspond to an original model (2,801 playouts per second) and an improved model (6,000 playouts per second). A move is selected at an uncertain point in the game for playout simulations. Experiments with the different models indicated that decision stability is achieved at around 200,000-300,000 playouts. Optimal time will vary depending on the speed of the model. Overall, this information allows us to better understand artificial intelligence in Go and allow for possible improvement in the field. It also allows us to determine the quantity of playouts to obtain the optimal reliability. This will save sizable amounts of time and computational power.

Keywords: Artificial Intelligence; Go, AlphaGo; Decision Stabilization; Win Rate; Playouts; Simulations; Optimal Reliability; Computational Power; Networks; Algorithms; Machine Learning

Introduction

Go, also called Baduk or Weiqi, is the oldest board game in the world. With approximately 4,000 years of age, it originated in China around 2356 B.C.E. Today, it is the most popular game in the world statistically with over a hundred million players. However, the game is still rather unknown in the west and mainly consists of players in Asia. Along with being the oldest game in the world, Go also takes the spot for being the most complex game.

With thousands of years of history behind Go, evolution is inevitable, especially in recent years within the field of AI. AlphaGo uses a variety of machine learning techniques in order to achieve its playing level. One key process is simulation, a process that requires much time and computing. However, is there an

optimal point where we can achieve the most reliable results along with the least time and computational power?

Background

When it comes to categorizing Go into a type of game, Go can be put into the combinatorial games category. However, unlike some combinatorial games, there is no “one size fits all” winning strategy. There isn’t even a winning strategy at all, at least from a human perspective. Since Go is played on a 19x19 board, the number of variations is overwhelmingly vast. In fact, there are 10170 possible configurations, more than the number of atoms in the world.

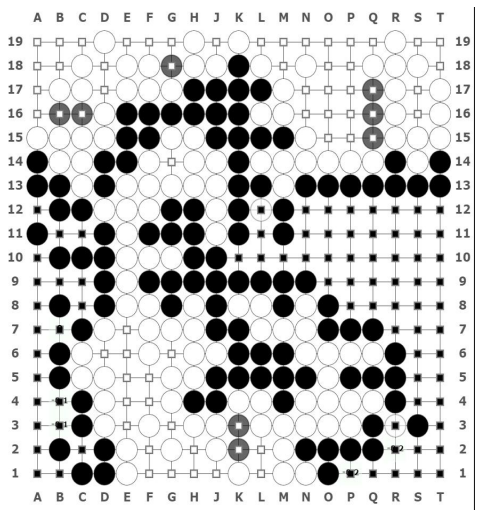

Figure 1: Finished game in the counting phase, each black or white square counts as 1 point while transparent stones are dead. In this game, white wins by 4.5 points

Amazingly, in recent years humans have found a way to in a sense create artificial intelligence capable of simplifying the complex game of Go into a game where a winning strategy exists. For example, Google’s DeepMind created AlphaGo, the first Go AI able to beat a professional human player. Although the preliminary versions may not be perfect, the structure and processes that they carry out were proven to be crucial in stronger versions later. Here, we will go over the first artificial intelligence created for Go which laid the groundwork and foundations for later versions: AlphaGo.

AlphaGo: An Overview

When it comes to artificial intelligence in Go, there are many different “AIs" that we can rely on. Some of these include KataGo, AlphaGo, LeelaZero, Golaxy, and much more. Although some may be stronger than others, all artificial intelligence (AI) in the field of Go perform the same basic actions that constitute a winning strategy. The biggest part of this winning strategy involves simulation or in other words, brute force.

Computers have calculational abilities far above that of humans, which is what gives them such a big advantage. Since humans cannot calculate as fast as a computer, we must rely on abstract thinking and shortcuts when we play. However, computers can take advantage of this computational advantage and utilize brute force to beat humans. Regarding this process, we call each simulation a “playout” in Go AI

terminology. AI will select a possible move and simulate a complete game, getting a result of either 1 or -1, which translates to win or loss, respectively.

Naturally, the more playouts AI is capable of computing, the stronger the AI will be. Bringing the conversation back to AlphaGo, AlphaGo consists of a combination of both supervised learning and reinforcement learning. A combination of artificial neural networks are implemented, which goes to show just how complex AlphaGo is.

AlphaGo: Training the Networks

Supervised Learning (SL) Policy Network

As mentioned before, AlphaGo combines numerous networks in order to achieve its goal. The first of these networks is the supervised learning policy network, a network with the role of carrying out the exploration of possible moves.

Fed with 30 million board positions played by professionals, the computer interprets the board position as a 19 x 19 x 48 input feature. To better understand this, it can be thought of exactly like the common input matrices that computers in machine learning take in. Due to the simple rules of Go, one board position can be played from many different directions (rotating the board). Because of these conditions, a game of Go is rotationally and reflectionally symmetric. This is ideal for training the network as it allows for data augmentation; eight symmetries can be generated.

Next, the backpropagation algorithm is applied to train the network. This is the exact same way normal neural network classifiers are trained. At the end of this process, the supervised learning policy network achieves a 57% accuracy with 50 GPUs and three weeks of training time.

Rollout Policy

The SL policy network has decent accuracy but it is too slow. This is why the rollout policy is needed. With only two neurons in each layer instead of three, the rollout policy is 1500x faster than the SL policy network. Although the accuracy is less than half of that of the SL policy network, the two can be combined to balance each other and carry out the job of exploration.

Figure 2: The rollout policy (left) has only two nodes in each layer as compared to three nodes in the SL policy (right)

Reinforcement Learning (RL) Policy Network

With the exploration aspect achieved already, we still need something to evaluate the possible moves. After all, the job isn’t done yet if the computer doesn't know how good the moves are. This is where the computer uses self-play to formulate higher accuracy networks. With reinforcement learning, the process is iterative, hence the term “reinforcement”. The SL policy network is duplicated and the computer iterates with the policy gradient reinforcement to improve the RL policy network. Within each iteration, AlphaGo

uses the RL network while its “imaginary” opponent uses an older version of the network. If the older version beats the newer version by a particular margin, the older version replaces the newer version.

After the game is finished, the parameters of the network are modified using backpropagation to increase accuracy. The version that wins continues to play games, staying as the “best” version until the older version beats it. This process takes one day with 50 GPUs.

Value Network

So what exactly is the RL policy network reinforcing in the first place? It is applied to optimize the move exploration network as well as something called the value network. The value network estimates the “value” or a certain board position. With a feature like this, the computer can determine how good a board position is. Paired with the RL policy network, better moves can be suggested. This process outputs a scalar value instead of a probability distribution and compares this scalar value to the result of the game in a cost function. The RL policy and value network combo plays 30 million games, selecting one position from each game as a sample for training.

Selecting the next Move

The computer can receive different advice, or suggestions, from these different networks and still needs to select the best possible move from these options. Next, the computer will simulate each of these moves and see which one will obtain the most wins. In order to keep track of these simulations, a search tree is created. AI will determine information for each move by this criteria (not in any specific order):

➢How good the move is

➢How good the board position is after the move is made

➢How many times AI chooses or visits the move

➢The simulated game result using the move

Monte Carlo Tree Search Algorithm (MCTS)

AI will use the MCTS algorithm to select the best moves. The commonly used heuristic algorithm consists of four parts: Selection, Expansion, Evaluation, and Backup.

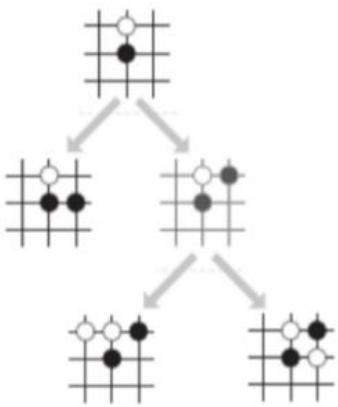

Figure 3: A small portion of the Monte Carlo Search Tree; each board position is a leaf node

As mentioned before, artificial intelligence in Go can exploit the nature of the board whilst training

and in this case, even evaluating. During MCTS, the board positions can be transformed randomly by rotating or reflecting in order to achieve the goal of averaging over different biases.

Selection plays the role of choosing which moves to play ahead of time. By the time the simulation process arrives at a later step, the move will already be prepared. However, this process will only be implemented on nodes that have the highest value, in other words, nodes that are promising. Expansion adds possible moves to the search tree as the next step of simulation. Each addition adds a leaf node to the tree using the SL policy network. This is because the SL policy is trained with human experience and human moves tend to be more abstract and diversified. In this step, the value network will be used for exploitation instead of the SL policy because it stores information from each leaf node, meaning that more information is being processed for higher accuracy.

Next, simulation/evaluation finishes the simulation and gets the result, either a win or a loss. In this step we will use the rollout policy instead of the others solely because speed is crucial in this step (rollout 1500x faster). Finally, we get to backup/backpropagation where the final move is decided by how many times it was visited/chosen in the simulation. This information is recorded and stored. Only the branches that emerge from our chosen move will be continued, all others will be discarded.

Method

Experimental Setup

AlphaGo is not open to the public, so we will need to use a different AI. In fact, the AI used is even better than AlphaGo: Golaxy. While using Golaxy, we can see how many playouts have been simulated at a certain time, making it easier to observe performance. Playout and win rate data were collected through actual games between professionals.

Data will be collected from the opening, midgame, and endgame using two models, a normal model and then an improved model. Three distinct games along with one “fair” move from each game was selected for the normal model. A fair move refers to when win rates are even for both sides and neither player has the advantage. The purpose of this is to eliminate bias from the existing situation of the game. Additionally, the overall situation of the game is close too, not just the situation in a particular move. In order to maintain consistency, we focus on the same move throughout all playout inputs in each trial. The same three games and moves will be used for the improved model.

Collecting Data

After setting up the experiment, we arrive at the point before the tested move is made. For example, if the tested move is move 89, we will arrive at move 88. Then, we will allow the AI to start simulation. Since the AI simulates playouts very fast, about 2,801 playouts per second, we will record the process and observe in slow motion.

Measuring the win rate at predetermined quantities of playouts (up to 500,000 and 1,000,000), we can gather enough data to construct multiple graphs. This will allow us to clearly see the performance and reliability of AI over time and playouts.

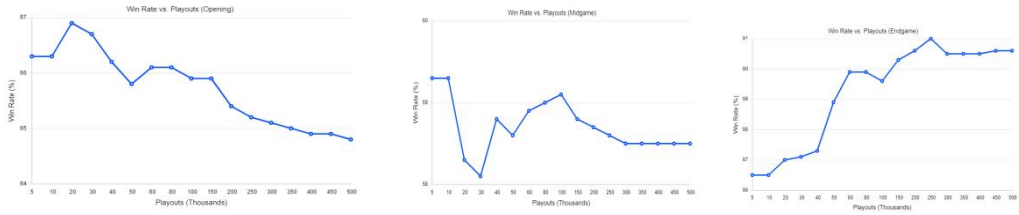

Figure 4: Experiment results for the opening (first), midgame (second), and endgame (third). The vertical axis represents win rate (%) while the horizontal axis represents playouts (thousands)

Next, we will use the improved model to verify consistency. To do this, playout capacity and speed can be increased; for the next experiment we will roughly double playout speed and capacity. The speed will then be approximately 6,000 playouts per second with the maximum number of playouts stopping at 1,000,000. All other variables and factors such as the move to be analyzed, the game in which the move is pulled from, and data collection method will remain constant.

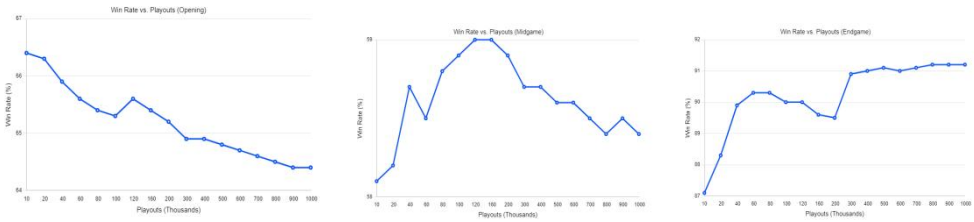

Figure 5: Experiment results with the improved model for the opening (top), midgame (middle), and endgame (bottom). The vertical axis represents win rate (%) while the horizontal axis represents playouts (thousands)

Analyzing Results

From these graphs, we can see that the win rate line starts to flatten out at about 200,000-300,000 playouts. We can draw the conclusion that the win rate prediction of a move stabilizes at about 200,000-300,000 playouts. Using our value of 2,801 playouts per second, this is equivalent to 1.19-1.79 seconds. Using our improved model of 6,000 playouts, this is equivalent to 0.56-0.83 seconds. This means that the optimal number of playouts for reliability is 200,000-300,000 after about 1.49 seconds and 0.70 seconds for the improved model.

Conclusion

In conclusion, having knowledge on the optimal stopping point will save good amounts of time and computational power. Understanding how AI in Go works can also provide insight into how improvements can be made. It can even be extended into other combinatorial games similar to Go such as chess, checkers, etc.

Although the AI used in this experiment was not AlphaGo, the values could be scaled to match the computational powers of different AIs. Since Golaxy is the strongest AI as of currently, of course AlphaGo and other AIs will have worse optimal playouts and time efficiency. However, later versions of AlphaGo such as AlphaGo Zero could have better optimal values.

AlphaGo Zero and Golaxy play moves strong enough to easily demolish any world champion. Now, AI has completely dominated Go, completely influencing the game in all aspects. Opening playstyles, strategies, and studying methods have all incorporated great amounts of AI. With AI growing as fast as it is in today's world, possibilities for further improvements for AI in Go are inevitable. Future versions of AlphaGo may very easily defeat the strongest AIs today.

References

[1] Britannica, Editors of Encyclopedia., 1998, go, [Online] (updated 24 Apr. 2017) Available at: <https://www.britannica.com/topic/go-game> [Accessed 14 Aug. 2022]

[2] Osínski, B., 2018, What is reinforcement learning? The complete guide, [Online] (updated 5 Jul. 2018) Available at:<https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/>[Accessed 14 Aug. 2022]

[3] Hui, J., 2018, AlphaGo: How it works technically? [Online] (updated 12 May 2018) Available at: <https://jonathan-hui.medium.com/alphago-how-it-works-technically-26ddcc 085319> [Accessed 14 Aug. 2022]

[4] YouTube, Simplilearn, 2020, Backpropagation in Neural Networks|Backpropagation Algorithm Explained for Beginners|Simplilearn, [Online] (updated 15 Nov.2020) Available at: <https://www. youtube.com/watch?v=GltT9b31fRY> [Accessed 14 Aug. 2022]

[5] GeeksforGeeks, 2022, ML|Monte Carlo Tree Search (MCTS), [Online] (updated 5 Jul. 2022) Available at: <https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/> [Accessed 14 Aug. 2022]

[6] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D., 2017, Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, [Online] (updated 5 Dec. 2017) Available at: <https://arxiv.org/pdf/1712.01815.pdf> [Accessed 25 Aug. 2022]

About the author: Kevin Yang (2007.07), male,American nationality, California, student,high school degree,Chaparral High School,research area :artificial intelligence machine learning.