Demystifying Alpha Go

MSc Essay Presentation

April 26th, 2018

Adnan Reza

a place of mind

UBC

Introduction

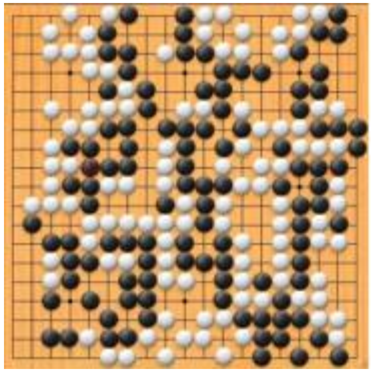
- While Go programs have been around since the 1970s, their performance has not come close to achieving professional level playing let alone defeating Go champions.
- The team of researchers at DeepMind were able to tackle this daunting challenge with their Go program: AlphaGo.
- AlphaGo’s journey towards achieving superhuman-level performance in Go came into the spotlight in April 2016 when it defeated Go world champion Lee Sedol.

Introduction

- While it is easy to be carried away by the enormous hype surrounding AlphaGo's achievement, it is a worthwhile exercise to delve deeper into how AlphaGo works and in the process demystify some of the hype surrounding this truly extraordinary achievement.
- This is the primary motivation of this essay. By looking deeper into AlphaGo's architecture, we can see that it is indeed a combination of techniques, some of which have been developed by researchers in the early days of computer Go, aided with state of the art hardware.

The Game of Go

- Go is played on a $19x19$ board with two players.
- One player plays using white stones while the other plays black stones.
- The board is initially empty; player 1 starts the game by placing one of his white stones in one of the grid intersections.
- This is followed by player 2 placing one of her black stones in an unoccupied grid point.
- Players are not allowed to move the stones once they are placed on the board.
- The game continues by players taking turns to put their respective stones in previously unoccupied positions on the board. A player can opt to skip a move, allowing the other player to effectively play two moves at once.



The Game of Go

- The goal of each player is to maximize their territory i.e. place as many stones on the board as possible.
- Each stone has *liberties* i.e. free adjacent cells. The adjacent stones of the same color form a *group*.
- A player can capture a group of the opponent’s stones by surrounding it with his or her own stones.
- This results in the removal of the whole group from the game board.
- The game concludes when both players have passed i.e. skipped their turns consecutively.
- The scores are then counted and each player is awarded points proportionally to the number of grid points occupied by each player.

Why is Go suited to a computational solution?

- Go is a sequential game of *perfect information*. This means that before making a move, each player is perfectly informed of all events that have previously occurred [5].
- More precisely, each player can see all the stones on the board throughout the duration of the game. This means that the outcome of the game is solely determined by the strategy of the two players [6].
- This fact makes Go an attractive problem to solve computationally - we simply need to design a program that solves for the optimal sequence of moves.

Why is Go challenging?

- One way to understand why Go is challenging to solve computationally, is to start by looking at two other classical games that are also of the sequential/perfect information variety.
- Chess and Checkers are two popular sequential games of perfect information which have achieved superhuman performance using computational solutions.
- Checkers and Chess have search spaces of 10^{20} and 10^{43} possible positions respectively [5, 7]. In comparison, Go has a search space of 10^{170} possible positions [8, 9], which renders exhaustive search infeasible.
- In other words, since Go has a large number of potential moves, it yields exponentially more ways for the game to unfold relative to chess and checkers [5].
- Another reason why Go is challenging to solve computationally is because it is difficult to accurately predict which player is more likely to win the game from any given board position. This limits the efficacy of simply applying machine learning approaches.

The Revolution: AlphaGo

- So, what makes AlphaGo so special relative to other Go programs? Looking deeper into DeepMind’s work [2], we can see that it is indeed to a large extent, a blend of tried and tested techniques, some of which have been developed by researchers in the early days of computer Go, aided with state of the art hardware.
- The first instance of a Go program leveraging the Monte Carlo method was Gobble in 1993 [15].
- This was followed by Crazy Stone, which was one of the first Go programs to incorporate MCTS which resulted in it outperforming most of its contemporary competition [16].
- Several other groups [8, 17, 18] have successfully applied CNNs in Go.

The Revolution: AlphaGo

- These important research contributions laid the groundwork for the creation of AlphaGo.
- While many of these techniques may not be novel in Computer Go, what makes AlphaGo truly revolutionary is the combination of these techniques i.e. their ensemble. We explore these techniques now.

Monte Carlo Tree Search (MCTS)

- Monte Carlo methods are a class of computational algorithms that rely on repeated random sampling to obtain numerical results [19].
- The main idea is to use randomness to solve problems that are deterministic in principle, but too complex to solve using other approaches e.g. when a problem is too complex to be solved analytically [5].
- An efficient way to solve the game search tree is to apply the Monte Carlo Method. This approach approximates a function using random sampling, where the mean of the samples converges to the real value of a function [5].
- The idea to integrate Monte Carlo simulations into the tree growing process, also known as Monte Carlo Tree Search (MCTS) was first adopted by the Go program Crazy Stone [16]. Ever since, this technique has become a staple in the computer Go community. We will look at the 4 different stages of MCTS later in the presentation.

Augmenting MCTS with CNNs

- However, MCTS on its own is not powerful enough to achieve the performance required. To address this, DeepMind augmented it with convolutional neural networks.
- To understand the role of CNNs in AlphaGo, we need to first discuss two concepts: local receptive fields and weight sharing.
- The concept of local receptive field means that each neuron is connected only to a small region of a fixed size in the previous layer [5]. The weight sharing assumption is: *while every neuron is responsible to its own subarea in the previous layer, the set of weights of connections (i.e. filter) is the same for all neurons.*
- We can think of each filter as a feature extractor and the more filters we apply, the more information we can obtain from the input.
- The learning task for the CNN was to obtain the weights within each filter.

Supervised Learning Policy Network

- The supervised learning policy network's goal is to *predict expert human moves.*
- Therefore, it makes sense that the input to AlphaGo's CNN is the current board setting [2] and the output is the prediction – the move a human would make.
- The team at DeepMind used ~160k games of Go professionals that were recorded on the KGS Go server [23].
- Random board positions in conjunction with associated player moves were selected from each game. The network's objective was to predict these actions.

Supervised Learning Policy Network

- The input to the CNN i.e. the current board setting was translated into 48 features. These features included important information like the color of the stone at each intersection and number of free adjacent cells (liberty) [2].
- To summarize the input layer of the CNN was a $19x19x48$ vector, which stored the value of every feature for each intersection of the game board.

Supervised Learning Policy Network

- The output layer was $19x19$, where each cell indicated the probability that a human player will put a stone in the corresponding intersection of the game board.
- This is the supervised learning portion of AlphaGo’s training. The supervised learning network is used at the selection stage of MCTS to encourage exploitation [2, 5].
- A good selection rule tries to find a happy medium between selecting known optimal moves and investigating unexplored moves.
- We can think of the CNN influencing MCTS to try out moves which have been scarcely explored but which seem good to the CNN.

Reinforcement Learning Policy Network

- Predicting human moves was never the end goal. It was to optimize the networks for maximizing the likelihood of winning and eventually reach superhuman level playing ability. This is where Reinforcement Learning comes into play.
- The system plays against itself, without training sets of games played by humans. This approach of self-play is not novel; it was popularized in the reinforcement learning community by Gerald Tesauro in the early nineties when he used it in TD-Gammon [22]. TD-Gammon reached the performance of the top human backgammon players by using this approach of self-play.
- Reinforcement learning [24] is a method of training an agent where it is not explicitly given the correct answer; instead the agent's objective is to maximize its reward. In the context of AlphaGo, the reward used was ± 1 for a win/loss upon termination of the game, and 0 for all other intermediate steps [2].

Reinforcement Learning Policy Network

- The key idea was that by letting AlphaGo play against other versions of itself, it can learn and get better at the game.
- Specifically, at a given iteration step, the current version of AlphaGo plays against a random instance of the supervised learning network from the previous iterations.
- The match ends and a reward is given, which tells the supervised learning network if its actions were correct and how the parameters of the network should be altered [2].
- AlphaGo’s supervised learning network was greatly strengthened using reinforcement learning here. While the reinforcement learning networks were stronger compared to the supervised learning networks, the best performance was achieved combining the two [2].

Value Network

- The value network has the same architecture as the supervised learning network, with the distinction that its job is: *given a position in the game to output a single value indicating a win or loss.*
- Additionally, the value network is not trained on human expert games like the supervised learning network; it instead uses the reinforcement learning network's games.
- In comparison to the supervised learning phase where ~160k games were used to train the networks, the reinforcement learning network was trained using ~30 million self-play games, using 50 GPUs for one day.
- This idea of approximating the value function by using reinforcement learning was used earlier by Michael Littman [25].

Exploiting symmetries in AlphaGo

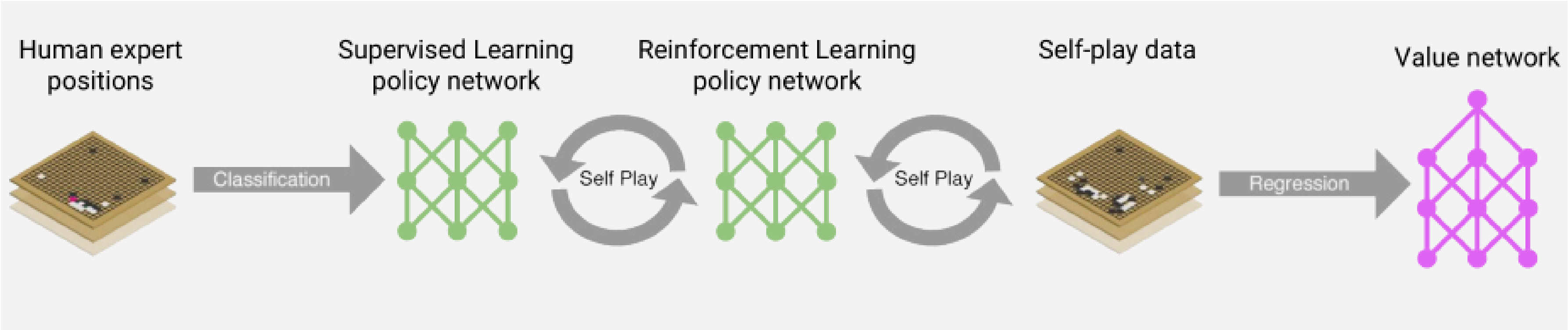
- CNNs operate by extracting local features from the input.
- While previous work [8, 17, 18] has shown that CNNs are particularly useful for Go due to translational invariance (evaluation of a position is not affected by shifting the board), the AlphaGo team argued that it hurt performance in large networks as it prevents the intermediate filters from identifying specific patterns.
- Instead, they exploited symmetries at run-time by dynamically transforming the group of 8 reflections and rotations (lines of symmetry is 4 + Order of Rotational Symmetry is 4).
- Batches of all 8 positions are passed into the policy/value networks and computed in parallel.
- For the value network the output values are simply averaged.
- For the policy networks, the planes of output probabilities are rotated/reflected back to their respective original positions and then averaged together to provide an ensemble prediction [2].

Summary of AlphaGo’s architecture

- To summarize, AlphaGo relies on a combination of MCTS and CNNs to guide the tree search procedure.
- We can think of the role played by CNNs in AlphaGo as somewhat like the evaluation function used by IBM’s Deep Blue [26], one notable distinction being the evaluation function for AlphaGo is learned and not designed or hand crafted [21]. The use of convolutional neural networks in AlphaGo provides a degree of intuition to the game-play [5].

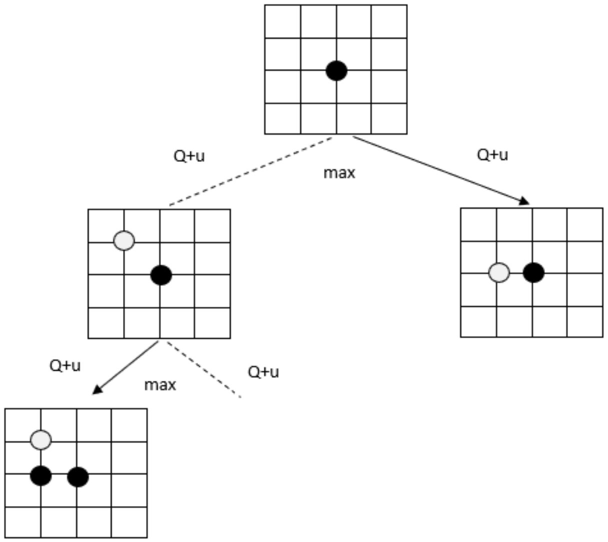
Summary of AlphaGo's architecture

- We can broadly classify AlphaGo's architecture into two phases: (1) Learning/Training and (2) Playing.
- ***Phase 1 – Learning:*** The first phase is where it essentially learns to play the game. This is done by training the neural networks on human games (SL) and self-play (RL). The output of this phase is the supervised learning, reinforcement learning & value networks.



Summary of AlphaGo's architecture (continued)

- ***Phase 2 – Playing:*** The second phase is where it plays the game by traversing the game tree using MCTS. Let's briefly see how this phase works:

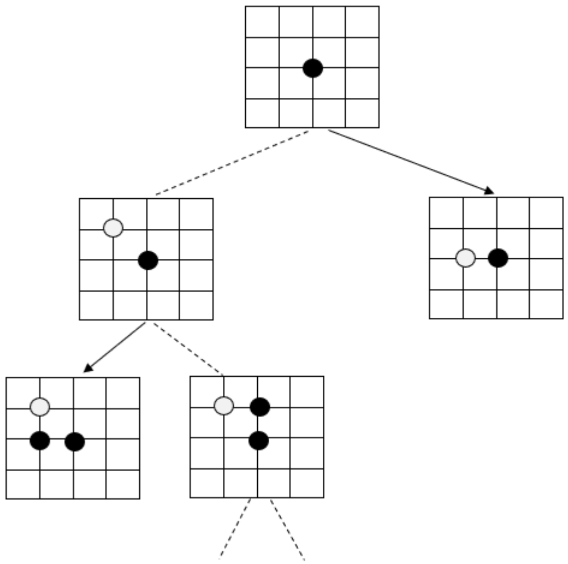


In stage 1 (***selection***) of MCTS, the edge with the maximum action value Q, plus a bonus u is selected.

The bonus u depends on the stored prior probability from the supervised learning network.

Summary of AlphaGo's architecture (continued)

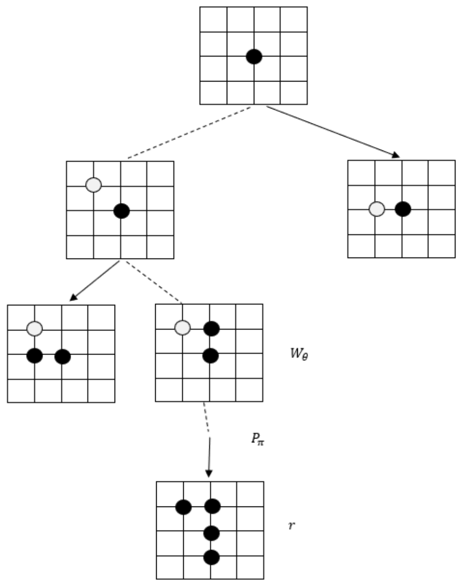
- ***Phase 2 – Playing:*** MCTS Stage 2



In stage 2 (**expansion**), the selected node is added to the game tree and the output probabilities are stored as prior probabilities for each action.

Summary of AlphaGo's architecture (continued)

- ***Phase 2 – Playing:*** MCTS Stage 3

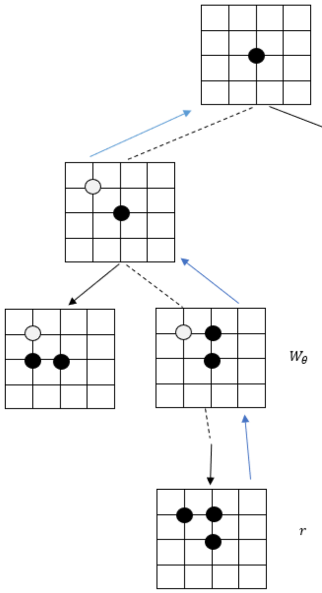


In stage 3 (**simulation**), the simulations from the newly selected node are run, where each simulation represents a complete game.

The value of node Q is computed by both the rollout classifier and the value network W_θ.

Summary of AlphaGo’s architecture (continued)

- ***Phase 2 – Playing:*** MCTS Stage 4



In stage 4 (**backpropagation**), the result of the simulation is then backpropagated i.e. returned to all the nodes in the path.

Concluding thoughts

- Go was considered the holy-grail of game-playing problems that we did not expect to be solved so quickly. The fact that AlphaGo defeated Lee Sedol is a truly remarkable achievement.
- If we look at the bigger picture, the win in and of itself is not the truly remarkable part. The approach used to win by the AlphaGo team is in my opinion much more important.
- By making extensive use of both supervised and reinforcement learning instead of using hand-crafted heuristics (like Deep Blue for example) bodes well for the future. By playing against previous versions of itself, AlphaGo became better and better at the game. The hope is that this approach can be generalized to solve other problems in AI.

Concluding thoughts

- However, a common criticism was that the supervised learning of AlphaGo's training involved training on a large dataset of games played by human players.
- One could argue that this had made AlphaGo biased to a certain degree towards imitating human play. Maybe if AlphaGo relied completely on self-play instead of having a supervised learning component, new strategies not popular among human Go players may have surfaced.
- If we go back to the 1990s, TD-Gammon, which relied only on self-play, explored strategies that were not usually pursued by human backgammon players of the time.

Concluding thoughts

- In late 2017, DeepMind tackled this criticism in the best way possible by introducing AlphaGo Zero [27], the next iteration of AlphaGo that relied solely on reinforcement learning.
- Rather than training on thousands of human games to learn to play Go, AlphaGo Zero skips this step and learns only through self-play.
- It rapidly surpassed superhuman level of play and defeated the previous champion Go program AlphaGo 100 games to 0 [27].

Concluding thoughts

- Interestingly, the new version is simpler: it combines the originally separate policy and value networks used in AlphaGo into one network allowing for more efficient training and evaluation. It also no longer uses rollouts (random simulations previously used to predict which player will win from the current board position).
- These key differences not only make AlphaGo Zero the new best computer Go program, it also makes it more general.
- However, there are still a few games that AI hasn't mastered: namely complex strategy games like StarCraft, where human performance is still superior.

Concluding thoughts

- AlphaGo's success has certainly paved the way for the AI community to tackle interesting new problems. The fact that most experts did not expect to see Go solvers reach this level of performance so quickly due to its intractable search space means that problems that we did not think of being solvable by AI may no longer be beyond us.
- While AlphaGo Zero is still in its infancy, researchers are optimistic that similar techniques may be applied to other structured problems like protein folding and have a positive impact on society.

Thank you.

Questions?

References

1. *Bouzy, Bruno, and Tristan Cazenave. "Computer Go: an AI oriented survey." Artificial Intelligence 132.1 (2001): 39-103*
2. *Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.*
3. *Byford, Sam. "Google's AlphaGo AI Defeats World Go Number One Ke Jie." The Verge, The Verge, 23 May 2017, www.theverge.com/2017/5/23/15679110/go-alphago-ke-jie-match-google-deepmind-ai-2017.*
4. *Müller, Martin. "Computer go." Artificial Intelligence 134.1-2 (2002): 145-179.*
5. *Hölldobler, Steffen, Sibylle Möhle, and Anna Tigunova. "Lessons Learned from AlphaGo."*
6. *Osborne, Martin J., and Ariel Rubinstein. A course in game theory. MIT press, 1994.*
7. *Schaeffer, Jonathan, et al. "Solving checkers." Proceedings of the 19th international joint conference on Artificial intelligence. Morgan Kaufmann Publishers Inc., 2005.*

References (continued)

8. *Maddison, Chris J., et al. "Move evaluation in go using deep convolutional neural networks." arXiv preprint arXiv:1412.6564(2014).*
9. *Van Den Herik, H. Jaap, Jos WHM Uiterwijk, and Jack Van Rijswijck. "Games solved: Now and in the future." Artificial Intelligence 134.1-2 (2002): 277-311.*
10. *Remus, Horst. "Simulation of a Learning Machine for Playing GO." IFIP Congress. 1962.*
11. *Zobrist, Albert L. "A model of visual organization for the game of GO." Proceedings of the May 14-16, 1969, spring joint computer conference. ACM, 1969.*
12. *Zobrist, Albert Lindsey. "Feature extraction and representation for pattern recognition and the game of go." (1970).*
13. *Reitman, Walter, and Bruce Wilcox. "The structure and performance of the Interim. 2 Go program." Proceedings of the 6th international joint conference on Artificial intelligence-Volume 2. Morgan Kaufmann Publishers Inc., 1979.*

References (continued)

14. *Boon, Mark. "A pattern matcher for Goliath." Computer go 13 (1989): 12-23.Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.*
15. *Brügmann, Bernd. Monte carlo go. Vol. 44. Syracuse, NY: Technical report, Physics Department, Syracuse University, 1993.*
16. *Coulom, Rémi. "Efficient selectivity and backup operators in Monte-Carlo tree search." International conference on computers and games. Springer, Berlin, Heidelberg, 2006.*
17. *Sutskever, Ilya, and Vinod Nair. "Mimicking go experts with convolutional neural networks." International Conference on Artificial Neural Networks. Springer, Berlin, Heidelberg, 2008.*
18. *Clark, Christopher, and Amos Storkey. "Training deep convolutional neural networks to play go." International Conference on Machine Learning. 2015.*
19. *Kocsis, Levente, and Csaba Szepesvári. "Bandit based monte-carlo planning." European conference on machine learning. Springer, Berlin, Heidelberg, 2006.*

References

20. *Chaslot, Guillaume, et al. "Monte-Carlo Tree Search: A New Framework for Game AI." AIIDE. 2008.*
21. *Burger, Christopher. "Google DeepMind's AlphaGo: How It Works." On Personalization and Data, 6 Feb. 2017, www.tastehit.com/blog/google-deepmind-alphago-how-it-works/*
22. *Tesauro, Gerald. "Td-gammon: A self-teaching backgammon program." Applications of Neural Networks. Springer, Boston, MA, 1995. 267-285.*
23. *W Schubert. KGS Go server, 2010.*
24. *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol.1. No. 1. Cambridge: MIT press, 1998.*
25. *Littman, Michael L. "Markov games as a framework for multi-agent reinforcement learning." Machine Learning Proceedings 1994. 1994. 157-163.*
26. *Campbell, Murray, A. Joseph Hoane Jr, and Feng-hsiung Hsu. "Deep blue." Artificial intelligence 134.1-2 (2002): 57-83.*

Extra - Guiding the Simulation Stage

- Rollouts or random simulations are used to predict which player will win from the current board position.
- Each simulation here represents a complete game. The quality of rollouts or simulations can significantly bolster MCTS [2]. Using a linear classifier that is trained on KGS matches, AlphaGo guides the simulations as follows:
  - A small area surrounding each legal move is selected and compared to precomputed patterns.
  - The input to the linear classifier is the array of indicators that indicate whether the selected area matches any precomputed patters.
  - The output is the probability of that move being selected.

Extra –CNNs

- In machine learning, a **convolutional neural network** (**CNN**, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery.
- **Feed-forward NNs are Artificial neural networks** (**ANNs**) or **connectionist systems** that are computing systems vaguely inspired by the biological neural networks that constitute animal brains.

Extra – Motivation behind CNNs

- Let's step back for a moment and try to understand the motivation behind this approach. As we saw in previous section, the mean value of simulations in MCTS only converges to the true value if it is provided an infinite number of simulations. This is certainly not possible in any domain, especially not in board games like Go where thinking time is limited [5, 19]. Thus, with a limited number of simulations, the quality of each simulation becomes much more important.
- The higher the quality of each simulation, the faster the convergence towards the optimal sequence of moves [5].
- Unlike MCTS, the best human players select a move and try to predict the outcome of the game after playing that move. There is a degree of intuition that the best human players make use of to guide their decision making [18].

Extra - How AlphaGo performed without search

- AlphaGo’s performance was tested based purely on policy networks against Pachi, the strongest open-source Go program. It is important to note that Pachi relies extensively on search Specifically, Pachi is a Go program which relies on 100,000 simulations of MCTS at each turn.
- At each move, the AlphaGo team selected the actions predicted by the policy networks that gave the maximum likelihood of a win. AlphaGo (based on only its policy networks) won 85% of the games it played against Pachi [2, 21].
- This was a truly remarkable result since Pachi relies extensively on search and it got outperformed by AlphaGo’s policy networks which relied on Convolutional Neural Networks. This highlights an important property of game play in Go: intuition is very important relative to long reflections [21].

Extra - Advantages of using MCTS

1. Using MCTS significantly reduces the number of explored nodes in the game tree. By only allowing promising branches, the tree grows asymmetrically with a low branching factor allowing only for promising branches [2, 5].
2. MCTS is well suited to parallelization since the simulations are independent. This property was leveraged by AlphaGo.

Extra - Disadvantages of using MCTS

1. MCTS is still not strong enough on its own to overcome the computational challenge of Go. Since the thinking time for the program is limited, using MCTS alone would not allow the program to gather enough statistics to precisely approximate the optimal values for the nodes [5].

2. Random rollouts used in the Simulation phase of MCTS may be unrealistic and hence provide a poor approximation of the target function [5].

3. While the Monte Carlo method has the elegant property that the mean value of all simulations converges to the true value, the only way to guarantee this property is to have an infinite number of simulations [19].

Extra - History and Evolution of Computer Go

The first scientific paper about Computer Go was published in 1963, where it considered the possibility of applying machine learning in Go [1, 10]. The first recorded instance of a Go program to defeat a human player, albeit, an absolute beginner was developed by Zobrist [11, 12]. It was primarily based on the computation of a potential function that approximated the influence of stones [1]. Since the beginning of this field of study, researchers have been working on sub-problems of the game, namely working with smaller boards or localized problems like life and death of groups [1]. Computer Go became an active research field in the eighties, when the first journal devoted to Computer Go was issued, in addition to early releases of several commercial programs [1].

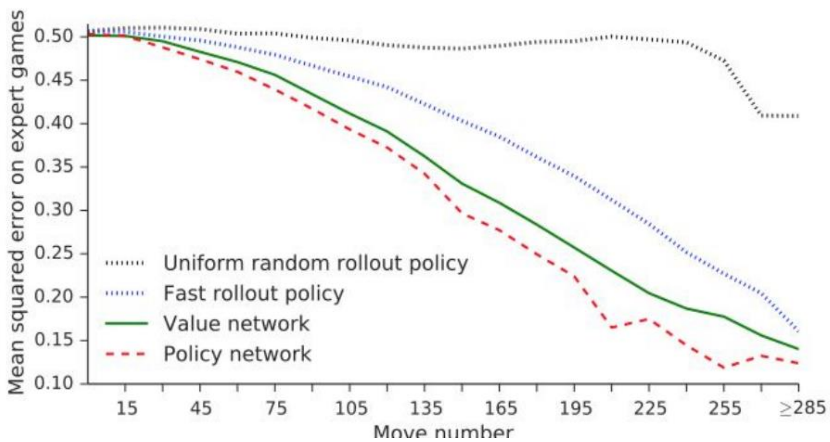
The first Go program to play at a level above a human beginner was designed by Reitman and Wilcox [13]. Wilcox's approach used abstract representations of the game board and reasoned about groups [1, 13]. The next notable breakthrough was by Boon in 1990 to use patterns to recognize common situations and to suggest moves [14].

The first instance of a Go program leveraging the Monte Carlo method was Gobble in 1993 [15]. By using the average of numerous simulations, Gobble assigned approximate values to possible moves; then it considered only the moves with the highest values. However, Gobble fared quite poorly relative to human Go players – it achieved a rank of only 25 kyu which translates to a very weak beginner [2, 15].

Crazy Stone [16] was the first Go program to incorporate Monte Carlo Tree Search (MCTS). It managed to outperform most of its contemporary competition which made use of conventional techniques. Since then, MCTS has been a staple in the computer Go community, eventually forming a critical part of AlphaGo's architecture [2].

Convolutional Neural Networks (CNNs) have been successfully used in Computer Go several times before AlphaGo [8, 17, 18]. DeepMind's AlphaGo successfully combined MTCS with CNNs to develop the first program to achieve superhuman performance [2].

Appendix – Performance comparison graph



We can see that the policy networks (RL+SL) performed best, closely followed by the Value Network. The fast rollout policy (simulations) performed at a level below the other two.