Jessica Malloy
HW4 Topic report

The problem

Go is an abstract strategy board game thought to be the oldest continuously played board game in history. The rules for Go are very simple, but the strategy involved is notoriously complex. In the game, each player takes a turn by adding a stone to any open space on the board. Once they are placed, the pieces do not move; however, they can be taken off the board if they are "captured", meaning that they are surrounded by enemy pieces on all orthogonal points. The object of the game is to capture as much of the board as possible. This lack of restriction and the large board size allow a variety of strategies, and decisions made early in the game can become very influential later on. Effective strategy in Go involves being able to read the board and determine which areas are safe and which areas will inevitably be captured. This is nontrivial, and requires looking many moves ahead.

Traditional AI programs designed to play games tend to have a few elements in common, including: hard-coded knowledge of the rules, a representation of the current board state and potential future board states, a tree search for generating legal moves, and an evaluation function to determine the utility of a given move. The tree search and evaluation components are especially difficult to develop in a software program for Go, due to the nature of the game.

First of all, the search space for Go is enormous. A 19x19 board has 361 possible locations, and for any given turn a player often has well over 100 possible moves—looking ahead just 4 moves can easily generate over 300 billion possible combinations. Given that early moves are often crucial hundreds of turns later, looking ahead can quickly become intractable.

Evaluation functions are also difficult to develop, since the effectiveness of any single move depends on the state of the rest of the board. Games can differ drastically depending on the players' play styles; in fact, there are estimated to be over 10^{170} possible legal board states. This variance means that any simple heuristic would be highly context-dependent and thus of limited usefulness.

For these reasons, creating a computer program that can play Go was long considered one of the biggest challenges in artificial intelligence. Where other game-playing programs rely on looking ahead and evaluating all possible moves using a hard-coded evaluation function, most people agreed that a successful Go program would require methods like probabilistic analysis, which are more advanced and closer to how humans may make in-game decisions.

How AlphaGo works

AlphaGo is a computer program that plays Go. In 2015, it became the first computer program to beat a professional human player, which was considered a major breakthrough in the field of artificial intelligence. Prior to this, the general consensus was that a strong computer Go player was at least 5-10 years away.

AlphaGo uses a combination of Monte Carlo tree search (MCTS) and convolutional neural networks (CNNs), combined with extensive training on both human games and self-play. General explanations of convolutional neural networks and of Monte Carlo tree search are provided, in order to provide a more thorough understanding of how AlphaGo uses both techniques.

Convolutional neural networks

Convolutional neural networks are a type of deep neural network that are typically associated with the field of computer vision, and often used to analyze image data. Their architecture is similar to the organization of neurons in the human visual cortex, in that each "neuron" in the architecture looks at an overlapping subset of an image. They work by creating localized representations of data and building off of these to create increasingly abstract features. This is easy to understand by looking at features found by a convolutional neural network trained in a face recognition task (Figure 1). Features at lower layers tend to be edges and curves at different angles, which are then combined to build more abstract features like eyes, noses, and eventually faces. Presumably within the context of Go, this hierarchical structure could be used to understand the likely outcome of a local fight, while also being able to place it within the context of the whole board.

In total, AlphaGo uses three neural networks, all of which take in the current board state as an input. These consist of two policy networks that evaluate all moves available, and a value network that computes the probability of winning, given the current board state.

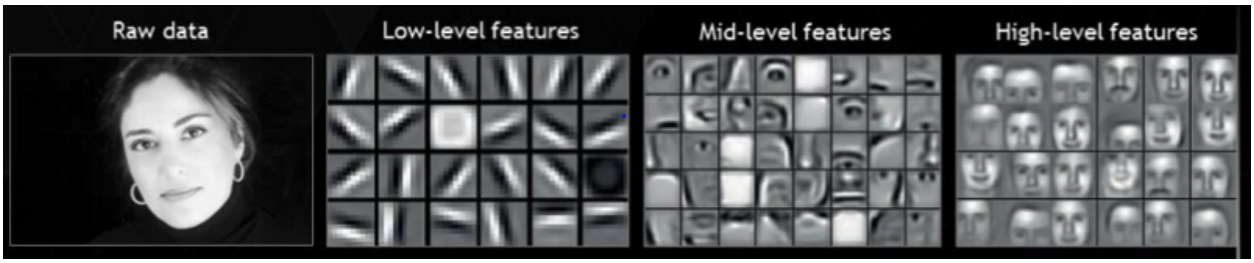

Figure 1. Increasingly abstract features from a CNN trained to recognize faces.

Monte Carlo Tree Search

Pure MCTS works by simulating several complete games, starting from the current state. The game state is modeled as a tree, where the current state is the root and its children are the states that result from all possible moves. The algorithm expands the tree by selecting successive moves until it reaches some leaf node *L*. If *L* is not a win or a loss, the algorithm then chooses a random child node *C*, expands it, and plays a random simulated game until it ends and an outcome is reached. The final result of this game is used to weight each node in the path from root, and nodes that lead to a win are more likely to be randomly chosen for future simulations. The equation used to adjust node weight should (ideally) balance the exploitation of paths with high average win rates and the exploration of moves with few simulations, and determines the overall shape of the expanded tree. After all playouts are completed, the move

with the most simulations is selected and becomes the new root, while the previous root and its other children are discarded.

Although MCTS converges slowly compared to other search functions, it offers significant advantages that happen to be especially well-suited to Go. Specifically, it does not require an explicit evaluation function, which is what makes faster search algorithms like alpha-beta pruning intractable. MCTS also tends to perform well in games with a high branching factor (for instance, Go) because it searches the tree asymmetrically and concentrates on promising subtrees. AlphaGo uses MCTS to perform a lookahead search through possible moves, and adjusts node weight using a combination of policy networks.

Supervised learning

In the first stage of training, AlphaGo learns to predict expert moves in human games of Go using a 13-layer convolutional neural network trained on 30 million positions, where the board state was fed as a 19x19 image input. The policy network $p_\sigma(a \mid s)$ takes as input randomly-sampled state-action pairs, where s is the board state and a is a move selected based on s, and outputs the probability distribution for all legal moves a. Supervised learning was used in the initial stage of training because it leads to faster, more efficient learning, and provides the network with immediate feedback. The researchers reported that the network alone attained 57.0% accuracy, and noted that small improvements in accuracy led to significant improvements in gameplay.

A second policy network was also developed with supervised learning. This roll-out policy $p_\pi(a \mid s)$ is a linear softmax policy using local pattern-based features. It only achieved an accuracy of 24.2%, but selected actions significantly faster than the larger policy network.

Reinforcement learning

In the second stage of training, AlphaGo improves on the previous policy network p_σ through reinforcement learning. Although supervised learning is quick and efficient, it is limited to data from human games, and can only learn to predict the move a human is most likely to make. In contrast, the goal of reinforcement learning is to predict the move that will win, which is the true goal of the policy network component.

This policy network p_ρ is also a convolutional neural network, identical in structure and initialized to the previous policy network p_σ. It also outputs a similar probability distribution $p_\rho(a \mid s)$ over all legal moves a selected from a board state s. The network was trained by playing games against randomly-selected previous iterations of the policy network, which prevents overfitting and stabilizes training. When a game outcome is reached, the weight change $\Delta\rho$ for each time step t in the game is computed using either a positive value for winning, or a negative value for losing. The weights are updated $\Delta\rho$ in the direction that maximizes the expected outcome. After training, the reinforcement learning policy network won in 80% of games played against the supervised learning policy network.

Value network

In the final stage of training, AlphaGo develops a value network $v_\theta(s)$ that predicts the probability of winning for a specific board state. It has a similar structure to the supervised and

reinforcement learning policy networks, except that it outputs a single prediction $v_\theta(s)$, rather than a probability distribution. The weights are trained by regressing on state-outcome pairs (s,z), with the network trying to minimize the error between the predicted outcome $v_\theta(s)$ and the actual outcome z. The network was trained on game data generated from the reinforcement learning policy network playing itself. Each data point originated from a separate game, because successive positions in the same game are highly correlated and lead to overfitting. The value network's evaluation was determined to be an effective estimate of the reinforcement learning policy by comparing its performance to a look-ahead search using the reinforcement learning network. It was only slightly less accurate, but required significantly less computation.

Move selection

AlphaGo combines the policy and value networks developed during training with a Monte Carlo tree search (MCTS) in an innovative but fairly complicated manner. A single tree traversal simulation is illustrated in Figure 2.

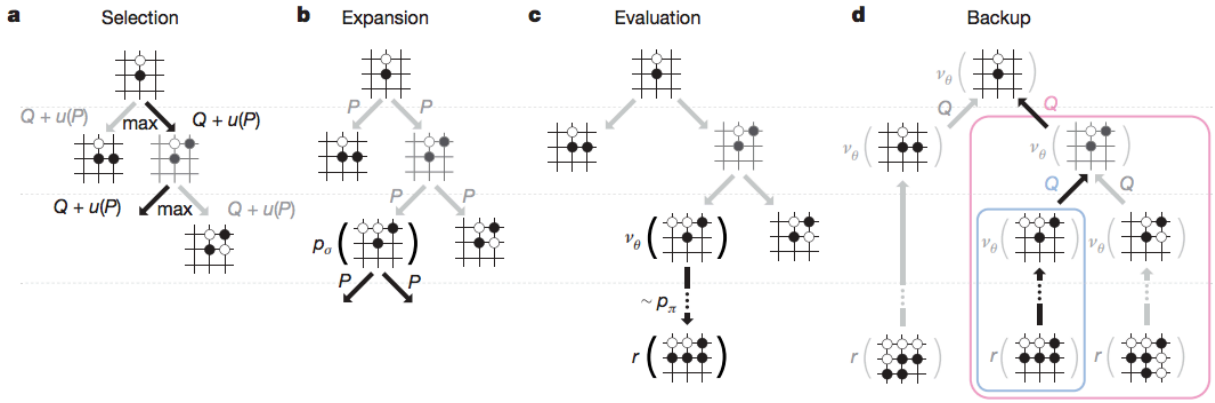

Figure 2. Monte Carlo tree search in AlphaGo. Note: the function r is used to compute whether a given board state resulted in a win or a loss. Games of Go often end in one player resigning, and then a determination of who captured the most territory by examining the board state. The outcome is almost never known automatically, so function r computes it at the end of a rollout.

Move selection begins with simulated tree traversal. Edges in the tree correlate to moves in the game, and a given edge (s, a) is defined by the action a selected from state s. Each edge is associated with an action value $Q(s, a)$, a visit count $N(s, a)$, and a prior probability $P(s, a)$.

Each simulation begins by selecting the move with the highest sum total of its action value Q and a bonus value u. This bonus value is proportional to $P/(1+N)$, in order to encourage exploration of promising but infrequently-visited nodes. If the resulting leaf node is expanded (i.e., visited for the first time), the supervised learning policy network p_σ is used to compute $P(s, a)$ for that node. The leaf node is then evaluated both by using the value network v_θ, and by the outcome of a random playout game using the fast rollout policy p_π. At the end of the simulation, each edge that was traversed is updated to increment the visit count N and compute the mean

action value Q from all simulation visits. When all simulations complete, AlphaGo chooses the child with the highest visit count from the root node.

A summary of how all policy networks developed during training were incorporated into MCTS:

- **Supervised learning policy network p_σ:** computes prior probabilities of a node when it is first expanded/visited
- **Reinforcement learning policy network p_ρ:** source of game data used to train the value network v_θ
- **Fast rollout policy network p_π:** used to evaluate the action value of an individual leaf node, along with v_θ
- **Value network v_θ:** used to evaluate the action value of an individual leaf node, along with p_π

Effectiveness of AlphaGo

AlphaGo was a significant improvement over similar Go programs. When AlphaGo was tested against several of the leading Go programs, it was able to win all but one of its 495 games. Furthermore, it was able to beat the next-strongest program, Crazy Stone, 77% of the time even after Crazy Stone was provided with 4 handicap stones. This indicated that it was likely several ranks above Crazy Stone, which was estimated to be at the level of an advanced amateur (about 6-dan).

More importantly, AlphaGo is the first Go program to reach a professional level when it defeated Fan Hui, a 2-dan professional, in a 5-0 match on a full-size board with no handicaps. Six months later, AlphaGo played Lee Sedol—a 9-dan professional player (the highest rank possible), and one of the best Go players in the world—and won four games, losing one. Prior to this, most experts believed a program at the level of AlphaGo was at minimum five years away from development.

The main breakthroughs that enabled AlphaGo's success were the development of effective functions for position evaluation, and a new tree search algorithm that efficiently evaluates and selects among available moves. In particular, Go is known for its deep and complex strategy, which limits the usefulness of evaluation functions utilizing direct instruction. Using a combination of supervised and reinforcement learning in deep neural networks, AlphaGo was able to develop an evaluation function that effectively approximates the optimal value function for Go. We know this because the reinforcement policy predicts the move most likely to lead to a win, meaning it's an estimate of the optimal value function. The value network is trained to predict the outcome of games played using the reinforcement policy, so its output is a double estimate of the optimal value function.

Current performance and future directions

After AlphaGo defeated Lee Sedol and attained the rank of 9-dan professional, it was updated to address weaknesses in its tree search. A known weakness of Monte Carlo search methods is that they attempt to prune sequences that are deemed less significant, which can sometimes lead to unlikely but significant line of play being overlooked in simulations. This

tends to occur when the success of a given sequence is highly dependent on the order moves are executed, which makes the sequence difficult to discover at random. Consequently, the algorithm is left with a relatively unexplored section of the search tree and insufficient knowledge of effective moves. Most experts agree that this is why AlphaGo lost to Lee Sedol in their fourth game.

AlphaGo became significantly stronger after being updated, and remained undefeated in all its matches against subsequent human competitors. Notably, it defeated Ke Jie, a 9-dan professional considered to be the best human Go player in the world, in 3 out of 3 matches at the Future of Go Summit held last year. Afterwards, AlphaGo was retired by DeepMind.

Later that year, developers at DeepMind developed an even stronger version of AlphaGo called AlphaGo Zero, which is based solely on reinforcement learning and perceives only the position of stones on the board. AlphaGo Zero attained superhuman performance in just 3 days of training; in contrast, earlier versions of AlphaGo required months of training to get to a similar level. AlphaGo Zero was able to surpass the strongest iteration of AlphaGo in just 40 days of training. AlphaGo Zero is also notable because the independence from human data means that the algorithm underlying it can be generalized to solve similar problems. A related program, AlphaZero, already does this, to an extent—it has mastered chess and shogi in addition to Go.

Go was long considered the ultimate challenge for game-playing in artificial intelligence; some experts predict that the superhuman abilities of AlphaGo and its successors will be the last major advancement in AI programs designed to play games. Regardless of whether that's true, the algorithms used in AlphaGo and AlphaGo Zero have potential applications in any problem that involves a large search space and unclear evaluation function. DeepMind is reportedly already working on generalizing AlphaGo Zero's approach to protein folding, and foresee similar applications in drug discovery and simulation of chemical reactions.

AlphaGo has also created new possibilities for Go strategy in human games. AlphaGo has been known to make unconventional moves that have later proved to be extremely influential. After beating several Go champions, players at all levels have started studying AlphaGo's strategy in order to gain insight into the game. Several professional players have stated that AlphaGo discovering novel strategies has ushered in a new wave of creativity and innovation in Go, and encouraging them to reconsider moves that were thought to be unplayable by traditional thinking.

Personal opinions

Initially, I chose to write about AlphaGo because I was interested in learning how to play Go. A friend of mine started learning how to play after working at AI2, so I already knew a little bit about the game and its connection to artificial intelligence from conversations with her. We were already planning on playing a few games together over spring break, and writing about a new game I learned how to play seemed like it would be relatively fun, for a homework assignment.

This isn't directly related to AlphaGo, but the most interesting thing I learned when researching this report was that, according to AlphaGo's evaluations, Ke Jie was playing

perfectly for the first hour of his second game against AlphaGo. He was 19 years old at the time, and ranked the best player in the world.

More generally, I thought the descriptions of AlphaGo's play style were very interesting. It plays very conservatively, especially compared to human players, and tends to prioritize a greater probability of winning over maximizing territorial gains. As a result, a lot of its moves, especially at the beginning, have been characterized as odd or inhuman. I also really liked that its unconventional playing style is encouraging human players to develop and innovate new strategies.

References

Good, Irving John. "The Mystery of Go." *New Scientist* 427 (1965): 172-174. url: http://www.chilton-computing.org.uk/acl/literature/reports/p019.htm.

"Overview of Computer Go." *IntelligentGo.org*, url: web.archive.org/web/20080531072850/http://www.intelligentgo.org/en/computer-go/overview.html.

Tromp, John, and Gunnar Farnebäck. "Combinatorics of go." *International Conference on Computers and Games*. Springer, Berlin, Heidelberg, 2006.

Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." N*ature* 529.7587 (2016): 484-489.

Silver, David, et al. "Mastering the game of go without human knowledge." *Nature* 550.7676 (2017): 354.

Maini, V. "Machine Learning for Humans, Part 4: Neural Networks & Deep Learning." *Medium* (2018). url: https://medium.com/machine-learning-for-humans/neural-networks-deep-learning-cdad8aeae49b.

Baker, Lucas and Fan, Hui. "Innovations Of Alphago." *DeepMind Blog* (2018).

"Going Places." *The Economist* (2017). url:https://www.economist.com/news/science-and-technology/21730391-learning-play-go-only-start-latest-ai-can-work-things-out-without.