AlphaGo Report

AlphaGo, created by the DeepMind team at Google is one of the most advanced AI systems in the game playing field. The game has a magnitude of more possible games than there are atoms in the universe, yet when put up against the best player in the world, with time constraints, it won. Since the breadth and depth of an Go game are exponentially greater than almost all other board games (breadth of 250, depth of 150), it is important to introduce algorithms other than the standard depth-first or breadth-first algorithm. This is why AlphaGo uses a mixture of Monte Carlo tree searches and deep reinforcement networks.

Instead of using a scoring heuristic like minimax, trees are simulated over and over, storing values like which nodes were used in won games. Each subsequent tree that is simulated is affected by the results of the previous tree, so it is less random, and reduces the branching factor more with every simulation. This way, instead of trying to predict how an opponent might move, like minimax, a policy network helps predict which moves will likely lead to a win. The important part here, is that with every simulation, the branching factor of "likely" moves is reduced significantly. Not only does this make the move prediction much faster, but it also led to a huge increase in win rate against skilled agents. For instance, when the AlphaGo agent played against the other best Go agent Pachi, using no searches, only its policy network, it won 85% of games. This is incredibly effective, seeing as a convolutional neural network only won 11% of its games against Pachi.

While AlphaGo doesn't use a typical heuristic as minimax would, the agent still makes use of something called a value network. This is what makes AlphaGo's Monte Carlo tree search so effective. The value network, similar to a heuristic, evaluates a game state and gives a score or probability of a win given the state. In this case though, no heuristic function is provided. Instead, the network learns how to evaluate each state based on millions of states and their end-game results. From here it will guess what moves it should make based off of how simulations faired making the same move.

The main problem with the value network, was that it only utilized reinforcement learning. In a game with so many viable move choices, this easily leads to overfitting of the data. To combat this problem, self playing was implemented, using random opponents. The result of this means that the networks were adjusted to account for the best possible moves in any situation, not just one single optimal move. This also caused the evaluation of any board state to require 15,000 times less computational power than its previous Monte Carlo reinforcement learning policy. Less computations, but the same accuracy is what ultimately led to the AlphaGo to win a staggering 99.8% of all games against other Go programs and even over 75% against them when handicapped by four moves.