Summary of How AlphaGo Beat The World Champion

AlphaGo wasn't the first algorithm to approach mastering the game of Go, however it proved to do exponentially better than current programs that are out there like Pachi and Fuego. The game Go is similar to that of chess and checkers, in which you meet the following criteria: First, there are two players, one of which who will be dubbed a winner, and the other will be dubbed the loser (unless there is a tie). Second, there is no hidden information on the board (for example, when you play poker, you don't know what cards other players have in hand, or what cards will be drawn from the deck). Each move made by a player is information that is accessible to both parties. Third, there will eventually reach a terminal state - meaning the board will reach a point in which there are no more moves to make and in which the winner can be determined.

If there is a terminal state, then Go could theoretically be solved looking at the current state of the game, and expanding out all the possible moves, opponent moves, etc. until the end of the game is reached and we can determine who is the winner. Going through all these possible states, however, would require much too much time and computational power. You can think of the possible sequences of moves being represented by b^d , where b is equal to the breadth (number of legal moves in a game) and d is the depth (how many moves in the length of the game). For Go, this would be roughly 250^{150} which is much too large. Below is how we manage to get around this, using tactics to cut the breadth and depth required to win a game.

Supervised Learning (SL) of Policy Networks

The first step to approaching this challenge is to use supervised learning to train AlphaGo to predict what moves would be made based on a large database of moves made by human professionals. The outcome of this is a policy network, which creates a probability map, mapping out the probability of a legal move being made. The performance of the SL policy network is based on its accuracy in determining the probability of a move being made.

Reinforcement Learning (RL) of Policy Networks

The second step is to improve the policy networks we have from supervised learning. The RL policy network is identical to the SL policy network, however now games are played between the current policy network and a randomly selected previous iteration of the policy network. With RL, once a terminal state is reached, we can reward the winner with a positive value (i.e. +1) and punish the loser with a negative value (i.e. -1). Through reinforcement learning, AlphaGo will always look to maximize its value. The weights assigned to each move that affect where to move to next are then updated depending on the value in terminal state.

Reinforcement Learning (RL) of Value Networks

The third step focuses on evaluating the value associated with a future position. As it is too challenging to determine all the values associated with every move (resulting in the aforementioned 250^{150} moves), we can instead focus on truncating the depth we need to search with an approximate value function that predicts the outcome of the game.

AlphaGo then combines these policy and value networks with a Montey Carlo Tree Search (MCTS), which cuts down the breadth of the game by randomly sampling moves to make based on the probabilities associated with the policy network. It won't branch out, but will instead sample long sequences of actions to get a stronger idea of the future state of the game.