A Review of the Alpha Go Algorithm

Sean Batir

August 10, 2017

1 Objective

The ancient game of Go provides an ideal toy example, or contextual background, for artificial intelligence researchers to develop the appropriate suite of techniques that an "intelligent game-playing agent," must possess. What made Go seem like a previously intractable problem in AI was the large search space required to exhaustively iterate through all possible combinations of outcomes in the game, which was estimated to be a total of roughly 13 million possibilities. If one thinks about the possible number of states of a Go game as the breadth, or width of a search tree, and the number of turns in a game of Go as the depth, or length of the search tree, then the relationship may be modeled by b^d. The majority of humans do not exhaustively compute and iterate over a search space of roughly thirteen million moves. Instead, we use a variety of heuristics to arrive at our final answer. Along a similar vein, the developers of the AlphaGo algorithm had to determine how to more "intelligently" search through such as massive search tree on the order of b^d

2 Techniques and System Architecture

The AlphaGo algorithm simply combines two existing and relatively well-characterized tools in the AI and machine learning community: deep neural networks and a tree search algorithm. For simplicity, we will divide our discussion of the system architecture between the two types of neural networks recruited by AlphaGo, and the Monte Carlo Search Tree Algorithm that is used to determine a winning strategy.

2.1 Value and Policy Network

The neural networks are essential to reducing the search space within every game of Go. While one neural network will determine an optimal **policy** for how to move units around a game of Go, the other network will determine an optimal, scalar **value** associated with maximizing the chance of winning. Optimizing the policy reduces the breadth, b, of the tree, while optimizing the value reduces the depth, d, of the tree. But how does this even happen? The creators of AlphaGo utilize two classical paradigms in machine learning to develop neural networks that possess a "winning policy." These paradigms are supervised learning and reinforcement learning. Initially, the developers provided the convolutional neural network with data from human expert players of Go, in the form of Go board images over time. By feeding the data through a convolutional neural network, the system extracted the latent space, or relevant features of board configurations belonging to expert, human Go players. After determining the features associated with a winning strategy, a reinforcement learning approach was used, which basically means that the game-agent played against itself many times in order to maximize its likelihood of winning, thereby rewarding (and thus reinforcing) game strategies that result in a win for the AlphaGo agent. Formally, the purpose of the agent playing against itself is to "gain an improved stochastic gradient ascent policy," which overall improves the policy for winning criteria.

The policy network enables the AlphaGo agent to sample possible actions given a specific game state, thereby storing the probability distribution of possible moves given a specific position. Therefore, this enables the breadth of the tree to be reduced significantly, removing board configurations that are highly improbable.

Likewise, the value network enables the agent to perform position evaluation and assign scalar values that describe board configurations or strategies, that are more likely to result in the

game agent winning a game of Go. The natural result of implementing position evaluation is a natural reduction in the depth of search tree, ultimately achieving the final objective of reducing a previously intractable search space. In laymen's terms, the algorithm is therefore capable of winning the game with a shallower tree depth and less overall turns or branching moves.

2.2 Monte Carlo Tree Search Algorithm

The Monte Carlo tree search algorithm is then used to actually traverse the "tree" of possible moves available to an AlphaGo player against any opponent. The Monte Carlo algorithm relies on lookahead search, and is only able to traverse the tree at all based upon the scalar values and probability distribution of board configurations generated by the value and policy neural networks described above.

3 Results

Alpha Go achieved a 99.8 percent winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program was capable of defeating a human professional player, a feat that was previously thought to be at least a decade away.