Review of the AlphaGo paper

Atanas Abrashev

March 11, 2017

The game of Go has been one of the most challenging games for Artificial Intelligence, owing to its enormous search space and the difficulty of evaluating the board. The paper introduced an approach that uses neural networks to evaluate board positions and select moves, trained by a combination of supervised and reinforcement learning. Combined with a state-of-the-art Monte Carlo search algorithm, the authors' program managed achieve a 99.8% winning rate against other Go programs and also defeated the European Go champion.

We learn that techniques such as alpha-beta pruning in the minimax algorithm and position evaluation can only take us so far - they require encoding domain knowledge about the game that is being played and sometimes the branching factor makes using those techniques intractable. Instead, the authors' approach requires no domain knowledge and tackles the big branching factor by playing out only the most promising moves, according to a policy trained using neural networks.

The neural network that picks moves, starts as a supervised-learning policy network, and is trained on a database of moves made by strong human players. It does not care about winning games, but rather picking the most-likely move a human would make. This network creates a probability distribution over all legal moves and the one with the highest probability is considered the best move. Its rate of success is 57%. The network is then improved using reinforcement-learning, using millions of simulated games, by having AlphaGo play itself. This puts a bias on winning games, rather than simply picking the best move according to expert human players. According to the authors, an agent that selects its moves best on such a policy network performs as strongly as recent state-of-the-art without doing any search at all.

To be able to implement searching, the authors use a weaker version of the move-picking network, which is too slow to be used in Monte-Carlo rollouts. Instead, they create a network which considers only a subset of the board state. This version, while not as strong as the first one, is a lot faster and enables move-selection during simulated playouts.

Next, the AlphaGo team created an evaluation function, which samples what they call a value-network. This network estimates the probability of winning the game, given the current board state. This is used to provide some estimation of how good positions are and gives some feedback to the Monte Carlo search about how deep or shallow it should attempt to search a given move. This position evaluator is trained on random samples of millions of simulated games between two instances of AlphaGo playing using it's stronger move-picking policy network.

Finally, the paper shows an implementation of the complete MCTS algorithm. It uses the weaker move-picking policy network to choose the next set of possible moves, because it's quicker and allows for a broader search space. Next, for each possible move, the value-network is used to give an estimation of how good this position is for winning. However, in addition to that, the algorithm considers simulated rollouts using the same weaker move-picking policy. These two approaches for move estimation give two independent guesses and are combined to create an evaluation function, which gives equal weight to both guesses to determine the final strength of each move.

The algorithm above was executed on a cluster of machines in a distributed way and the final product, as we now know, managed to beat the world champion, Lee Sedol. While all the techniques have been previously explored, it is the novel approach of combining them that managed to produce a general algorithm that solves a problem previously thought to be intractable. The fact that it manages to do so without encoding any domain knowledge also gives hope about the potential future applications in the field of Artificial Intelligence.