Mastering the game of Go with deep neural networks and tree search – Recap

Games like Chess, Othello or Go provide perfect information to determine the outcome of a game, under perfect play by all players. Unfortunately the search tree of Go has a branching factor about 250 and a depth of 150. Furthermore one move should be made within 5s and it´s difficult to evaluate board positions and moves. Till 2016, Go programs were only able to compete at Amateur level. The goal is to shift this border to a professional level, by developing methods that might be useful in other domains too.

In Chess, opening books an alpha-beta-pruning can be used to reduce the breadth of the search tree, and the depth can be reduced by position evaluation, which can be determined by handcrafted features.

This approach does´nt work well for Go. So in general Go programs use Monte Carlo rollouts to reduce the breadth of the search tree, which search to maximum depth without branching at all. To reduce the depth, AlphaGo uses a truncated MCTS that terminates rollouts before the end of the game and uses a value function in place of the terminal reward. So, the value of each state in the search tree is estimated but the values and the policies to select actions become more accurate as more simulations are executed. It is common to use MCTS enhanced by policies that are trained to predict human expert moves. While other Go programs are limited by value functions based on a linear combination of input features, AlphaGo uses a policy and value network for look ahead search. The policy network to predict the probability of a human move, the value network to predict the outcome of a game at the state and a fast policy network to rollout. The search tree can be reused at subsequent time steps. Compared to Deep Blue, AlphaGo evaluates about thousands of times fewer positions.

The policy and the value network are using similar architectures. Both have a 19 x 19 x 48 input layer and 13 hidden layers, but differ in the last layers. While the policy network outputs a probability distribution, the value network only indicates win or loss. The stacked 48 features in the input layer are directly derived from the raw representation of the game rules. For example, stone color, liberties, captures, legality, and so on. The hidden layers show a typical CNN with a 5x5 filter used to capture local statistics and repeated layers of 3x3 convolutions with ReLU activation.

First, a supervised learning policy network is trained to predict moves. 30 million positions from 160,000 games, played by experts, were used as training data. After training on 50 GPUs for 3 weeks, the network was able to predict expert moves with 57% accuracy. As the goal is to win games rather than to predict moves, the policy network gets further adjustments by reinforcement learning. The final outcomes are optimized by self-play for one day. A second, faster policy network with only 27% accuracy was trained that makes it possible to run rollouts to the end of the game.

The purpose of the value network is to estimate the probability of the current move leading to a win. This is useful in choosing what moves to make. The training by states from a database does´nt generalize well, since the states of a game are very similar and all share a common outcome. To mitigate this problem, a new dataset with 30 million distinct positions from separate games was created. The reinforcement learning took 1 week.

The final version used 40 search threads, 48 CPUs and 8 GPUs. Playing against other bots, AlphaGo won 494 out of 495 games. A distributed version was used to play against Fan Hui, a multitimes European Go champion. The main search tree was calculated by a master CPU, 176 GPUs for policy & value network and 1,202 CPUs for rollout policy networks. AlphaGo won 5 – 0, a result that was believed to be at least a decade away.