Analysis of “Mastering the game of Go with deep neural networks and tree search” by DeepMind

Go AI Background:

Previous state of the art Go programs are based on Monte Carlo Tree Search (MCTS) rollouts with policies (linear combinations of input features) to predict human expert moves. The policies are used to reduce the search space since search space for Go is so large (b^d, b~=250,d~=150). Brute force search to play Go is infeasible for any computer. MCTS rollouts work by randomly sampling the search space and searching to end game without branching by sampling long sequences of actions for both players from the aforementioned policy. The average of the rollouts is then used to evaluate the best position. This approach can be used to achieve strong amateur play.

AlphaGo Improvements:

AlphaGo uses neural networks to improve its (three) policies and a value network.

AlphaGo applies a Supervised Learning (SL) neural network to train a policy to predict human expert moves. The SL network alternates weighted convolutional layers with rectifier nonlinearities and a final soft-max layer. The network is trained from 30 million positions from the KGS Go Server. The trained network was able to predict expert moves with a 57% accuracy compared to a best of 44.4% from other research groups.

The SL policy network achieved much improved accuracy but was slow. Therefore a smaller/faster SL network was trained using a weighted linear softmax of small pattern features. This achieved an accuracy of 24.2% but only took 2μs to select an action compared to 3ms for the larger SL network.

The RL policy network aims to predict best moves from playing itself vs human expert play. Structurally the RL policy network is identical to the SL policy network and even initialized to the same weights but it is trained differently. After training, the RL policy network was able to win >80% of the games against the SL policy network and 85% of games with Pachi. Previous SL-only networks were able to win ~ 11% of games with Pachi (open source MCTS heuristic).

The last network is the value network. It’s function is to estimate the value given a particular game state. In order to do this the RL policy network architecture is tweaked to output a single prediction/value that estimates the probability that the move leads to a win. The network is trained through a state-outcome dataset of 30 million positions derived from separate self-play games. The value network single evaluation approached the accuracy of Monte Carlo rollouts using the RL policy network but used 15,000x less computation.

Putting this all together, AlphaGo starts the game simulation from the current player position and runs to endgame in 4 distinct phases.

Selection Phase:
Given current position the move with the highest value is selected
Expansion Phase:
“Leaf” positions may be expanded. The new position is processed once by the policy network and the output probabilities are incorporated into the value calculation.
Evaluation Phase:
At the end of a simulation, the endgame position values are the combination of the value network output and the MCST rollout with the fast SL policy output.
Backup Phase:
Values are propagated up the tree.

In AlphaGo the SL policy network performed better than the RL policy network (see expansion phase). According to the paper this was presumed due to the nature of how humans play Go. The optimal value network was the RL based value network described above.

Results:

In 2016 DeepMind’s AlphaGo accomplished this feat by beating the European Go Champion Fan Hui and (after this paper was published) 18-time world champion Lee Sedol. In addition, AlphaGo was shown to beat other top Go programs 99.8% of the time (out of 495 games). The most impressive thing about the AlphaGo accomplishment to me is the paper’s claim that during the Fan Hui match AlphaGo evaluated 1,000 less moves than Deep Blue did vs Kasparov. This suggests that AlphaGo represents a vast improvement in “intelligence” rather than simply an improvement just based on its ability to more quickly brute force search a game tree.