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

Go was believed to be intractable due to its complexity for two main reasons: it's a large game(b=250, d=150) and weak/limited value functions(i.e. how to evaluate the board at state). Before AlphaGo, the prevailing state-of-the-art technique was to reduce depth and breath of the search space while using Monte Carlo Tree Search(MCTS). MCTS picks a node and plays random moves to the end to simulate game play then backpropagates to estimate the value at each state; eventually the policy converges to optimal play and the evaluations converge to optimal value function. However, the resulting programs only reached amateur dan level.

With a combination of deep neural networks and tree search(MCTS), the authors were able to develop a program that plays at a level 'of the strongest human players'. The program employs convolution neural networks: a value network to evaluate positions and a policy network to sample actions. First, the pipeline begins with a supervised learning(SL) policy network trained on 30 million amateur matches. The SL network consist of 13-layers, alternating between convolutional layers and rectifier nonlinearities. A final softmax layer outputs a probability distribution over all legal moves a, resulting in a SL policy network p(a|s) with 57% accuracy of predicting next move. The network is trained using stochastic gradient ascent to maximize the likelihood of human moves. The second stage of the pipeline focuses on improving the policy network to predict the "best" move by using reinforcement learning(RL). The SL policy is copied to initialize the RL, then the RL policy essentially plays itself by randomly selecting previous iterations of the policy network. Using a reward function, weights are updated at each time step using stochastic gradient ascent that maximizes the expected outcome. When played against the strongest open-source Go program, Pachi, the RL policy network won 85% of the games, using no search. Since the RL network is slow(3ms) for MCTS, they trained a faster(3μs) but less accurate rollout policy. The final stage of the pipeline focuses on the evaluating the position of the board by estimating a value function that predicts the outcome. The value function is estimated using the RL policy network using regression to minimize the MSE between the predicted value and the actual outcome. The network initially had an error of 37% but after self-play the error decreased to ~23%. Lastly, the 4 neural networks are combined in an MCTS algorithm. The policy network discovers the current best moves, the tree is then transversed by simulation where the value network and outcome of the fast rollout policy network combine to estimate an evaluation for the node. Finally, at the end of simulation, the action values are updated and we get the optimum move.

AlphaGo is far superior to the previous generation programs. A single-machine AlphaGo 'is many dan ranks stronger than previous Go program, winning 494 out of 495 games (99.8%).' Further, without rollouts, AlphaGo exceeded the performance of other Go programs, suggesting 'that value networks provide a viable alternative to Monte Carlo evaluation in Go'. The AlphaGo team was able to develop effective move selection and position evaluation to overcome the complexity of Go