Research Review of AlphaGo

by Avery Heizler

Brief Summary:

Mastering the game of Go with deep neural networks and tree search, published in Nature, and delves into the novel approach that Google's DeepMind team used to create the computer program **AlphaGo** that uses **value networks** to evaluate positions on the game board and **policy networks** to select moves for the game Go. It also incorporates **Monte Carlo Tree Search** simulation with these value and policy networks to evaluate the value of each game position in the search tree to identify the most promising move. Altogether, there were four neural networks used in the creation of AlphaGo.

Stage 1: Supervised Learning

The first stage of training the AlphaGo program was to build upon prior work on predicting expert moves in the game of Go by using **supervised learning**. The team created a 13-layer deep convolutional neural network (CNN), which they called **SL policy network**, which alternated between convolusional layers with weights and rectifier linearities. A final soft-max The SL policy network trained on randomly sampled state-action pairs from 30 million positions from the KGS Go Server. The policy network took as input a simple representation of the board state and output the *probability distribution* over all the next legal moves. These moves were based upon what expert human players would have selected given the state of the game.

The network was able to predict expert moves on a held out test set with an accuracy of 57.0% using all of the input features, and an accuracy of 55.7% using only raw board positions and the move history as inputs. The small improvements in accuracy led to large improvements in playing strength. Larger networks achieved better accuracy but were slower to evaluate during the search. A faster but less accurate rollout policy, using a linear softmax of small pattern features yielded an accuracy of 24.2%, using just 2μs to select an action, rather than 3ms for the policy network.

Stage 2: Reinforcement Learning

The second stage of the training pipeline aimed at improving the policy network by policy gradient reinforcement learning. The **RL policy network** was identical in structure to the SL policy network. Games were played between the current policy network and a randomly selected previous iteration of the policy network. By randomly selecting previous iterations of the policy network as opponents, the team was able to stabalize the training and prevent overfitting to the current policy.

When the RL policy network played head-to-head against the SL policy network, the RL policy network won more than 80% of the games. The RL policy network also played against the strongest open-source Go program called *Pachi*, a sophisticated Monte Carlo search program that ranked at 2 at amateur *dan* on KGS. Using no search at all, the RL policy network won 85% of the games against *Pachi*.

Stage 3: Reinforcement Learning of Value Networks

The final stage of the training pipeline focused on posiition evaluation, estimating a value function that predicts the outcome from a position of games played by using the same policy for both networks. The team wanted to know the optimal value function under perfect play, so they estimated the value function for the strongest policy, using the RL policy network. This neural network had a similar architecture to the policy network, but the output was a single prediction instead of a probability distribution. The team trained the weights of the value network by regression on state-outcome pairs, using stochastic gradient descent to minimize the mean squared error between the predicted value and the corresponding outcome.

This naïve appoach of predicting game outcomes from data consisting of complete games leads to overfitting. When the policy network trained on the KGS data in this way, the value network *memorized* the game outcomes rather than *generalizing* to new positions. This achieved a minimum mean squared error of 0.37 on the test set, compared to 0.19 on the training set. In order to mitigate this problem, the team generated a new self-play data set consisting of 30 million distinct positions, each sampled from a separate game. The RL policy network played each game against itself until the games terminated. Training on this data set led to mean squared errors of 0.226 on the training set and 0.234 on the test set, indicating minimal overfitting.

Stage 4: Searching with Policy and Value Networks

The evaluation of policy and value networks required several orders of magnitude more computation than traditional search heuristics. In order to efficiently combine Monte Carlo tree search with deep neural networks, AlphaGo uses an asynchronus multi-threaded search that executes simulations on CPUs, and computes policy and value networks in parallel on GPUs. The final version of AlphaGo consisted of 40 search threads, 48 CPUs, and 8 GPUs.

The SL policy network performed better in AlphaGo than the stronger RL policy network. It is presumed this is due to the nature that humans select a diverse beam of promising moves, whereas the RL policy network optimizes for the single best move. However, the value function derived from the stronger RL policy network

performed better in AlphaGo than a value function derived from the SL policy network.

In order to evaluate AlphaGo, the team ran an internal tournament among variants of AlphaGo and several other Go programs including Crazy Stone, Zen, Pachi, and Fuego, all of which are based on high-performance Monte Carlo tree search algorithms. The results of the tournament suggest that single-machine AlphaGo is many *dan* rankss higher than any other Go programs to date, winning 494 out of 495 (99.8%) against other Go programs. AlphaGo also played games with four handicap stones against Crazy Stone, Zen, and Pachi, with a winning rate of 77%, 86%, and 99%, respectively. The distributed version of AlphaGo was stronger, winning 77% of games against single-machine AlphaGo and 100% of its games against other Go programs. Even without rollouts AlphaGo exceeded the performance of all other Go programs, demonstrating the value of using networks to provide a viable alternative to Monte Carlo evaluation in Go. However, the mixed evaluation ($\lambda = 0.5$) performed the best, wining $\geq 95\%$ of games against other variants. This suggests that the two position-evaluation mechanisms are complementary; that the value network approximates the outcome of games played by the strong but impractically slow RL policy network, while the rollouts can precisely score and evaluate the outcome of games played by the weaker but faster Fast Rollout policy network.