Research Review of AlphaGo Paper

The AlphaGo paper describes a successful effort to improve computer play of the Go board game to superhuman level by applying deep learning in concert with existing techniques for game playing.

The previous state of the game-playing art, which has faciliated superhuman level play in a variety of games like backgammon, chess, and scrabble, circumvent the exponential b^d possible move sequences by two strategies. First, position evaluation can yield the value of certain branches to reduce depth search, which helps with games like chess (b≈35, d≈80). Unfortunately, Go (b≈250, d≈150) is likely too complex for this kind of evaluation. Second, reduction of the breadth of the search is achieved through sampling of probabilistic policies $p(a|s)$, the probability distribution of all possible actions a in a state s. Such policies are refined using techniques like Monte Carlo Tree Search (MCTS) to build up values of various game states. The state of the art previous to AlphaGo utilizes MCTS enhanced with expert human moves to narrow the search for optimal moves.

The AlphaGo approach introduces deep convolutional neural networks to the mix of techniques for improving Go play. The 19x19 go board with three possible states for each position (black, white, or empty) very much resembles the kind of raster data (e.g. images) that deep neural networks excel at working with. Deep convolutional neural networks are employed for the two primary game playing roles: (1) policy networks for sampling actions at particular board states and (2) value networks for evaluating board state values.

Policy networks are constructed from convolution layers with ReLU activation and a softmax output layer that represents the probabities of all actions at a particular game board state. A 13-layer policy network was trained on 30 million positions from the KGS Go Server using gradient descent back-propagation. A trained policy network predicted expert moves 55.7% of the time using only raw board input positions and move history as inputs, compared to 44.4% of the previous state of the art. Per Figure 2a in the paper, increasing the testing accuracy from 50% to 58% increased the win rate from less than 10% to over 50%, a massive jump in playing strength. A faster by but less accurate rollout policy using linear softmax of small pattern features can select a game action in 2 µs instead of 3 ms.

The above supervised learning (SL) approach is augmented by a second training stage of reinforcement learning (RL) using the same network architecture and weights to stabilize training and prevent overfitting. Reinforcement learning is achieved by playing games between the current policy and randomly selected previous iterations of the policy network with end-game rewards of +1 for winning and -1 for losing. Stochastic gradient ascent updates weights at all time steps in the direction that maximizes the expected outcome.

The trained RL policy networks won more 80% of games against the SL policy networks and, without any search, won 85% of games against Pachi, an open-source Go playing program ranked at 2 amateur dan on KGS and executes 100,000 simulations per move.

The final training stage utilizes a similar convolutional neural network architecture for estimating board position values. These networks output a single value for each board position instead of the probability distribution output of the policy networks. Estimates of position values are achieved by playing out moves for both players using above RL policy networks. The value networks are trained by minimizing the mean squared error between the outcome z and the estimated value v for each input board position using stochastic gradient

descent.

To prevent overfitting from training on highly-correlated successive game positions, a data set of 30 million game positions all sampled from different games was used. After training, the value estimation of a board position inferred by the trained value network approaches MCTS accuracy using the RL policy network but with 15,000 times less computation.

AlphaGo uses an MCTS that make uses of the outputs of the policy and value networks described above. The action value *Q(s, a)*, the visit count *N(s, a)*, and the prior probability *P(s, a)* are stored for each search edge *(s, a)*. *P(s, a)* is just the probability output of the policy network for an action *a* at the state *s*. *Q(s, a)* is a function of the weighted sum between the board position value output of the value network and the value generated from a fast rollout to the end game from the state s.

From a root state (presumably the current game board state), a simulation is run without backup to the end game by selecting the action with the highest *Q(s, a)* plus a bonus that is a function of *P(s, a)* at each time step to maximize action value. However, this action value decays with repeated visits to encourage exploration. The action values *Q* and the visit counts *N* of each search edge (s, a) is updated at the end of the simulation. At search completion, the most visited action from the root state (action with highest *N(s, a)*) is the next move.

To handle the orders-of-magnitude-more computation required by combining MCTS with deep neural networks, AlphaGo executes multi-threaded simulations on CPUs and computes policy and value networks in parallel on GPUs. The final version of AlphaGo used 40 search threads, 48 CPUs, and 8 GPUs. A distributed version of AlphaGo that can run on multiple multiple machines with 40 search threads, 1,202 CPUs and 176 GPUs was also implemented.

This distributed version of AlphaGo won 77% of games versus the single-machine AlphaGo and 100% of games against all other Go programs. It also won 5 straight matches against Fan Hui, a professional 2 dan and the winner of the 2013, 2014 and 2015 European Go championships. The supervised and reinforcement learning employed in AlphaGo showcased the automatic intake of tens of millions of board positions and many more potential moves in contrast to the hand-crafted strategies employed by Deep Blue's chess strategy. When playing against Fan Hui, AlphaGo evaluated thousands of times fewer positions than Deep Blue it its chess games versus Kasparov, indicating perhaps a more intelligent, human-like approach to move selection. Fan Hui's defeat was the first ever victory of a Go playing program over a professional-ranked human being in Go.