Research Review – AlphaGo by DeepMind

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

Background: Go is a two-player zero-sum game that attracts a lot of attention from AI researchers due to the complex nature of the game. While the number of possible sequences of moves for large games such as chess is around 35^{80}, this number is around 250^{150} for Go, making exhaustive search infeasible. Previous research has shown that reducing the depth of the search tree by using an approximate value function and reducing the breadth by sampling actions from a probability distribution can lead to achieving weak amateur level play in Go. A more successful approach has been using Monte Carlo Tree Search (MCTS) that use Monte Carlo rollouts that sample long sequences of actions without branching at all. However, the success with recent examples using this approach have been limited to strong amateur play, primarily owing to using shallow policies based on linear functions of input features.

Goal: Deep convolutional networks had a lot of success in fields such as image classification and playing Atari games based on visual inputs. The goal of the AlphaGo team was to use these networks in playing Go, by feeding the board state as a "visual input", to overcome the limitation of simplistic policies used during tree search and achieve better results against human professional players.

Methodology: The AlphaGo team primarily used two deep neural networks to get their agent to pick better moves. They used a "value network" to evaluate board positions, and a "policy network" to select moves. In other words, while the value network reduces the depth of the search tree, the policy network reduces the breadth, allowing the networks to be used with MCTS.

The policy network consisted a 13-layer convolutional network that trained with supervised learning (SL) using human expert moves taken from KGS Go Server. This was supplemented with a fast policy network similar to previous research to speed up action sampling during rollouts. The weights of the SL policy network were then transferred to a policy network with identical structure that used policy gradient reinforcement learning (RL). The RL policy network then played games with randomly selected previous versions of the policy network to maximise the outcome, which is winning more games. This random sampling of opponents also helped with avoiding overfitting to a particular policy.

The next step was to create a value network with a similar structure to the policy network, with the exception that it output a single prediction instead of a probability distribution. The network weights were regressed with state-outcome pairs and updated with stochastic gradient descent. An important differentiator of this network was the training set. Instead of simply feeding KGS dataset, a new self-play dataset was used that consisted 30 million distinct positions sampled from separate games to avoid memorising game outcomes. This approach reduced mean squared error (MSE) on the test set from 0.37 to 0.234, proving more generalisation considering the MSE on the training set was 0.226.

Finally, the team combined the policy and value networks with the MCTS algorithm, where the tree is descended completely without backup. At each state of the simulation, an action is selected that maximise the action value plus a bonus that is proportional to the prior probability which decays with each visit to encourage exploration. Once a leaf is reached, it is evaluated by the value network and by the fast rollout policy network, and then the results of these evaluations are combined. The mean evaluation of all simulations following an action and the visit count to that action is accumulated and the most visited action is picked at the end of the simulation.

Results: The SL policy network on its own achieved 57% win-rate against the test set, beating the state-of-the-art algorithms developed by other research teams which achieved 44.4% at max. The improved RL policy network that started with SL policy network weights and trained further with playing against previous versions of the policy network won more than 80% of the games against the SL policy network, proving the importance of additional training with randomised opponents of itself in previous iterations. The algorithm was then tested against the strongest open-source Go program "Pachi" that trained purely on human expert moves and executed 100k simulations per move, and RL policy network won 85% of the games. A final evaluation was done between the AlphaGo agent that combined policy and value networks with MCTS algorithm, and various Go programs where each agent was given 5s to make a move. Alpha Go achieved a 99.8% win rate against all other programs. A distributed version of AlphaGo that run on 1202 CPUs and 176 GPUs as opposed to 48 CPUs and 8 GPUs achieved 100% win rate against other programs, showing clear superiority against all other state-of-the-art programs. This evaluation was followed by an official match with Europena Go champion Fan Hui where he lost all 5 games against AlphaGo. The team highlighted that during this match, AlphaGo evaluated thousands of times fewer positions than Deep Blue did against its chess match against Kasparov, showing the power of intelligent position evaluation.