Summary - AlphaGo with Deep Neural Networks and Tree Search

By Nithin Devanand

One of the basic concepts in AI is deciding about what action can be taken as next step, from the details that is available to the agent. This is done by exploring the search tree. Many a times, we have come across the search space that grows overly huge and the agent had to resort to different techniques such as minimax, alpha beta pruning, iterative deepening etc… to reduce the search space. Evaluation function and heuristics played a big role in reducing the depth of search space. Deep Blue used these techniques to beat human at the game of chess. The game of Go was more complex with branching factor of ≈250 and depth of ≈150. Hence the search through all states pace was exhaustive. In this write up, I am going to summarize the article "Mastering the game of Go with deep neural networks and tree search"[1]

AlphaGo is a computer program developed by DeepMind aimed at defeating human professional player in the game of Go. The goal was to solve it with little domain knowledge or expert input about the game. AlphaGo brings together the techniques like Monte-carlo Tree Search(MCTS), Deep Convolutional Neural Networks to achieve it.

Design

In AlphaGo, the board positions are captured as a 19X19 image and is passed to the Convolutional Layer to have a representation. A new Deep Neural Network approach called Value networks is then used to evaluate the board positions and sampling actions using Policy networks to select the moves. These reduced the effective depth and breadth of the search tree. AlphaGo's neural network were trained with pipeline consisting of several layers of machine learning. The architecture consisted of 4 layers - with three policy neural network and a value network.

The first stage of pipeline – 13 layer **Supervised Learning(SL) policy network** were trained with 30 million positions from expert human moves. This network predicted expert moves. A faster, lesser accurate **Rollout Policy Network** was also trained from human expert positions, using a linear SoftMax of small pattern features with weights pi. This allowed to faster actions during monte-carlo simulations to predict next move.

The next stage **Reinforcement Learning(RL) Policy Network** of training pipeline aimed at improving policy network outcome by policy gradient reinforcement learning. This training involved self-play where current version of the RL policy network played against randomly selected previous version of it. The reward functions were used, and weight updated at each time step which eventually led to a stronger policy. The final stage **Value Neural Network** in training pipeline, outputs a single prediction of the outcome for the current position. The value network weights were trained by regression on state-outcome pair.

Initially, when the game outcomes were predicted from data with complete games, the value network memorized the game outcome rather than generalizing for next move. It resulted in overfitting. To overcome this issue, data set of 30 million distinct positions, each sampled from separate game was used for self-play in RL policy network until game ended.

How AlphaGo Searches for next move?

AlphaGo uses Monte-Carlo Tree Search(MCTS) to search for the next best move in combining the power of policy and value networks. The MCTS searches in four phases a. Selection b. Expansion c. Evaluation and d. Backup.

Each edge of the tree stores an action value(Q), visit count(N) and Prior Probability(P). Search starts from the root node and traverses the tree to reach a leaf node, which is **selected**. When it reaches the leaf node, it **Expands** further by calculating the prior probabilities using the SL policy network. When the visit count for a successor node exceeds a threshold, the successor state/node is added to the search tree. The leaf node is then evaluated using two different ways, one by value network and second by random **simulations** until the end of the game, using fast rollout policy. These evaluations are then combined into leaf evaluation. At the end of simulation, the action values and visit counts are rolled up and each traversed edge is updated. When the search completes, algorithm will select the most visited move from the root position.

Summary

AlphaGo was able to achieve the goal of beating human professional player Fan Hui, a feat previously believed to be at least a decade away. AlphaGo exceeded the performance of all other Go programs with four handicap stones, with just value network with just rollout and even without rollouts. Distributed AlphaGo integrates Supervised, Reinforcement learning in high performance tree search engine which can be scaled

References

[1] Mastering the game of Go with deep neural networks and tree search - https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf

[2] Montecarlo Tree Search- https://www.youtube.com/watch?v=onBYsen2_eA