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

1. Introduction

The present review discusses the development of Google DeepMind's AlphaGo, an artificial intelligence computer program designed to play the 2-player board game Go (1). In Go, playing pieces are called stones – one player takes the white stones and the other takes the black. Stones are placed by players in turns on unoccupied line intersections (points) on a game board with a 19 x 19 grid of lines. Stones cannot be removed once placed, but are removed when one or more stones get surrounded by the opponent's stones on all orthogonally adjacent points – this is known as capture. Winning the game is therefore a matter of finding strategies to surround a larger area of the board than the opponent. The paper points out that applying exhaustive search strategies to games with enormous search spaces would be impractical. Therefore, there is a need to drastically reduce the breadth and depth of the search tree in large games like Go. To this end, the DeepMind team employed a novel combination of Monte Carlo Tree Search (MCTS) and deep neural networks to build AlphaGo, which achieved superhuman performance. Notably, the high performance was demonstrated in a 2016 match against the human European Go champion, where the program defeated the human 5 games to 0.

2. Methods

The AlphaGo program takes as input a 19 x 19 image of the game board and uses three "policy networks" to select moves and a "value network" to evaluate board positions. Here, the policy networks include a supervised learning (SL) policy network p_σ, a fast policy network p_π, a reinforcement learning (RL) policy network p_ρ. The SL policy p_σ is a 13-layer convolutional neural network, trained by a dataset of human expert moves, to predict the next best move. The fast rollout policy p_π is trained to rapidly and randomly sample the search space and predict the next best move by playing out the game to maximum depth in each rollout. It should be noted that the fast policy is faster but less accurate than the SL policy network. In the second training stage, AlphaGo uses the RL policy network (which is initially identical to the SL policy network) to generate a new dataset by making it play against randomly selected previous versions of itself many times. Network weights of winners are updated at each time step to maximise the outcome (more wins). Then, the program trains the value network v_0 using data from games of self-play to assign a score to each current player position based on its probability of winning or losing. To test the whole program, the paper presents two implementations of AlphaGo – a single machine version (40 search threads, 48 CPUs, and 8 GPUs) and a distributed version (multiple machines; 40 search threads, 1,202 CPUs and 176 GPUs). The novelty of the AlphaGo program lies in its combination of supervised and reinforcement learning to train deep neural networks and all these evaluations tie in with MCTS in the search algorithm.

3. Results

Both single-machine and distributed versions of AlphaGo outperformed other computer Go programs, with the performance of the distributed version higher than that of the single-machine one. Perhaps more noteworthy is that in the Go match against Fan Hui, AlphaGo not only include the fact that the program defeated the human 5 games to 0, but also that it selected thousands of fewer positions than IBM's DeepBlue did in the chess match against the world champion Kasparov.

Reference

1. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016 Jan 28;529(7587):484–489.