Alpha Go Nature Paper

Paper Here

Paper's Goals and Techniques

The paper's goal was to find a way to build an effective Go program using neural networks to evaluate board positions and select the best move during game play. Due to the complexity of Go, there has yet to be such a program. This because of the large number of possible moves leading to a large search depth (length of game) and large search breadth (legal moves from a player's position). Common approaches to this problem has been to use Monte Carlo Tree Search which can simulate a game to completeness without branching the tree search. Go programs such as Crazy Stone, Pachi, and Fuego use MCTS but have yet to be a competitive to professional, human Go players.

Deepmind proposes a solution that uses a combination of two deep neural networks to evaluate the board's position (finding the "policy"), one neural network to select a move (determining the "value"), and MCTS search to use these networks to quickly simulate move sequences and select the best move.

In order to find the "policy", Deepmind proposes a combination of two deep neural networks. First a Supervised Learning (SL) network is trained. It's made of 13 layers consisting of convolutional layers and nonlinear rectifiers. The game's board state, a 19x19 image, is what's inputed into the network to be convoluted. The network has a softmax layer at the end to provide a probability distribution of the legal moves available. The SL network was trained on 30 million different board state's provied by the KGS Go Server dataset. Stochastic Gradient Ascent was use to optimize the network by maximizing the expected outcome. The training accuracy was at about 57%.

The second deep neural network used to improve finding the policy, was a Reinforcement Learning Network (RL-policy). The RL-policy network picked up where the SL network left off by being initialized with the same weights as the resulting weights of the SL network. The RL-policy network then was trained by playing itself. To prevent overfitting the RL-policy would play against an early iteration of the RL-policy network that was randomly selected. Each iteration was a game simulated to completeness. After each game the RL-policy network would be rewarded $+1$ for winning and a -1 for losing. This reward was used to update the weights. After training the RL-policy network was evaluated against the previously trained SL network and the Pachi Go program. It won 80% of the time against the SL network and 85% of the time against the Pachi program.

The network trained for move selection was another Reinforcement Learning Network (RL-value). To train this network, 30 million distinct positions were generated from the training of the RL-policy network. The RL-value trained on pairs of board-position and moves taken using regression. Stochastic Gradient Descent was used to optimize the network and Mean Squared Error was use to compute the cost. Unlike the policy networks the RL-value network returns a single value.

Finally, a Monte Carlos Tree Search was designed to use the policy networks and value networks. The MCTS algorithm combined the probability distribution of legal moves from the policy networks and the move's value from the RL-value network to quickly simulate games and find the best move.

Summary of Paper's Results

To evaluate the Go program, the used two versions:

- A single-machine version that executed simulations asynchronously on multiple threads using 40 threads, 48 CPUs, and 8 GPUs. And,
- A distributed version across multiple machines and using 40 threads, 1,202 CPUs, and 176 GPUs

The program was evaluated against 5 Go programs: Crazy Stone, Zen, Pachi, Fuego, and GnuGo. Deepmind's program performed extremely well winning 494 out 495 games. The program was then evaluated against a human player, European champion Fan Hui. The Go program won all 5 games, becoming the first program to defeat a professional Go player.