Research review: Mastering the game of Go with deep neural networks and tree search.

AlphaGo is more human-like unlike previous approaches. Instead of using brute force over millions of positions, AlphaGo looks ahead by playing the remainder of game in its imagination, using Monte-Carlo tree Search. Apart from using MCT search it uses deep learning neural networks to guide its search, making the search more powerful.
There are two types of neural networks used:
1. Policy network
2. Value network

During each simulated game play, the policy network suggests intelligent moves to play, while the value network evaluates the position that is reached. Finally AlphaGo chooses the most successful move in that simulation.

The value network is used at the leaf nodes to reduce the depth of the tree search.
The policy network is used to reduce the breadth of the search from a node (guiding towards promising immediate actions).

Supervised-learning (SL) policy network: looks at a board position and attempts to decide the best next move to make.

Policy nets are trained to do the moves that most likely a human would do given a board state. It means that this net outputs a histogram that shows the probability of each action given an input state. Finally picking up the move with highest probability. The policy network is a convolutional neural network. It takes in a state and computes the next best action. In this case the state is the game board (19 x 19) along with some in game parameters (color of piece, number of moves made, etc). This comes out to a total input of 19 x 19 x 48. It has various convolution and pooling hidden layers much like the ones used for image recognition.
The SL policy network is trained on millions of examples of moves made by strong human players on KGS. It's the most human-like part of AlphaGo, in that its goal is only to replicate the move choices of strong human players.

The "reinforced-learning (RL) policy network", is a refined version that is trained more intensively using millions of additional simulated games. In contrast to the basic training described above, which only teaches the network to mimic a single human move, the advanced training plays out each simulated game to the end in order to teach the network which next moves are likely to lead to winning the game. The team synthesized millions of training games by playing the reinforced-learning (RL) policy network against previous editions of itself from earlier training iterations.

The “rollout network” is a simplified version which doesn’t look at the entire 19×19 board, but instead only looks at a smaller window around the opponent’s previous move and the new move it’s considering. This lowers the strength of RL policy network, but it computes about 1,000 times faster, making it suitable for reading calculations.

The Value network:
Instead of guessing a specific next move, it estimates the chance of each player winning the game, given a board position. It provides overall position judgement. This judgment is only approximate, but it’s useful for speeding up reading. By classifying potential future positions as “good” or “bad”, AlphaGo can decide whether or not to read more deeply along a particular variation. If the position evaluator says a particular variation looks bad, then the AI can skip reading any more moves along that line of play. The position evaluator is trained on millions of example board positions.

Reading is accomplished with the same Monte Carlo Tree Search (MCTS) algorithm used by most state-of-the-art Go AIs. But AlphaGo out-smarts other AIs by making more intelligent guesses about which variations to explore, and how deeply to explore them.

Work-flow:

1. From the given board state, choose few next possible moves. Only SL policy network is used. Using RL policy network restricted the search space and only focused on the ‘obvious best’ move. It means, the move which doesn’t seem promising right now but is a really potential one to lead to a win in the later stages of game might get ignored if we use RL policy network.

2. For each possible next move, evaluate the quality of that move in any of the two ways: Either use the position evaluator on board state after forecasting that move or run a deeper Monte Carlo simulation that reads into the future after that move.

Mixing Coefficient, or the judgment parameter which is actually a 50/50 mix which in turn judges the quality equally using the position evaluator and the simulated rollouts by MTCS.

AlphaGo also had 99.8% winning rate against other computer Go programs, demonstrating its dominance. DeepMind’s research also revealed the level of computational power required to conquer such a task. The final version of AlphaGo used 40 search threads, 48 CPUs, and 8 GPUs. Though a distributed version with 40 search threads, 1,202 CPUs, and 176 GPUs was also implemented, the program’s competitiveness in terms of Elo rating exhibited diminishing returns.