Alpha Go relies on alterations and combinations of past game-playing algorithms. The Monte Carlo tree search algorithm, for example, estimates values of different states in a tree by performing rollouts on what's determined to be the most promising move. This allows the AI to sample different actions. This has been a popular means of past go-playing AIs. Other game-playing algorithms, such as depth-first minimax search with alpha beta pruning, has historically not been as effective in the game of Go as it has been in games like chess, checkers or Othello. Monte Carlo is an alternative to minimax that estimates the optimal value of interior nodes through two approximations(one for estimating the value function of a simulation, and another for a simulation policy that chooses actions based on a search control function with a bonus for encouraging exploration). Over time, the simulation gains increasingly accurate statistics.

Symmetries are another solution that are commonly used in game playing algorithms, but they "actually hurt performance in larger networks." This is because of created limitations around identifying asymmetric patterns. However, Go does still take some advantage of symmetries to ensure they are not repeating computation for the same play in multiple orientations.

Alpha Go works as a combination of elements from these past strategies. The system works by passing the board as a 19x19 image that is fed to a value and policy network to determine the next best move. The policy network(a probability distribution over legal actions) is used for sampling actions in a similar way to Monte Carlo. They also use a reinforcement learning policy network that learns the best moves through rounds of self play.

The value network(recursively computed via minimax or negamax search) predicts game winners based on learning from past games. This network works as a replacement to the rollouts found in Monte Carlo tree search, as "rollout-based position evaluation is frequently inaccurate."

The two evaluation functions are used as complementary to one another, where the value network runs much stronger & slower, and the rollouts of the policy network runs much faster with less accuracy.

Training was done both on self-play games and by pulling from the KGS Go server. Additional training was done by choosing moves from multiple games in a single round in order to keep the network from falling in to repetitive patterns. Time-based weights are also added to actions for this purpose. By adding a weight that decays the value over time, the algorithm is encourages to continue exploring. Finally, further training of the system comprised of live games against ranked human opponents.

Due to the computational power required to run Alpha go, developers used an asynchronous multi-threaded search to execute simulations on CPUs, while the policy and value network computations run in parallel on GPUs. The program uses 40 search threads, 48CPUs and 8 GPUs. A distributed version also exists that runs with 1,202 CPUs and 176 GPUs.

Alpha Go resigns if its overall evaluation drops below 10% probability, though with it's high winning rates it doesn't happen often. In a final test against top player Fan Hui, Alpha Go was able to run with thousands fewer evaluations than Deep Blue. Developers credit this to the positions being selected more intelligently(using the policy network) and precisely(via the value network).