This is a summary of the following paper:
https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf

Mastering the Game of Go With Deep Neural Networks and Tree Search: Summary and Review

AlphaGo is an AI system designed to learn and play Go. Go, like Chess, is a fully observable, discrete, non-stochastic, adversarial, game. The complexity of Go, like Chess is in the search space available. Chess has an average of 35 possible moves each turn, and an average of 80 turns per game. Go has about 250 possible moves each turn and on average 150 turns per game. As a result, exhaustive search is infeasible when playing Go

Because search is intractable, most AI agents, instead of valuing a move based on all of the move's possible outcomes, create an approximation (using a heuristic) to value the best move. State-of-the-art systems use Monte Carlo tree search to reduce the number of possible moves considered based on a policy function.

With AlphaGo, Google's research team demonstrates that deep convolutional neural networks are a feasible alternative to Monte Carlo tree search, and when combined with Monte Carlo search, approximate the value of Go moves better than other state-of-the-art systems. Though not mentioned in this paper: convolutional neural networks are an emerging algorithm in the field of machine learning. Classically, these algorithms are computationally infeasible, but General Purpose Graphics Processing Units (GPGPUs) can reduce the training time of these networks from months to days. Google chose to leverage this advancement to build AlphaGo.

The high level structure of the system is as follows: given the current state of the board, the agent uses a search tree to explore possible moves. It selects which moves to explore and expand based on the "value" of the move which is determined by a "Value Network." It also uses the weighted sum of a "Policy Network" and a Monte Carlo rollout to determine the probability that a 'winning expert' would choose the move. The algorithm is less likely to explore a node the more times that node has been visited; this encourages the algorithm to branch out. Finally, the most visited node is selected as the chosen move for that turn.

It is notable that when determining the policy network output, even when the weight of the Monte Carlo rollout is reduced to zero, this output is still effective. This implies that the Policy Network it is a suitable alternative to Monte Carlo rollout in determining the value of a move. With a weight of 0.5, the algorithm functions optimally implying that the two methods are complementary. For the interested reader, an appendix is included at the end of this summary detailing the network structures.

In a tournament against other state of the art AI Go agents, AlphaGo won 99.8% of matches. With a handycap of 4 (opposing agent gets 4 free moves), it still won >77% of matches against state-of-the-art Go agents. In 2015, AlphaGo beat Fan Hui (professional 2 dan and winner of the European Go Championships in 2013, 2014, and 2015) five matches to zero.

Appendix: Structure of Neural Networks used in AlphaGo

The Policy Network accepts a board state as input, has 13 alternating convolutional and nonlinear rectifying layers, and finishes with a softmax layer which outputs an array of probabilities representing the probability of likely moves by an expert player. The network is trained using 30 million games from the KGS Go Server. The network predicted moves by pro players with a 57% accuracy compared to state-of-the-art systems which predict 44.4% accuracy.

This network is then trained using a set of games played against itself (actually against random previous versions to prevent over-fitting to the opposing agent). This value more accurately reflects winning moves rather than moves that experts would choose. The output of this net is calculated once on each board state as the search algorithm expands nodes and is weighted with the output of a Monte Carlo rollout on that same board state. However, even when the weight of the Monte Carlo rollout is reduced to zero, this output is still effective, implying that it is a suitable alternative. With a weight of 0.5, the algorithm functions optimally implying that the two methods are complementary.

The Value Network uses an identical structure to the Policy Network, except that the output is a single number predicting the outcome of the game given the board state. This follows the same theory used in deep neural net image recognition - the first layer feeds to a layer that recognizes small features, and those features are fed to a layer which recognizes sill larger features until the network can identify the board as a “winning” or “losing” board state. This network is trained on a set of 30 million random states taken from different games (since consecutive board states in the same game are not isolated). This output is used as noted above to determine which leaf nodes to expand.