How AlphaGo beat Lee Sedol

In 2016 the Deepmind team achieved something that was thought to be at least a decade away, they developed a program that beat the world champion in the board game "Go". In this summary we'll explore how Deepmind did it.

Perfect information games all have an optimal value function $v^*(s)$. This value function takes as input the board state and determines the output of the game under perfect play by all players. Games are solved by exhaustively computing $v^*(s)$ for all position until the game ends. In larger games such as Chess and Go, the game tree becomes gigantic. This makes computing v*(s) for every board position infeasible. This difficulty is overcome using 2 techniques: [1]

1. The tree is truncated below a certain level and replaced with an approximate value function $v(s) \approx v^*(s)$
2. By sampling actions at random and creating a probability distribution of wins/losses. An example of this is Monte Carlo Tree Search (MCTS)

As mentioned above, to explore such massive game trees such as the trees in Chess and Go, an approximation of the optimality function is used $v(s) \approx v^*(s)$. The way it has been used so far was by manually designing an evaluation function $v(s)$, designed by consulting experts of the game. When the winner is easily determined by looking at the board, manually designing an evaluation function is more useful. This applies to many games including Chess. However, in Go it is not usually obvious who is going to win just by looking at the board.

The Deep Networks

To overcome this, Deepmind used deep neural networks. The state of the board is used as an image that is then used as an input to a deep convolutional neural network. That network is trained on large datasets of previous games. The end result is a network that takes as input the board state as an image and outputs the likelihood of winning the game, thus acting as an evaluation function. This network is named the "value network".

To explore the massive game tree of Go, the team needed another technique to tell the algorithm which branches are worth exploring and which are not. To achieve this, another deep neural network is used. Unlike the value network above that takes as input the board state and outputs the likelihood of winning, this network takes the board state as input and outputs a probability distribution across all the available board positions. Meaning it outputs the probability of each available board position to lead to winning game.

This network is used to determine which branches are worth exploring and which are not, thus significantly reducing the search tree. This network was named the Policy Network. They implemented 2 versions of this policy network; one slow but accurate and one much faster but a little less accurate. The faster policy network was named the rollout network. The rollout network is used during search as will be explained in the next section.

Monte Carlo Tree Search (MCTS)

The MCTS is a search technique which depends on picking random branches from the game tree and registering which lead to a win and which to a loss. The process is repeated a large number of times creating a probability distribution among the nodes showing which ones are more likely to lead to a win; higher probability nodes are typically further explored. AlphaGo uses a version of the MCTS that uses the rollout network mentioned in the previous section to choose which branches to explore. The higher the score from the rollout network the higher the probability it will be explored by the MCTS.

The Final Score

AlphaGo does all of this before making any move. It chooses a move based on a final score that is calculated by combining the result from the value network and the result from MCTS guided by the policy network. The combined score gives the algorithm a mix between intuition provided by the value network, and reflection provided by the simulated games done by the MCTS.

So in the end there are 3 main components that make up the AlphaGo algorithm:

1. The value network
2. The policy networks (including the rollout)
3. The MCTS

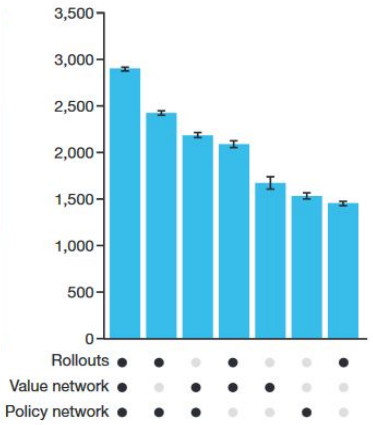

Figure 1: a graph comparing the performance of the different versions of AlphaGo [1]

When the team tested the performance of AlphaGo using only 1 or 2 of those components, the algorithm performed worse. Only by combining the three components did AlphaGo reach the impressive performance it did as shown in Figure 1.

Results

The Elo rating is used to measure a player's strength in playing Go. To evaluate AlphaGo's strength, it was compared with the Elo rating of several state of the art Go playing algorithms. The results are shown in Figure 2. Fan Hui, a professional Go player, was also added to the comparison to give some perspective.

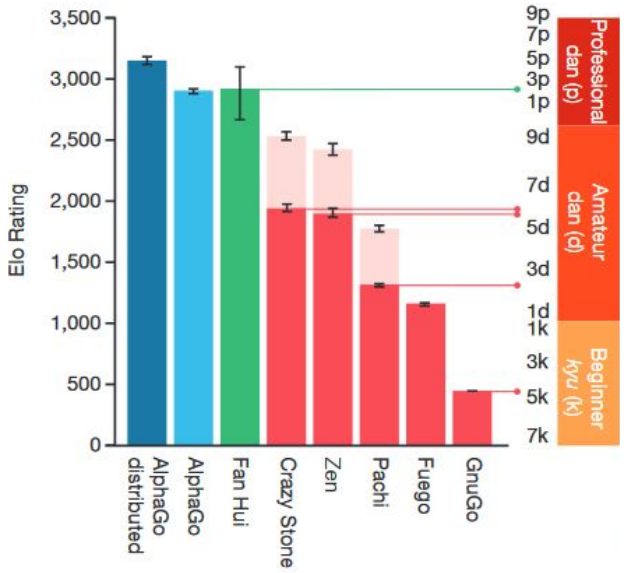

Figure 2: a comparison between AlphaGo and other Algorithms [1]

Figure 2 shows that AlphaGo surpases all the best game playing algorithms, as well as being on par with the professional player for the undistributed version of the algorithm. The distributed AlphaGo even surpases the professional player by a non-trivial margin. This alone was an impressive result on its own. In addition, in 2016 AlphaGo succeeded in defeating Lee Sedol with a score of 4 - 1, thus achieving what was thought to be unreachable for at least a decade to come.

References:

1. Mastering the game of Go with deep neural networks and tree search *Nature*, Vol. 529, No. 7587. (27 January 2016), pp. 484-489, doi:10.1038/nature16961 by David Silver, Aja Huang, Chris J. Maddison, et al.