Summary of AlphaGo algorithm performance

A minimax is a strategy of considering every possible move that a player can make in a given position, every opponent response, until all possible positions are generated for a few moves ahead, evaluating them using relatively simple evaluation function and then backtracking with assumption that at each turn the players select the best available responce. It proved to be very effective for selecting moves in chess and chekers and has lead to superhuman performance. However there are two problems of applying the minimax stratagy to Go. The number of moves that are possible in Go at the beiginning is ~250 as compared to chess ~35. This alone makes exhaustive search infeasible. Another problem in evaluating Go is that it is much harder to accurately estimate a value of a given Go position compared to chess especially in the beginning of games.

To solve these problems the previous efforts of go engine designers focused on

1) reducing the effective branching factor by limiting the number of moves that needs to be considered in a given position. It is done by using a policy for selecting best moves based on analysis of expert games.

2) estimating the value of the position with the Monte Carlo Rollouts and Monte Carlo Tree Searches. Monte Carlo Rollout means playing the game to the end from a given position using a probabilistic algorithm for selecting next move. This is repeated many times. The value of the position is then the percentage of games that was won during simulation. Monte Carlo Tree Search builds a search tree similar to a regular minimax tree and estimates the value of each node using Monte Carlo Rollouts.

The AlphaGo team applied deep convolutional neural networks to improve several aspects of the above process. Using the expert games as a guide the team trained two supervised learning network policies for selecting next move: Fast policy that generates less accurate moves quickly for Monte Carlo Rollouts and the SL policy network Ps for accurately predicting expert moves. The SL policy is about 57% accurate which considerably improved on previous results of 42% accurasy. The SL policy network was then used as initial conditions for the reinforced learning policy network Pr. The aim of Pr network was winning the game instead of predicting the expert moves. To train Pr network, its most current version played games against its random previous iterations . The network trained in this way won 80% of the games against the original SL network. The final step was to train a value network that would estimate a value of a given position. To do it millions of games were played by RL network against itself. The generated positions were used to teach the value network to estimate the value of the position.

Finally all the networks were combined to be used during the game in move selection algorithm

The move selection algorithm works in the following manner. The root node corresponding to the present position is expanded using the SL Ps policy network. Each of the next nodes is given a score by the network. Only the most promising node is chosen and the procedure repeats at the deeper level until the specified search depth is reached. At this point the value of each leaf in this tree is evaluated using a weighted linear combination of the value network evaluation and the Monte-Carlo rollout using fast policy network. The results of the evaluation are then used to update the value of each node that would affect it's probability of selection during the next simulation. In addition, to encourage exploration the number of times the node was visited negatively affects its chances of being selected during the next simulations. Finally after several simulations are conducted the node visited most often is selected as the next move.

The AlphaGo significantly improved performance of Go engines, beating all of its competitors even when handicapped. It also achieved impressive results against a human master beating the former European Champion in official match 5-0

[1]Silver, D., Huang, A., Maddison, C.J. Mastering the game of Go with deep neural networks and tree search. Nature 529 484–489 (2016)