Payam Mousavi

A brief summary of the google Alpha-Go

Alpha-GO is a game-playing agent recently created by the google Deep mind. As of the publication date of the paper, the program was able to beat the European Go champion. Later, it went on the beat the world champion. Contrary to IBM's deep blue that beat Garry Kasparov (the world chess champion) in 1997, Deep Blue uses machine learning techniques to improve its performance by learning from playing other opponents as well as itself. The Deep blue used brute force tree search along with carefully designed (by domain experts) evaluation functions (i.e. Heuristics).

The Alpha-Go program is based on Reinforcement-Learning (RL) whereby an 'agent' is trained by obtaining rewards as it explores the search space. Additionally, deep neural networks are used to encode 'value' and 'policy' functions that are required in the RL algorithms. Specifically the following AI techniques are used:

1. Tree search procedures – A modified tree search algorithm is introduced that combines the state-of-the-art Monte-Carlo methods with the value and policy networks to obtain a more efficient way to search the tree.
2. Deep convolutional networks - To encode the evaluation function guiding the tree search. Note that these networks are 'trainable' and improve with more play. In total, one value network and two policy networks were used. The value network is similar to the chess evaluation function and provides an estimate for how good each position is. The policy networks in turn, will help in deciding which actions to take given the current state. The policy network was first trained on 30 million positions played by human experts and was able to achieve a predictive accuracy of 57%. Taking it a major step further, using RL, these policy networks were further trained by self-play using the outcome of each game as an additional training data point. The value network was then re-trained using the positions reached during the self-play. The value functions become better as the game goes on, as there are many more possibilities at the beginning of the game.

In brief, the following procedure is used in the tree search:

Each edge of the search tree stores an action value Q(s,a), visit counts N(s,a), and the prior probability of reaching that state P(s,a).The tree is traversed starting from the root node, and at each time step t, the action with the maximum value is chosen. Each node is evaluated using two different functions: by the value network, and by the outcome of a "random rollout" played until the terminal node. These two (very different) value functions are then combined into a single value function using a mixing parameter lambda. At the end of each simulation, all values including, the action values and the visit counts are updated. Upon the completion of the search, the algorithm picks the most visited move in the tree.