Vaishak Kumar

Artificial Intelligence Nanodegree

AlphaGo

Game playing agents face the task of winning games that have b^d possibilities where b is the branching factor and d is the depth of the game tree. In games such as Go where b=250 and d=150, it is impossible to exhaustively search the entire tree. Most game-playing agents employ two types of strategies:

1. Reduce the depth and at the last level approximate a utility value for that board
2. Reduce the branches by sampling actions randomly or based on some heuristic

The authors of this paper introduce the usage of deep learning alongside Monte Carlo Tree Search (MCTS) strategies to create a go player. The term deep learning refers to the usage of multi-layered artificial neural networks trained by gradient descent learning.

First a neural network $p_\sigma(a|s)$ is trained using supervising learning to predict what move an expert would make at a given game state. Input is fed in as a 19X19 image and the network uses convolutions to extract features from the image, and then uses them to predict the next move.

Then a reinforcement learning policy network p_ρ which is identical in structure to p_σ is initialized to the weights learnt by the supervised learning network. It is then trained by making it play games with its previous iterations. The reward function is zero at all non-terminal states, +1 for winning and -1 for losing. The reinforcement learning policy network won 80% of the matches over the supervised learning policy network.

Lastly, a neural network is trained to approximate the function $v^p(s)$ that predicts the value of a certain state. This network "learns" a value function whereas in other approaches such a function would be hand-engineered. The input is once again a 19X19 image of the board, and the neural network performs regression to output the utility value of the board.

The policy network and the value network are jointly used to determine the next move. A weighted average of the score from the value network and action score determined from a Monte Carlo rollout to the leaf node is used to determine the next move. The rollout probability of a given branch is given by supervised learning network at each step. However, the probability decays with each visit to that node so that the agent also explores new branches instead of always relying on exploiting moves guaranteed to maximize the chances of winning.

Obviously, AlphaGo performed better on machines with her computational capacity. AlphaGo run using distributed GPU computing beat all other commercial and open source Go playing software. Interestingly, pure value network evaluation without Monte Carlo rollouts was also enough to defeat all other software players. However, the mixed case (λ=0.5) provided the best

performance. AlphaGo achieved the remarkable feat of beating Fan Hui, a highly ranked professional Go player.