Research Paper Review: AlphaGo paper

by Anand Saha <anandsaha@gmail.com>

Introduction

Go is a two player, turn taking, deterministic game of perfect information.

Two main factors make Go very complex to solve:

- Go has an average branching factor ‘b’ of ~250 options per node (chess ~35)
- Go has an average depth ‘d’ of ~150 moves (chess ~80)

These factos make the state space of Go (b^d) enormous to search end to end using traditional techniques.

Paper’s Goals

- The goal was to create a computer Go program to beat world class Go professional players, without any handicaps on a 19x19 board, given that:
  - The search space of Go is just enormous
  - The evaluation of board positions and moves is challenging
  - Go combines game playing with human intuition, which is challening to imitate by a computer program

Techniques used

AlphaGo’s novel approach used two deep neural networks called *policy network* (for sampling actions) and *value network* (for evaluating positions), which were used in a *Monte Carlo Tree Search* program (for traversal) to determine best moves.

1) The **policy network** was used to predict the next set of moves which are likely to win probabilistically from a given board position - thereby reducing the branching factor.
   - This was achieved by training the policy network on 30 million moves from past games played by expert humans, using supervised learning.
   - Then it was made to play itself thousands of times, using reinforcement learning. This enabled the network to know what the winning moves are and also develop it’s own intuitions on the winning moves.
   - Once trained, the policy network produced a *policy function* which gives out the probability distribution of game actions given a game state.
   - The policy network had 13 layers and had an accuracy of 57% on test dataset (which is impressive)
2) The **value network** was used to estimate the outcome (win/loss) of a board position without actually traversing till the end of the tree (thereby reducing the depth)
   - The value network produced a *value function* which predicts the outcome of a board position played using a particular policy function (The strongest policy function was choosen from (1)).
   - The value networks were trained using the policy functions via reinforcement learning by self play. This resulted in generalisation of the approach.
3) Finally, it used **Monte Carlo Tree Search** (MCTS) which made use of (1) and (2) to determine the best move given a board state.
   - The best move is determined by simulating game play.
   - The crux of MCTS algorithm is in a loop which has four main steps. (Note that when you are at a certain board state, you have knowledge of the game tree only to a certain depth beyond which you haven’t expanded the tree yet. Remember, this is an enormous tree.)
     - **Selection**: When you are at a state (node) in the tree, you try to select a winning action to traverse to the next state. But how do you know which are the winning actions? This is given by the policy function. The policy function takes in a state and gives back an action which has high probability of winning. (That’s how the policy network is used). You keep selecting subsequent actions and traverse down the tree until you come to a leaf state (this is not leaf of the *actual tree*, this is leaf of the *known tree*)

        - **Expansion**: Once you have reached a leaf state, it has to be expanded to reveal potential winning moves. It is possible that at this state, there are very many possible actions to take to reach various winning states. The policy function is invoked to generate probability distribution of actions for possible child states.
        - **Simulation**: In this step, first the value function is used to estimate the outcome of each action (win/lose) revealed in expansion step. Then the potentially winning child states are taken and games are simulated by taking actions choosen based on a policy. This is called a *rollout policy*. A simple policy can be the random policy, where actions are choosen at random. AlphaGo used a fast rollout policy. Each of these simulations will eventually converge to a win or lose terminal state.
        - **Back Propagation**: In this step, the outcome of the terminal state is propagated up through the path to the root. This information is then propagated up the tree and the statistics (action value, visit count and prior probability) of all the nodes in the path to root are updated.
    - With enough simulation, the stats tend to favor the strongest moves and their score goes up.
    - The strongest of them is then selected as the next move.
    - The simulation happens as many times as possible within the cutoff time for a move.

AlphaGo also used **Computer Vision** to ingest the board position as a 19x19 image and used convolution networks to construct a representation of the position.

Paper's results

- Technically:
    - The paper introduced a new approach to solving Go, by efficiently integrating Policy networks and Value networks with MCTS
    - Made use of Supervised learning along with Reinforcement learning to generate both logic and intuition of the game
- Impact wise:
    - AlphaGo achieved this feat at a time when the community had estimated another decade for this to be a reality.
    - AlphaGo defeated the human European Go champion 5-0, and a 9-dan rank player 4-1.
    - AlphaGo defeated all of it's fellow computer Go players