DeepMind's ALPHA-GO Game Agent Review Paper Reading

By Aaron Balson

08-March-2017

Introduction

Go, a complex game that demands more of human intuition than sheer logic (unlike Chess), was considered to be one of AI's hardest feat to defeat professional human players because of its search space with huge branching (b~250) and depth (d~150) factors. It was thought the humans would prevail at least another decade – before DeepMind's Alpha-Go program bested the reigning European champion Fan Hui by 5-0 in October 2015, thus shocking the world and ending the man vs machine series with dominance.

Design Goals

Alpha-Go team decided to reduce the depth of search by use of value functions to predict the outcome of state ***(s)*** and thereby truncating subtrees. The breadth of the search may be reduced by sampling actions from a policy ***p(a|s)*** that is probability distribution over possible moves ***(a)*** in position ***(s)***.

Network Techniques Used

Based on the design goals, the agent cannot be successful if it had to apply those same techniques used in Chess or Othello (with inferential data). Hence it has to employ Monte Carlo Tree Search (MCTS) algorithm with deep convolutional neural networks. Agent is trained using a pipeline of several stages of machine learning to read the board state in the form of 19x19 image and select the best possible move with less look ahead search.

The first stage of pipeline is a 13-layer supervised learning policy network (SL) which is built on prior work of 30 million positions from the KGS Go Server for predicting expert moves using stochastic gradient ascent, hence this layer is used to pick the most likely moves during game (thus reducing breadth search). The SL policy network $\boldsymbol{p_\sigma(a|s)}$ alternates between convolutional layers with weights σ, and rectifier nonlinearities. A final softmax layer outputs a probability distribution over all legal moves ***(a)***.

The second stage of pipeline improves the first one by policy gradient reinforcement learning (RL). The RL policy network $\mathbf{(p_p)}$ is identical at the beginning to SL $\mathbf{(p_\sigma)}$ but it is improved by making it play against itself for millions of times over time. A new data set is generated by playing games of self-play with the RL policy network. RL is trained to pick the best possible move (towards winning) for the given board state during the game.

There is a third policy called Fast Rollout (FR) which is trained like SL to predict the most likely moves but it is much faster (and less accurate) than SL, and is used during search phase to evaluate value function by fast rollout till end game terminal leaf.

The final stage of pipeline, Value Network, focuses on position evaluation estimating a value function $\boldsymbol{v_p(s)}$ that predicts the outcome from position ($\boldsymbol{s}$) of games played by using policy ($\boldsymbol{p}$) for both players. It estimates the value function $\boldsymbol{vp_\rho}$ for our strongest policy, using the RL policy network ($\boldsymbol{p_\rho}$). This neural network has a similar architecture to the policy network, but outputs a single prediction instead of a probability distribution.

Search Techniques Used

Alpha-Go combines the policy and value networks in an MCTS algorithm that selects actions by look-ahead search.

a. [Selection phase] Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $\boldsymbol{u(P)}$ that depends on a stored prior probability P for that edge.
b. [Expansion phase] The leaf node may be expanded; the new node is processed once by the policy network $p\sigma$ and the output probabilities are stored as prior probabilities P for each action.
c. [Evaluation phase] At the end of a simulation, the leaf node is evaluated in two ways: using the value network $\boldsymbol{v\vartheta}$; and by running a rollout to the end of the game with the fast rollout policy $p\pi$, then computing the winner with function $\boldsymbol{r}$.
d. [Backup phase] Action values Q are updated to track the mean value of all evaluations $\boldsymbol{r(\cdot)}$ and $\boldsymbol{v\vartheta(\cdot)}$ in the subtree below that action.

It is worth noting that the SL policy network $p\sigma$ performed better in Alpha-Go than the stronger RL policy network $p\rho$, presumably because humans select a diverse beam of promising moves, whereas RL optimizes for the single best move. However, the value function $v\vartheta(s) \approx v\, p\rho(s)$ derived from the stronger RL policy network performed better in Alpha-Go than a value function $v\vartheta(s) \approx v\, p\sigma(s)$ derived from the SL policy network.

Agent Performance

Evaluating policy and value networks requires several orders of magnitude more computation than traditional search heuristics. To efficiently combine MCTS with deep neural networks, Alpha-Go uses an asynchronous multi-threaded search that executes simulations on CPUs, and computes policy and value networks in parallel on GPUs. The final version of Alpha-Go used 40 search threads, 48 CPUs, and 8 GPUs. We also implemented a distributed version of Alpha-Go that exploited multiple machines, 40 search threads, 1,202 CPUs and 176 GPUs.

Result

The results of the tournament suggest that single machine Alpha-Go is many *dan* ranks stronger than any previous Go program, winning 494 out of 495 games (99.8%) against other Go programs. To provide a greater challenge to Alpha-Go, it also played games with four handicap stones (that is, free moves for the opponent); Alpha-Go won 77%, 86%, and 99% of handicap games against Crazy Stone, Zen and Pachi, respectively. The distributed version of Alpha-Go was significantly stronger, winning 77% of games against single-machine Alpha-Go and 100% of its games against other programs. Finally, the distributed version of Alpha-Go against Fan Hui, a professional 2 *dan*, and the winner of the 2013, 2014 and 2015 European Go championships, won the match 5 games to 0. This is the first time that a computer Go program has defeated a human professional player, without handicap, in the full game of Go—a feat that was previously believed to be at least a decade away.