AlphaGo by the DeepMind Team

Goals and Techniques

Go has been viewed as the most challenging of classic games for AI owning to its enormous search space and difficulty of evaluating board position and moves. In fact, general consent estimated that a computer program beating a human professional player in the full-size game of Go to be at least a decade away. Using well known techniques, advancements in machines learning and Artifical Intelligence, and computing advancements, AlphaGo's goal was to attempt to beat professional players in full-size games of Go.

Monte Carlo Tree Search

Monte Carlo Tree Search is the analysis of focusing on the most promising move by expanding the search tree based on random sampling of the search space. Here are the steps:
Selection – starting from root and select successive child nodes down to the leaf, making decisions based on edges with maximum action values Q and bonus u(P).
Expansion – the leave node may be expanded. The new node is processed by the policy network and output probabilities are stored as prior probabilities for each action.
Evaluation – leaf nodes are evaluated using the value network, and by running a rollout to the end of the game with fast rollout policy
Backup – Action values Qs are updated to track the mean value of all evaluations

Fast Rollout Policy

It is an evaluation function trained to predict expert human moves in a dataset of positions. It is the probability of winning given a certain state, and taking a certain action. To prevent using the full search tree, fast rollout policy is use quite frequently to evaluate the Monte Carlo Search Tree.

Supervised Learning of Policy Networks

Using a 13-layer CNN of weights and rectifier nonlinearities, the model trained and validated on 30 million positions from the KGS Go Server database. The SLPN achieved an accuracy of 57% predicting expert player moves.

Reinforcement Learning of Policy Networks

Basing off of the model weights from SLPN, RLPN further improved the policy network by playing against randomly selected previous iteration of the policy networks. To prevent from overfitting, RLPN played against pool of randomized opponents. For the reinforcements, reward function r(s) is defined as follows:

- $r(s) = 0$ for $t < T$ (reached time limit)
- r(final state) = 1 (won)
- r(final state) = -1 (lost)

Reinforcement Learning of Value Networks

RLVN has a similar architecture to the RLPN, but outputs a prediction instead of probability distribution. The weights of RLVN were trained with regression on state and outcome pairs. The goal is to minimize MSE between predicted value and the corresponding outcome.

To prevent overfitting, a dataset of 30 million distinct positions were generated (self play). Note that these data came from different games to minimize correlation between data points.

Final Algorithm

Using the Monte Carlo Tree Search at each time step, an optimal action is selection for state s.

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))$$

u(s, a) is the bonus function that is proportional to the prior probability but decays with repeated visits to encourage exploration.

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

When the algorithm reaches a leaf node, the leaf node is evaluated in two ways:

1. Value Network
2. Fast Rollout Policy

It turned out that the optimal algorithm took in account of both evaluation.

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

For each leaf node, number of visits are recorded as well as the average action value. Once the search is completed, the algorithm chose the most visited move from the root position.

Results

With 5 second computation time per move:

Single Machine AlphaGo vs. any previous Go program: 99.8% win rate

Single Machine AlphaGo 4 handicap vs. Crazy Stone: 77% win rate

Single Machine AlphaGo 4 handicap vs. Zen: 86% win rate

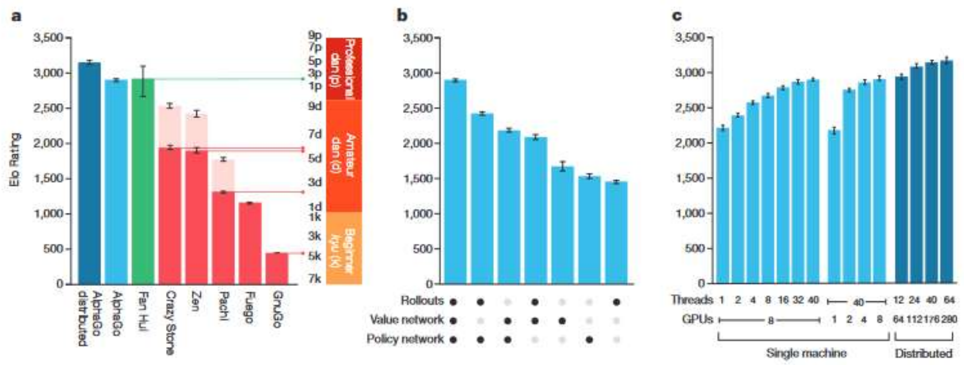
Single Machine AlphaGo 4 handicap vs. Pachi: 99% win rate

Distributed AlphaGo vs. Single Machine Alpha Go: 77% win rate

Distributed AlphaGo vs. others: 100% win rate

Without 5 second computation time per move:

AlphaGo vs. Fan Hui: 100% win rate



a. Tournament achievements (Elo Rating) between different Go algorithms. Showing 95% confidence levels, showing Distributed AlphaGo was a near 5 dan player

b. Tournament achievements between AlphaGo algorithm configuration and combinations, showing a combination of quick rollout, value and policy network can achieve the best Elo rating.

c. Tournament achievements between AlphaGo hardware configuration and combinations, showing how Elo rating performance correlate with number of threads, GPUs, and distributed vs. single machine.