AlphaGo review

Exhaustive search of Go's game tree is unfeasible due to its large game **breadth**, b (number of legal moves per position) and **depth**, d (game length).

Depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s.

Breadth of the search may be reduced by sampling actions from a policy of legal actions a in position s. For example, **Monte Carlo rollouts** search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over Monte Carlo rollouts can provide an effective position evaluation.

Previously attempts at solving Go involve **Monte Carlo tree search (MCTS)**. Monte Carlo rollouts are used to narrow the search to only promising actions, and to sample actions during rollouts. This approach has achieved strong amateur play.

AlphaGo achieved better position evaluation using **deep convolutional neural networks**, which evaluated positions using a value network, and sampled actions using a policy network.

A quick position evaluation overview:

1. A **supervised learning (SL)** policy network $\mathbf{p_\sigma}$ is trained directly from expert human moves.
2. A **fast policy** $\mathbf{p_\pi}$ that can rapidly sample actions during rollouts is used with **supervised learning (SL)** to predict human expert moves.
3. A **reinforcement learning (RL)** policy network $\mathbf{p_\rho}$ is then improved by policy gradient learning. A new dataset is generated by self-play with the RL policy network.
4. A **value network** $\mathbf{v_\theta}$ is trained by **regression** to predict the expected outcome in positions from the self-play data set.

The **supervised learning (SL)** network resulting from a 13- layer **deep convolutional neural networks** predicted expert moves on a held out test set with an accuracy of 57.0%. Small improvements in accuracy led to large improvements in playing strength, but larger networks which achieve better accuracy are slower to evaluate during search.

Reinforcement learning (RL) policy network is trained by playing games between the current RL policy network and a randomly selected previous iteration of the RL policy network. Randomizing from a pool of opponents stabilizes training by preventing overfitting to the current policy. When played head-to-head, the RL policy network won more than 80% of games against the SL policy network. Without any search, the RL policy network won 85% of games against Pachi, compared to 11% of games against Pachi by previous SL with convolutional networks.

Reinforcement learning of **value networks** for position evaluation is done by regression of on state-outcome pairs, with stochastic gradient descent to minimize the mean squared error (MSE) between the predicted value $\mathbf{v_\theta(s)}$, and the corresponding outcome. New self-play data set where each game is played between RL policy networks to game termination is generated to minimise overfitting issues. The position evaluation accuracy of the RL value network, compared to Monte Carlo rollouts using the fast rollout policy, is consistently more accurate.

AlphaGo combines the policy and value networks in an **MCTS algorithm** that selects actions by lookahead search. Each edge (s, a) of the search tree stores an **action value Q(s, a)**, **visit count N(s,**

a), and **prior probability P(s, a)**. The tree is traversed by **simulation**, starting from the root state. At each time step t of each simulation, an action a_t is selected from state s_t so as to maximize action value plus a bonus(s,a) $\propto P(s,a)/ (1+N(s,a))$ that is proportional to the prior probability but decays with repeated visits to encourage exploration. When the traversal reaches a leaf node s_L at step L, the leaf node may be expanded.

The leaf position s_L is processed just once by the **SL policy network**. The output probabilities are stored as prior probabilities P for each legal action a, $P(s,a)= P_\sigma(a|s)$. The leaf node is evaluated both by the **value network** $v_\theta(s_L)$; and the outcome of a random rollout played out until terminal step T using the **fast rollout policy**. Both evaluations are combined. At the end of simulation, the action values and visit counts of all traversed edges are updated. Each edge accumulates the visit count and mean evaluation of all simulations passing through that edge. Once the search is complete, the algorithm chooses the most visited move from the root position.

Interestingly, the SL policy network outperformed the stronger RL policy network in AlphaGo, even though the value function derived from the stronger RL policy network performed better than that derived from the SL policy network in AlphaGo.

Even without rollouts, AlphaGo exceeded the performance of all other Go programs. Nevertheless, the mixed evaluation performed best. The two position-evaluation mechanisms seem complementary: the **value network** approximates the outcome of games played by the strong but impractically slow **RL policy**, while the **rollouts** can precisely score and evaluate the outcome of games played by the weaker but faster rollout policy.

AlphaGo now plays at the level of the strongest human players. An effective **move selection** and **position evaluation** functions for Go have been developed based on **deep neural networks** that are trained by a novel combination of **supervised** and **reinforcement learning**, resulting in a new search algorithm that successfully combines **neural network evaluations** with **Monte Carlo rollouts**. AlphaGo integrates these components together, at scale, in a high-performance tree search engine. AlphaGo evaluated far fewer positions; by selecting those positions more intelligently, using the **policy network**, and evaluating them more precisely, using the **value network**—an approach that is perhaps closer to how humans play.