One-Page AlphaGo

Fei Xia

This review summarizes the breakthrough of game Go [1] in 2016.

Solving a game of perfect information can usually be boiled down to search the optimal value in a tree containing b^d possible paths, where in chess $b \approx 35, d \approx 80$ and in Go $b \approx 250, d \approx 150$. Obviously, tackling problem of Go is infeasible by using exhaustive search or simple heuristics. However, there are some effective strategies we can apply: (1) Reduce breadth of search by sampling actions from policy $p(a|s)$ (2) Reduce depth of search by position evaluation (3) Combine policy and value with Monte Carlo tree search (MCTS). The general steps are:

- Train a supervised learning (SL) policy network p_σ directly from human moves using a 13-layer convolutional neural networks. The input are $48 \times 19 \times 19$ images (for example, its component stone color is $3 \times 19 \times 19$), the output is probability over all legal moves by using a softmax layer. Accuracy is 55.7%.

- Train a fast policy p_π that can rapidly sample actions during rollouts. This uses a **linear** softmax of small pattern features. Accuracy is 24.2%, but it takes $2\mu s$ to select a move, compared to $3ms$ in p_σ.

- Train a reinforcement learning (RL) policy network p_ρ to improve the supervised policy network by optimizing for outcome of the game. This adjusts the policy network towards winning the game instead of maximizing the prediction accuracy. p_ρ essentially has the same structure with p_σ and its weights are initialized with same values $\rho = \sigma$. The game is played between current policy network p_ρ and a randomly (to combat overfitting) selected previous iteration of policy network.

- Train a value network ν_θ to predict the winner of games played by reinforcement learned policy network against itself. This network has a similar architecture to the policy network, but with one more feature plane (current player color) and with the output changed to a single prediction (regression, mean squared error loss). Predicting game outcomes from complete games can easily lead to overfitting, since successive positions are strongly correlated, differing by just one stone. So newly generated self-play data between RL policy network is used. The data is sampled from separate games containing 30 million distinct positions.

- Combine policy network, value network and fast policy with MCTS. A standard MCTS procedure contains four steps: Selection, Expansion, Evaluation and Backup. To make it more easily to understand, we only roughly talk about how it selects state in simulation (refer the math in original paper for exact formulation if you are interested):

  state score = value network output + fast rollout policy outcome + SL policy network output

  High state score (or stone move) will be selected. Value network output and fast rollout policy outcome are evaluation functions and are evaluated at the leaf node (but note that to evaluate fast rollout, we need to go until terminal step); SL policy network output is a probability of action at current stage, acting as a bonus of selection score. The score will be decayed by number of visits to encourage exploration. Note that RL policy network is only used for helping to derive value network and not used directly in MCTS.

That's it, the AlphaGo program that beats human!

References

[1] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.