2017/4/8

Mastering the game of Go with deep neural networks and tree search

Author: XiJia

Goals

- Game-> Go
- Evaluating function-> Deep neural networks and Tree search
- Use ‘policy networks’ to select moves
- Use ‘value networks’ to help select moves which would win finally

Techniques

- Use ‘policy networks’ to select moves
    - supervised learning from human expert games

This is a simple method to learn from observing KGS competition, that means, let the computer study “ Fixed thoughts”, and it would know how to move in a given situation. However, these competitions are limited, so the second method uses.

Specifically, *the policy network is trained on randomly sampled state-action pairs (s, a), using stochastic gradient ascent to maximize the likelihood of the human move a selected in state s*

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma}$$

They trained a 13-layer policy network as SL policy network.

- reinforcement learning from games of self-play

This method is computer plays go with itself during dozens of competition, and judges which are better moves through finally results. Because of self-play, it can have numerous competition. And this is also a reinforcement learning using self-play.

They used the reward function to train the RL network, that is zero for all non-terminal time steps t < T. The outcome zt = ± r(sT) is the termi- nal reward at the end of the game from the perspective of the current player at time step t: +1 for winning and −1 for losing. Weights are then updated at each time step t by stochastic gradient ascent in the direction that maximizes expected outcome

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial\rho} z_t$$

-Use ‘value networks’ to evaluating board positions

This method uses for evaluating which is better in the overall game. And also this is trained by self-play because the limited data of human competition in Go would make failures in the deep learning process.

They train the weights of the value network by regression on state-outcome pairs (s, z), using stochastic gradient descent to minimize the mean squared error (MSE) between the predicted value vθ(s), and the corresponding outcome z

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

When playing chess, the above two methods cooperate with each other:

- This first method ‘policy networks’ would judge which are considerable moves in the current situation. These moves were searched by using Monte Carlo tree to know the better moves. Though the whole process of computation, the second method ‘value network’ would help the first method to cutting numerous insignificant moves in trees to surge the efficiency of whole computation.
- The second method can also give a suggestion in the forecast move. Therefore these two methods scores can be equally added to make the final decision.

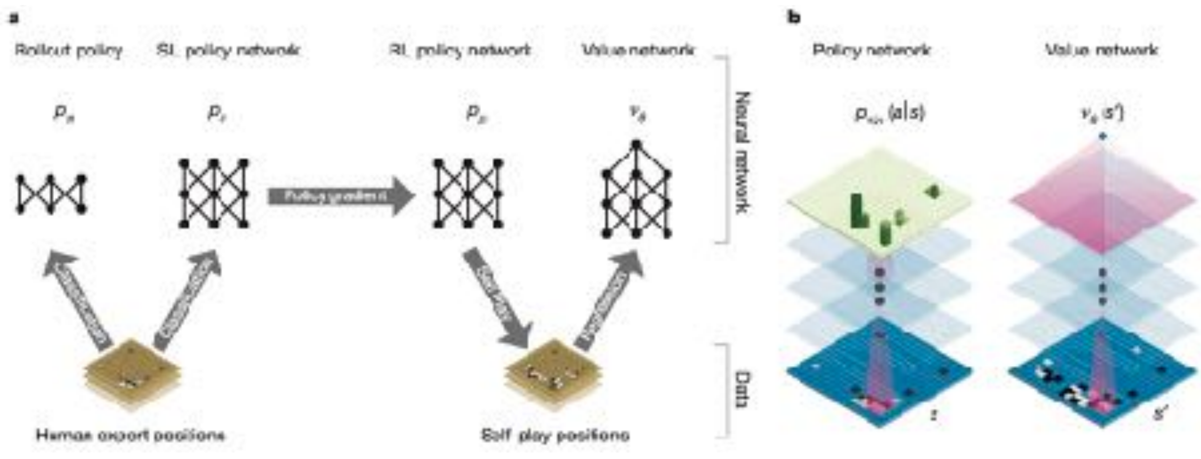

Figure 1. Neural network training pipeline and architecture

- Combine the policy and value networks in an MCTS algorithm

Monte Carlo rollouts search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p.

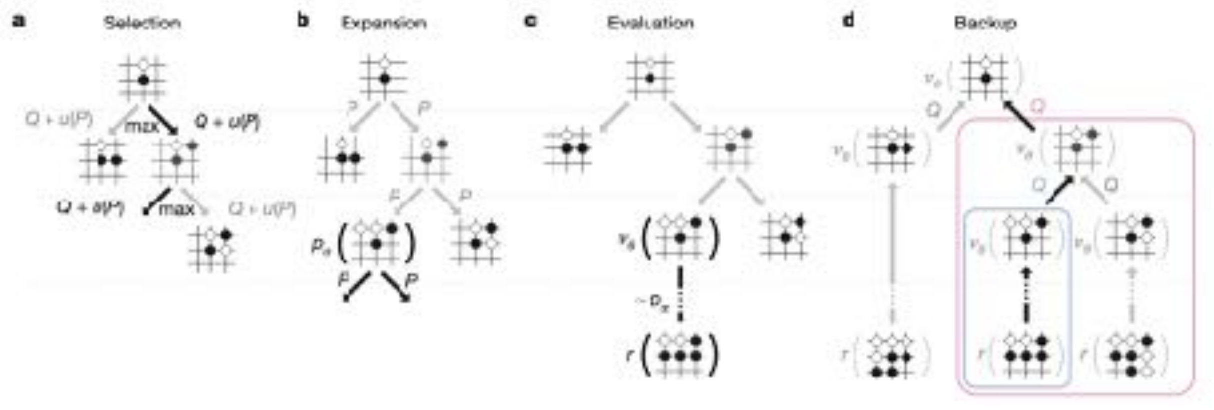

Figure 2. Monte Carlo tree search in AlphaGo

a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus u(P) that depends on a stored prior probability P for that edge.

b, The leaf node may be expanded; the new node is processed once by the policy network pσ and the output probabilities are stored as prior probabilities P for each action.

c, At the end of a simulation, the leaf node is evaluated in two ways: using the value network vθ; and by running

a rollout to the end of the game with the fast rollout policy pπ, then computing the winner with function r.

d, Action values Q are updated to track the mean value of all evaluations r(·) and vθ(·) in the subtree below that action.

Result

- *SL policy networks predicted expert moves on a held out test set with an accuracy of 57.0% using all input features, and 55.7% using only raw board posi- tion and move history as inputs, compared to the state-of-the-art from other research groups of 44.4% at date of submission.*
- The results with equal additions by two methods are much better than they evaluate by themselves respectively.
- *When played head-to-head, the RL policy network won more than 80% of games against the SL policy network. We also tested against the strongest open-source Go program, Pachi14, a sophisticated Monte Carlo search program, ranked at 2 amateur dan on KGS, that executes 100,000 simulations per move. Using no search at all, the RL policy network won 85% of games against Pachi. But only using the SL network, it*

just won 11% of games against Pachi23 and 12% against a slightly weaker program, Fuego.

- AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0.

The figures and formulas and some paragraph come from the article Mastering the game of Go with deep neural networks and tree search