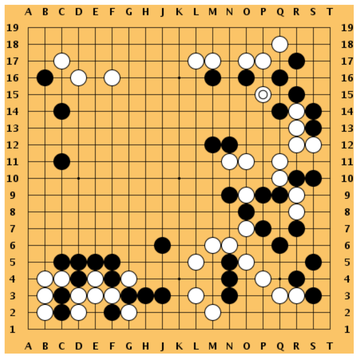
Learn to play Go

Albert Liu (albertpl@stanford.edu)

Motivation & Challenges

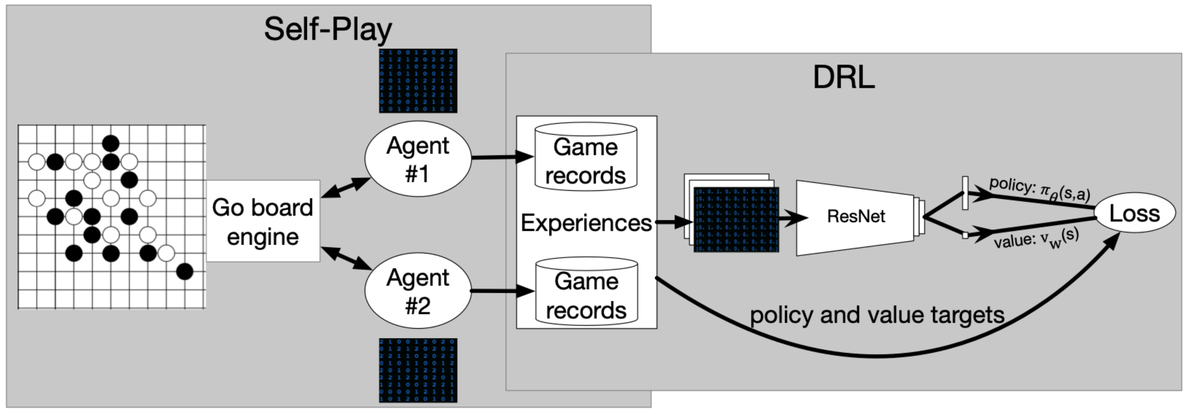


- simple rules but mastering requires many years of study by human
- considered hardest classic board game and grand challenge for AI
- AlphaGo and AlphaGoZero have rocked the Go and AI world
- Challenges
  - state based search is intractable due enormous search space ($\sim 10^{170}$), large number of legal move per state(~ 250)
  - massive computational requirements for MCTS and training DNN
  - lack of domain expertise

Problem Definition

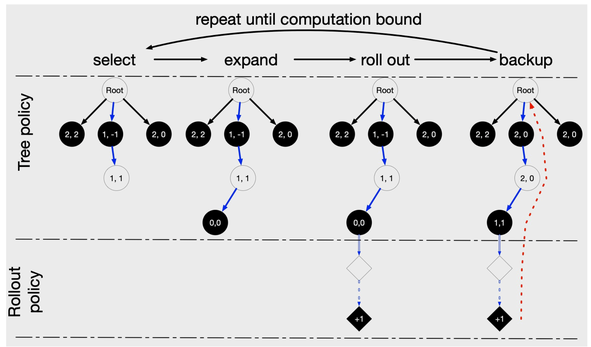
- Pachi built-in *UCT* engine as our oracle which is said to achieve highest amateur expect level (KGS 7 dan) on 9×9 board.
- To learn playing Go without incorporating lots of heuristics and predefined patterns beyond basic Go rules
- To understand pro and con of various approaches.

System architecture



- $\pi_\theta(A_t|S_t)$ A probability vector for taking each action A_t for S_t.
- $v_w(S_t) \in [-1, +1]$

Monte Carlo Tree Search

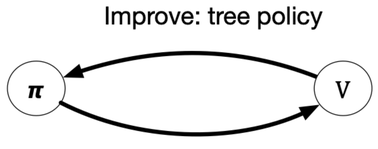


- Tree policy: UCB1

$$a \leftarrow \arg\max_a q(s,a) + c\sqrt{\frac{\log \sum_{a'} n(s,a')}{n(s,a)}}$$

Select best within tree.

- Rollout policy: random
  Sampling breaks curse of dimensionality



Deep Reinforcement Learning

- REINFORCE with Baseline
  $\pi_\theta(s,a)$ to approximate police. $v_w(s)$ as baseline to reduce variance.

$$\theta_{t+1} \leftarrow \theta_t + \alpha(r_t - v_w(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t)$$
$$w_{t+1} \leftarrow w_t + \alpha(r - v_w(s_t))\nabla_w v_w(s_t)$$

- Actor-critic with Baseline
  TD error as approximation of advantage.

$$\theta_{t+1} \leftarrow \theta_t + \alpha(r + v_w(s_{t+1}) - v_w(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t)$$

- combining DNN and MCTS
  - $\pi_\theta(s,a)$ as priors for expansion
  - $v_w(s)$ as estimated value, no rollout
  - supervised training, i.e. as close as possible to statistics from tree search.

$$a \leftarrow \arg\max_a q(s,a) + c\pi_\theta(s,a)\frac{\sqrt{\sum_{a'} n(s,a')}}{n(s,a)+1}$$

$$\mathrm{l}(\theta, w) = \sum_i (v_w(s_t) - r_t)^2 - \frac{\sum_a n(s_t,a) \log \pi_\theta(s_t,a)}{\sum_{a'} n(s_t,a')}$$

Preliminary Results

All results are averaged over 1000 games of 9×9 board.

- MCTS

	random	random	oracle
rollouts	100	1000	1000
win rate	0.92	0.99	0.16
time	2.4	29.3	35.1

- REINFORCE with baseline v.s. oracle

iteration	win rate
1	0.025
2	0.0265
3	0.0375
4	0.055

Analysis

- MCTS pachi uses **heavy** rollout policy, which entails rule based pattern, such as, if the last move has put its own group in *atari* we capture it; *Nakade*a move is played inside the eyeshape to prevent the opponent from splitting it into two eyes. Also Pachi applies heuristics based priors when expanding new node.
- Our MCTS uses random rollout policy and not as strong as it could be.
- Self-played based DRL approaches require