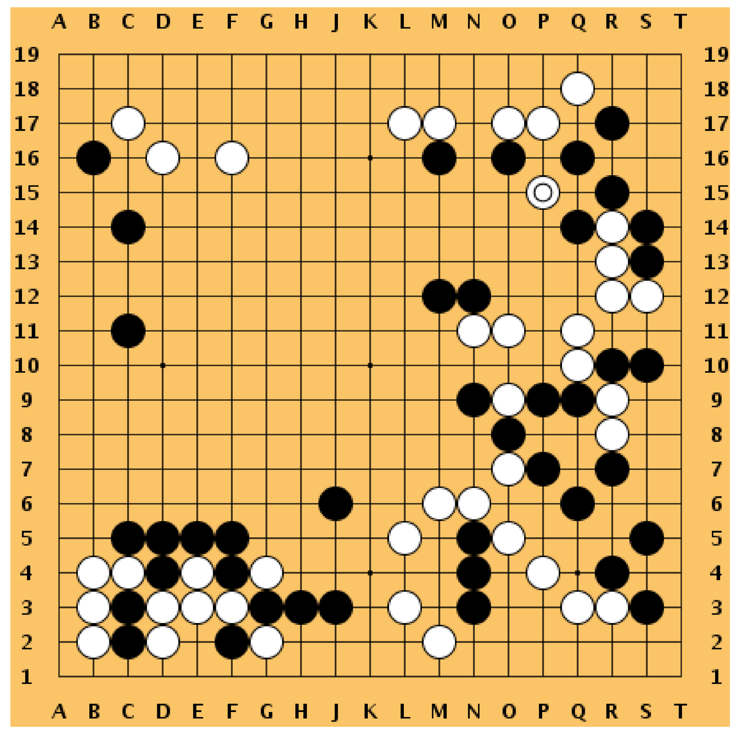
LEARN TO PLAY GO

Albert Liu (albertpl@stanford.edu)

MOTIVATION & CHALLENGES

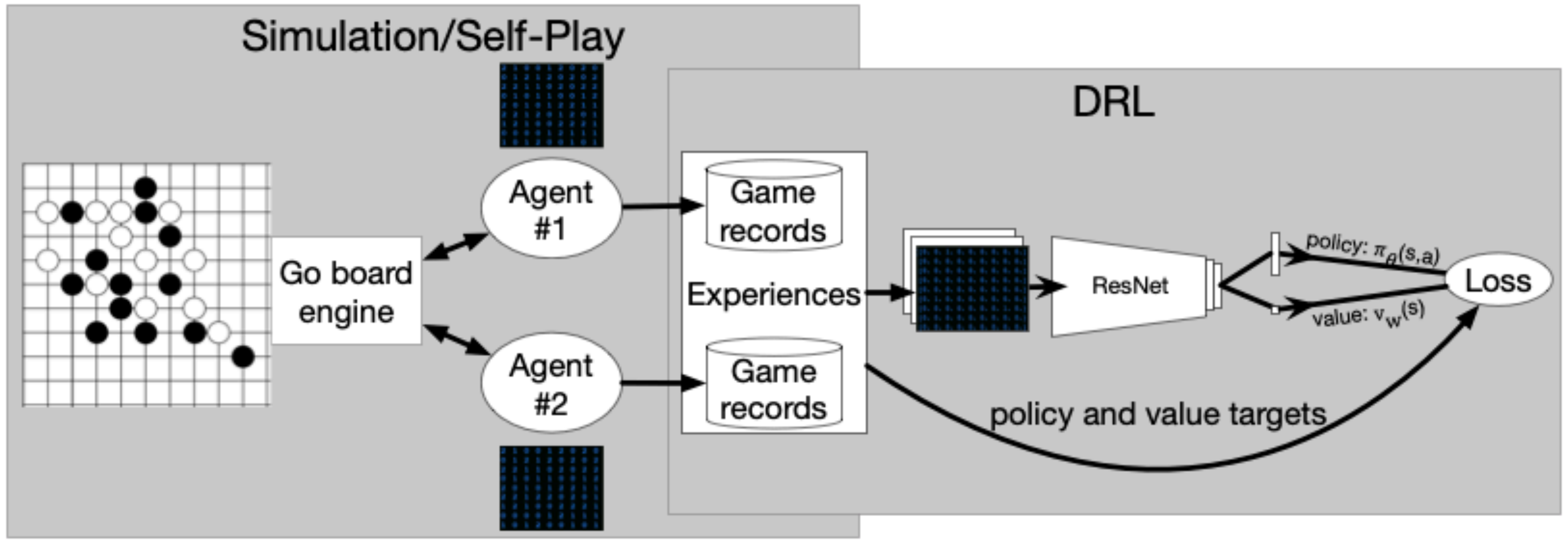


- Simple rules but it takes many years of study to master
- Considered hardest classic board game and grand challenge for AI
- AlphaGo and AlphaGoZero have rocked the Go and AI world
- Challenges
  - Search is intractable due to enormous search space ($\sim 10^{170}$) and large number of legal move per state(~ 250)
  - Massive computational requirements for MCTS and training DNN
  - Lack of domain expertise

PROBLEM DEFINITION

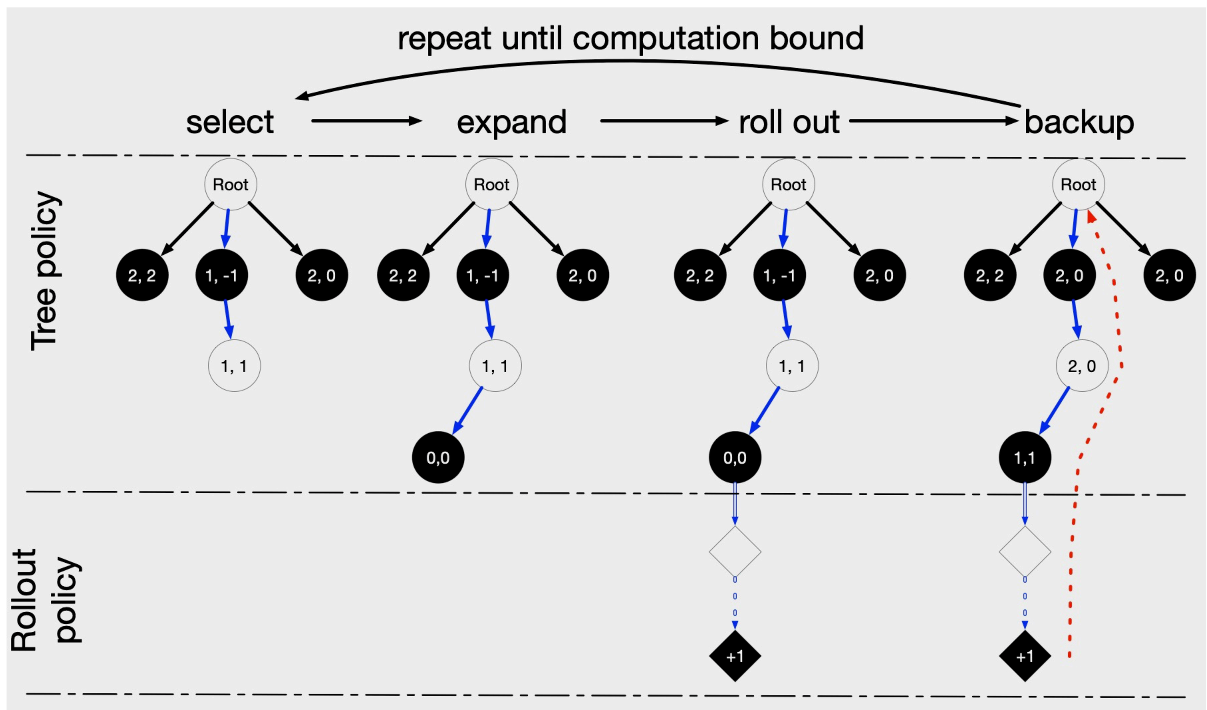
- Formulate Go as a turn-taking, two player, zero-sum game of perfect information. The input for each agent is current board positions and histories. The output (action space) is any legal position and a pass action.
- The goal is to have agent learn to play Go without incorporating lots of heuristics and predefined patterns beyond basic Go rules
- Pachi built-in *UCT* engine as our oracle which is said to achieve highest amateur expect level (KGS 7 dan) on 9×9 board.

SYSTEM ARCHITECTURE



- $S_t \in \mathbb{R}^{9\times 9\times 17}$
- $\pi_\theta(s,a) \sim \mathbb{P}(A_t = a | S_t = s)$
- $v_w(s) \sim V_\pi(S_t = s)$

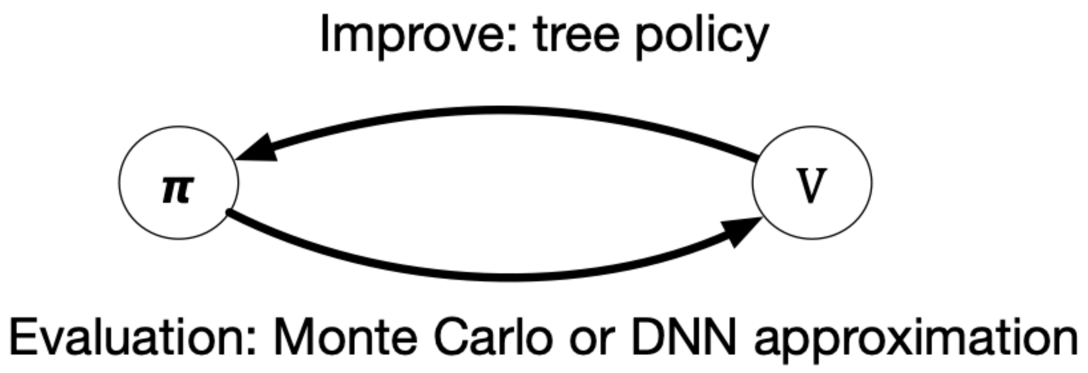
MONTE CARLO TREE SEARCH



- Tree policy(UCB1): select next within tree

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

- Rollout policy: random sampling
- After simulations, select based on visit counts



DEEP REINFORCEMENT LEARNING

- REINFORCE with Baseline
  $\pi_\theta(s,a)$ as approximation of 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.

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

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

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

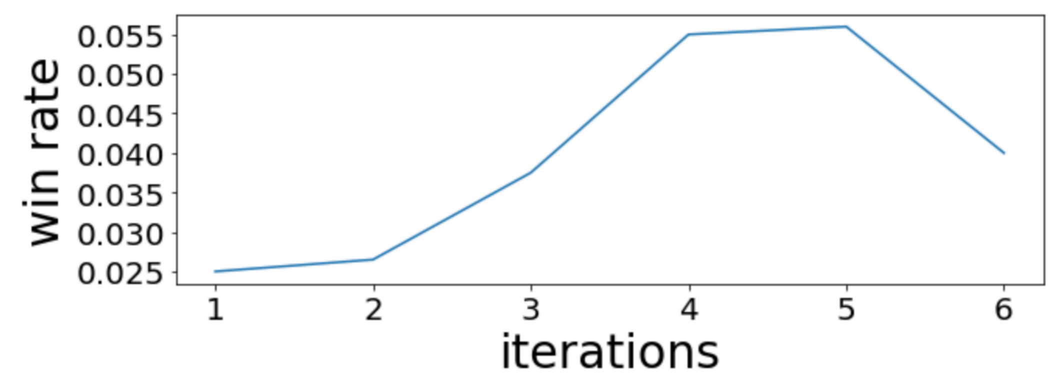
$$l(\theta,w) = \sum_i (v_w(s_t) - r_t)^2 - \frac{\sum_a n(s_t,a)\log \pi_\theta(s_t,a)}{\sum_{a'} n(s_t,a')}$$

PRELIMINARY RESULTS

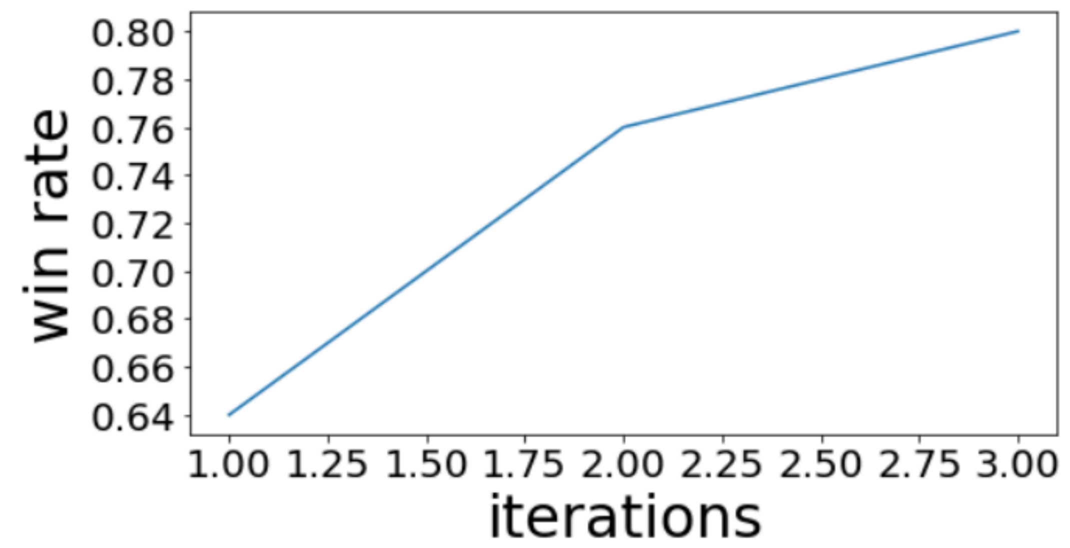
- MCTS, 1000 games, komi=0.0

	baseline	baseline	oracle
# simulations	100	1000	1000
win rate	0.92	0.99	0.16
time per game	2.4s	29.3s	35.1s

- REINFORCE with Baseline vs oracle
  2000 games, 10 epochs/iteration, komi=0.5



- Combined v.s. baseline
  100 games, 10 epochs/iteration, 500 simulations, komi=0.5



- All results are on 9×9 board

ANALYSIS

- MCTS pachi uses **heavy** rollout policy, which entails rule based pattern, such as *atari* and *Nakade*. Also Pachi applies heuristics based priors when expanding new node.
- Our MCTS uses **light** rollout policy and not as strong as it could be.
- For MCTS, the number of simulations per search is important for accurate policy evaluation. We manage to run ~ 0.5ms per simulation.
- Policy gradient method may converge to local optimal where π_θ almost becomes deterministic and therefore we don't explore other actions. This is particularly bad for the starting moves.
- It is computationally expensive to gather experiences through self play. It takes > 3 hours to run 100 games with moderate 500 simulations per move.