Predictron

A recurrent value function

DQN with recurrent value function

- DQN might learn to “remember” not “plan”
- Because a fixed-depth neural network is not capable of representing an “algorithm”
- Recurrence relation e.g. Bellman equation might need something with varying depth
- RNN might allow a value function to mimic the Bellman equation itself!
- This could lead to much more data efficient

Silver, David, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, et al. 2016. “The Predictron: End-To-End Learning and Planning.” *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1612.08810.

Adaptive computation time for RNN

- Predictron needs to decide for itself “how long to run”

Graves, Alex. 2016. “Adaptive Computation Time for Recurrent Neural Networks.” *arXiv [cs.NE]*. arXiv. https://doi.org/10.475/123.

AlphaZero's MCTS

Konpat Preechakul
Chulalongkorn University
October 2019

Model is known, but adversary is not known

- Adversary is a “part” of environment
- Model is then not a true model
- In a specific setting: two-player zero sum game
- We could use **self-play + model = env**
- Assuming that my adversary is myself
- We can go very far with this …

Goal

- Improve the prior policy
- Give a better value target

Not from the environment, from the model

Bird-eye view

- Having a policy
- Plan (Tree search) for many steps
  - Having a better policy
- Fit the current policy to the better policy
- Repeat

Better policy

- Search the tree using some **heuristic**
- Better paths are given more importance over time
- Better policy = “the most traversed path”

Better value target

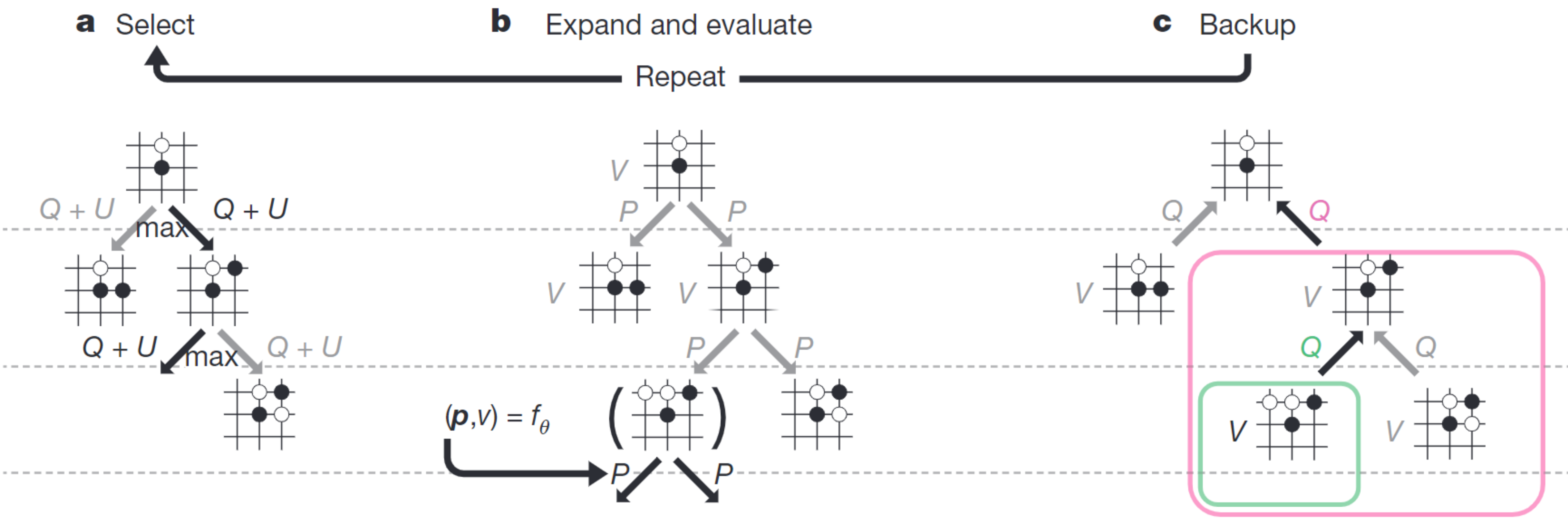
- Value target comes from the “real” value of those paths

Planning action by argmax U

$$U(s,a) = Q(s,a) + c_{puct} \cdot P(s,a) \cdot \frac{\sqrt{\Sigma_b N(s,b)}}{1+N(s,a)}$$

Observe that N changes overtime
Our search paths will change overtime
Balancing between prior belief Q and unknown exploration

Monte Carlo Tree Search



Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, et al. 2017. “Mastering the Game of Go without Human Knowledge.” *Nature* 550 (7676): 354–59.

An edge contains

- N = number of visits (edge)
- Q = best estimate of the return (from v)
- P = prior policy (from policy network)

State contains

- V = best known estimate of the return (-1, 1)
  - From value network
  - From environment (if it ends)

Progresses

- Q gets values from future V
  - Future V has lower **bias**
  - Hence we get better estimate of Q
  - Which could lead to better policy
- N represents the better policy
  - Using multiple trials to reduce the variance even more

```python
def search(s, game, nnet):
    if game.gameEnded(s): return -game.gameReward(s)

    if s not in visited:
        visited.add(s)
        P[s], v = nnet.predict(s)
        return -v

    max_u, best_a = -float("inf"), -1
    for a in game.getValidActions(s):
        u = Q[s][a] + c_puct*P[s][a]*sqrt(sum(N[s]))/(1+N[s][a])
        if u>max_u:
            max_u = u
            best_a = a
    a = best_a

    sp = game.nextState(s, a)
    v = search(sp, game, nnet)

    Q[s][a] = (N[s][a]*Q[s][a] + v)/(N[s][a]+1)
    N[s][a] += 1
    return -v
```

https://web.stanford.edu/~surag/posts/alphazero.html

Architecture consideration

- We want to use only “one” network for both us and our adversary
- We need a notion of “canonical” state
  - That doesn’t depend on the player
  - Instead of alternating between players
  - We use the same player with alternating state

Learning policy and value networks

- After taking so many actions for an episode
  - Each action is from MCTS
  - Hence the actions are already from a "better policy"
- We will get a reward (win, lose) at the end
- Data = list of (state, action, win/lose)
- Minimize for policy network

- Minimize for value network