Recitation 4

Monte Carlo

Monte Carlo (MC)

Update Rule:

$$V(S_t) \leftarrow \text{average}(Returns(S_t))$$

Incremental Update:

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

where return is the sum of discounted rewards:

$$\underline{G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T}$$

DP (Value/Policy Iteration):

- Iterate through all possibilities

$$\underline{\sum_{s',r}} \underline{p(s', r \mid s, a)}[r + \gamma \underline{V(s')}]$$

- assumes full knowledge of env
- One step bootstrap: biased estimate

Monte Carlo Learning:

+ Collect samples from episodes

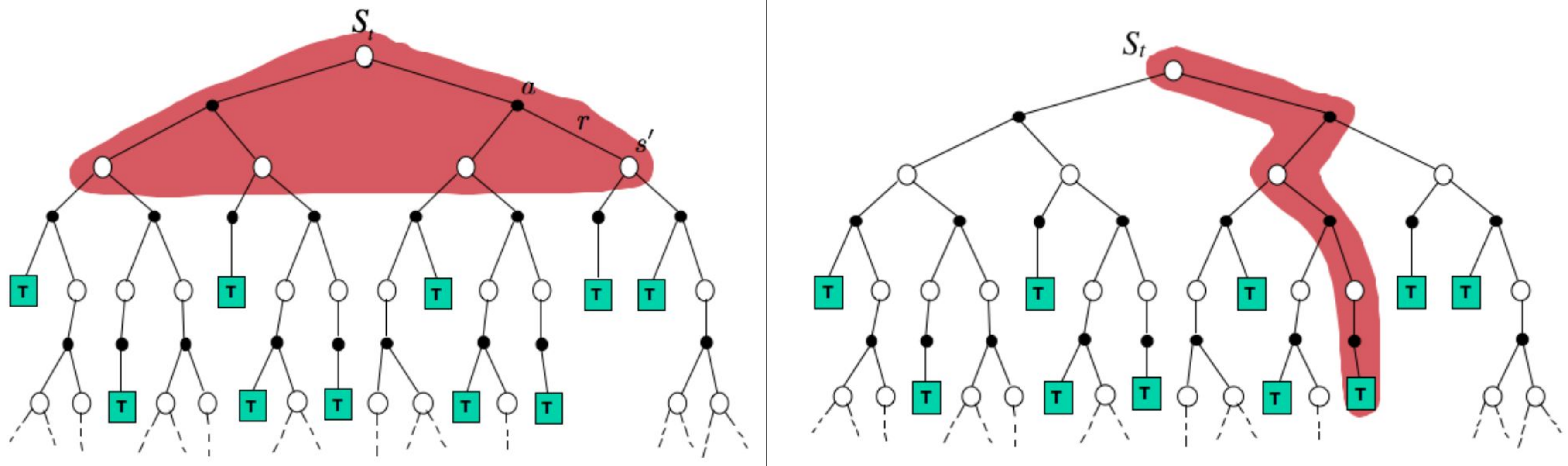
$$\pi\colon\ S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, \underline{R_T}$$

What does this assume?

- mean return: unbiased

Pro or con?

DP vs MC



Monte Carlo (MC)

Every-visit also exists… different convergence property

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$: Aggregate backwards

$G \leftarrow \gamma G + R_{t+1}$

$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

$Q(S_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$

How would you modify the above to also generate a policy?

Temporal Difference Learning

Temporal Difference Learning

New update rule:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{target: an estimate of the return}} - V(S_t)\Big]$$

- \+ Can learn before reaching a terminal state
- \+ Much more memory and computation-efficient than MC
- \- Using value in the target introduces bias

Motivation for TD learning and N-step returns

TD(0)

MC estimate

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right]
\end{aligned}
$$

Approximate with v

N-step returns

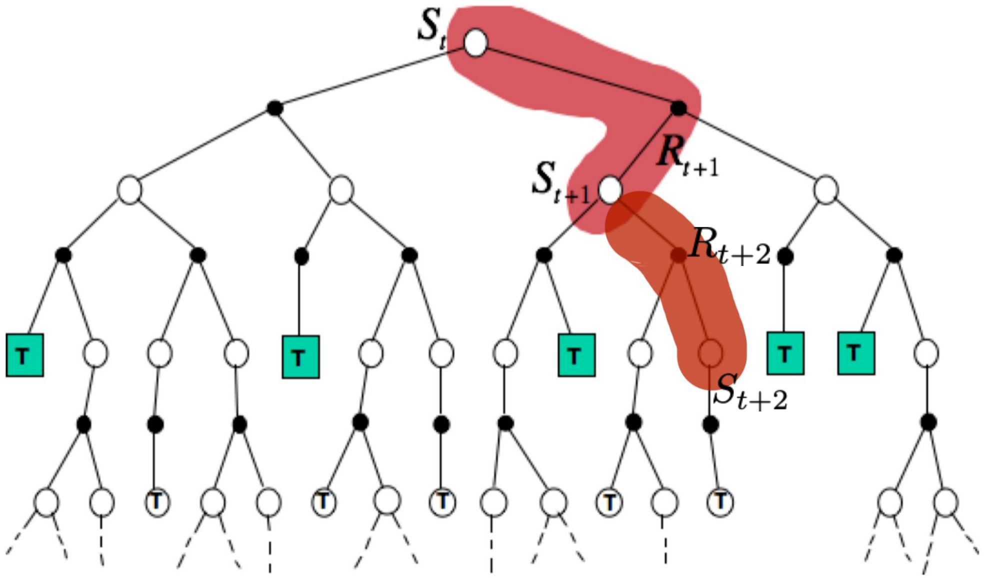
$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \\
&= \mathbb{E}_\pi\left[\sum_{i=0}^{N-1} \gamma^{i+1} R_{t+i} + \gamma^N \sum_{k=0}^{\infty} \gamma^k R_{t+k+N+1} | S_t = s\right] \\
&= \mathbb{E}_\pi\left[\sum_{i=0}^{N-1} \gamma^{i+1} R_{t+i} + \gamma^N v_\pi(S_{t+N}) | S_t = s\right]
\end{aligned}
$$

N-step returns

Less reliance on v

N-step returns

$$V(s_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) - V(S_t))$$



Q-learning: Off-policy TD Learning

1-step Q-learning update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

- **Key benefit: off-policy!**
- Only require state, action, reward, and next state drawn from the MDP
- Doesn’t depend on the policy anywhere!
- Is foundation for many sample-efficient RL methods

Deep Q-learning

- What happens if the state space and action space are too large?
  - Use function approximation to approximate the Q-values!
- Use gradient descent to take a step towards minimizing the Bellman error:

$$L = \left(\underbrace{\mathrm{sg}(R_{t+1} + \gamma \max_{A_t} q(S_{t+1}, A_{t+1}, w))}_{\text{Target value}} - \underbrace{q(S_t, A_t, w)}_{\text{Prediction}} \right)^2$$

Tabular

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{A_t} q(S_{t+1}, A_{t+1}) - q(S_t, A_t) \right]$$

Function Approximation

$$w \leftarrow w + \alpha \left[R_{t+1} + \gamma \max_{A_t} q(S_{t+1}, A_{t+1}, w) - q(S_t, A_t, w) \right] \nabla_w q(S_t, A_t, w)$$

Target Networks

$$L = \Big(\underbrace{\mathrm{sg}(R_{t+1} + \gamma \max_{A_t} q(S_{t+1}, A_{t+1}, w))}_{\text{Target value}} - \underbrace{q(S_t, A_t, w)}_{\text{Prediction}} \Big)^2$$

- One problem with deep Q-learning: nonstationary targets
  - Updating the network weights changes the target value, which requires more updates
  - Unintended generalization to other states S' can lead to error propagation
- Solution: calculate target values with a network that's updated every T gradient steps
  - Network has more time to fit targets accurately before they change
  - Slows down training, but not too many alternatives (recently: functional regularization)

Experience Replay

- Problem #1: neural networks undergo **catastrophic forgetting** if they haven't been trained on a (similar) sample recently
- Problem #2: online samples tend to be very correlated, which leads to unstable optimization
- Solution: keep large history of transitions in a "replay buffer," then optimize the Bellman error wrt random minibatches

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

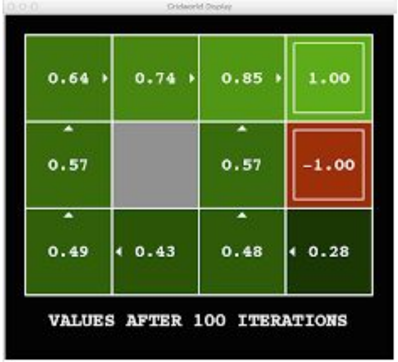
$$I = \left(r + \gamma \max_a Q\left(s', a', \mathbf{w}-\right) - Q(s, a, \mathbf{w}) \right)^2$$

Prioritized Experience Replay

Monte Carlo Tree Search

Problem: Large State-Action Space

Trying to estimate the value at every state (solving the full MDP) is often infeasible



MC and TD still try to estimate Q/V value function for every possible state or state-action

- Too much memory for tabular (10^48 states for chess)
- NN may be undefined at unseen states, "similar" states may have completely different values and optimal paths

Online Planning

- Use internal model to simulate trajectories at current state, find the best one

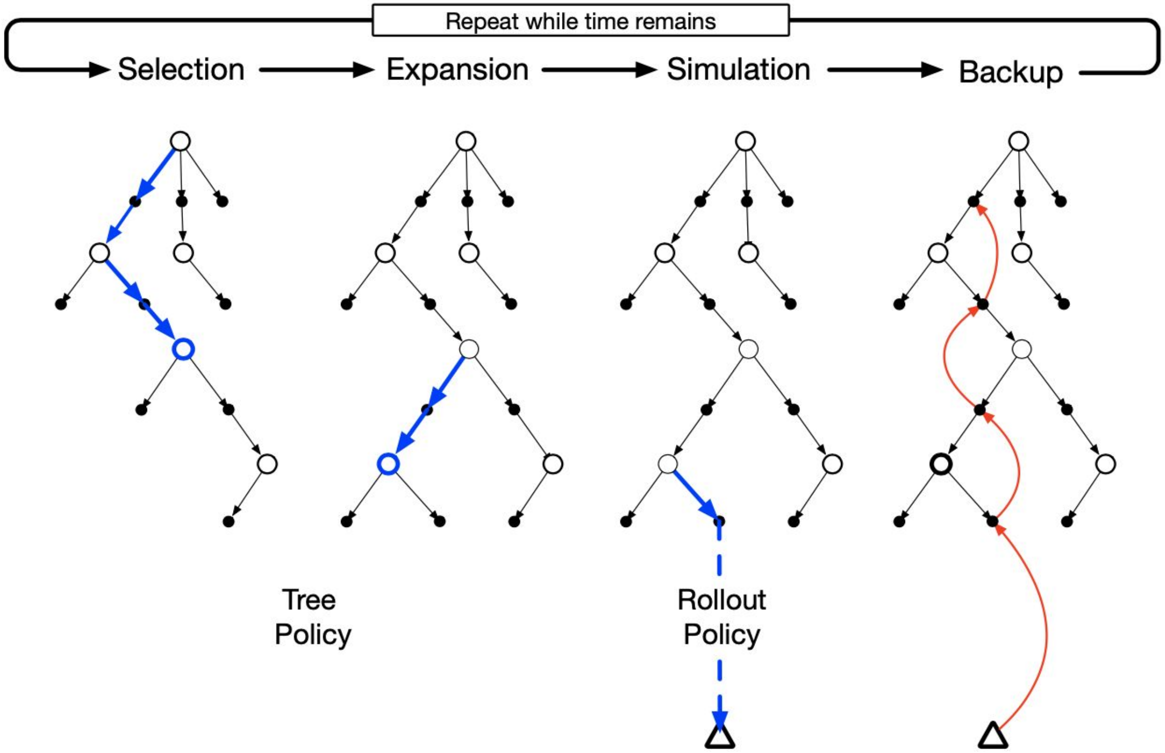
Monte Carlo Tree Search (MCTS):

- Only estimate value function for relevant part of state space
- Consider only part of the full MDP at a given step

MCTS

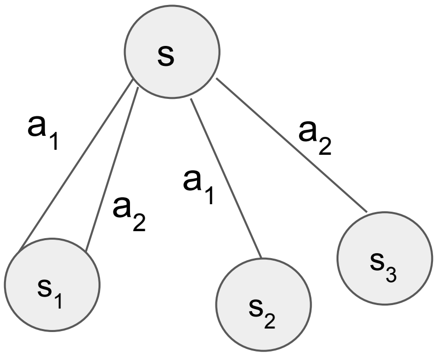
node = state

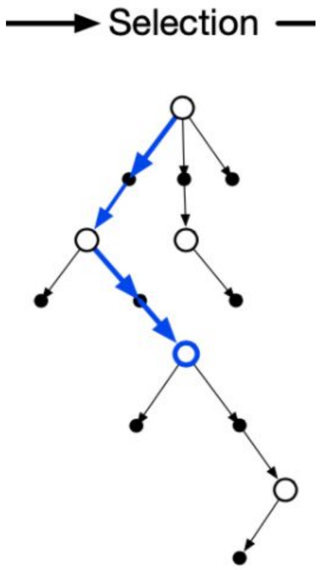
edge = action



- Tree: Stores Q-values for only a subset of all state-actions
- MC method: require episode termination to update values

Selection





Given:

- current state of agent = root node
- Empty or existing tree with Q-values

Steps:

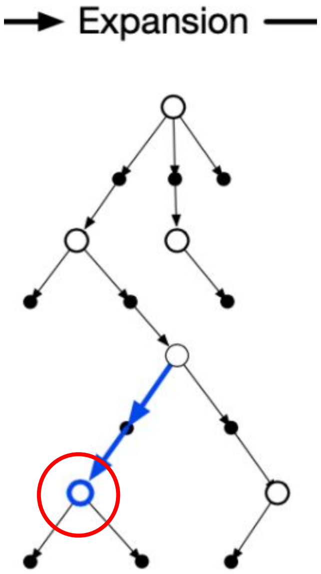
"children" = actions, don't know all possible (s,a) → s' transitions

```
function MCTS_sample(node)
    if all children expanded:#selection
        next = UCB_sample(node)
        outcome = MCTS_sample(next)
```

$$A_t = \operatorname{argmax}_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

- keep executing UCB repeatedly until you reach frontier of tree (state that is not a node)

Expansion



Given:

- at a new state ***s*** not part of the tree

Steps:

- Based on some rule, possibly add this new state to the tree
    - ex: if depth of this state < max depth
- Take random action ***a*** (since no Q-values available), receive reward ***r*** if available
- **G** = Simulation(***s***, ***a***)
- Store Q(***s*, *a***) = gamma***G** + ***r***
- return gamma***G** + ***r*** to propagate return to parent node

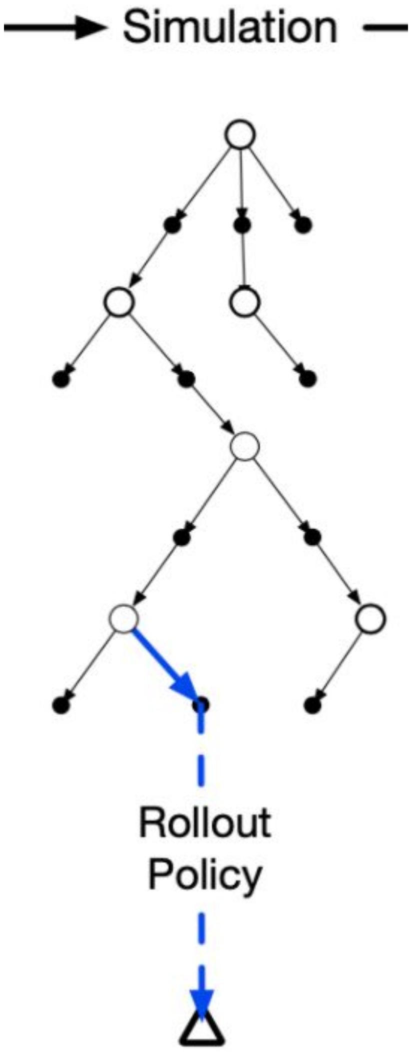
Simulation

Given:

- at a new state ***s*** not part of the tree

Steps:

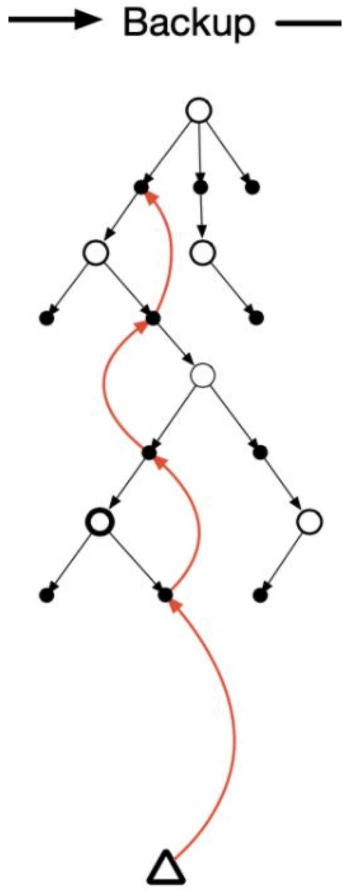
- If at terminal state, return reward
- use very fast policy to determine action ***a*** to take
    - ex: random policy
- **G** = Simulation(***s***, ***a***)
- return gamma***G** + ***r***   (Do Not store Q-value)



Backup

- Propagate return from the recursive calls
- Calculate return at each state

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$



MCTS Overall

- For the current state of agent, repeatedly perform the previous steps until some criteria
    - ex: time limit
    - ex: Q-value convergence within some threshold
- Execute the best action
- Reuse the subtree of the successor state and repeat!