Chapter 5: Monte Carlo Methods

- Dynamic programming requires complete knowledge of the environment, that is, complete knowledge of $p(s',r|s,a)$.
- RL doesn't need prior knowledge of $p(s',r|s,a)$. Instead it just needs:
  - to be able to observe the state s of the environment
  - to know what actions are available from any state
  - to receive a numerical reward (possibly stochastic) after taking an action
- Two major classes of methods:
  - Monte Carlo methods
  - Temporal-Difference methods

Monte Carlo Methods

- MC methods are ways of solving the RL problem based on averaging sample returns. (Recall R_t is the reward and G_t is the return.)
- Focus on episodic MDPs: under every policy the episode terminates in a finite number of time steps T, which may be random.

$$G_0 := R_1 + R_2 + \ldots + R_T$$

- Only after the termination of episode is value function estimate and the policy changed.
- Repeat:
  - With current policy π, run some episodes
  - Update the value function and policy π
- Where does the term “Monte Carlo” come from?

Monte Carlo Methods and Bandits

- Recall that in multi-arm bandit problem, the return is just a single reward.
  - We averaged the returns of each bandit to estimate q(a), the expected return of bandit a.
  - Thus the bandit algorithms (ε-greedy, UCB) were MC methods

- Now we are going to average the returns over multiple episodes.
  - Within each episode, the state changes and there are multiple rewards.

- Can convert the MDP problem into k-arm bandit problem:
  - Consider each deterministic policy as a bandit. Use ε-greedy over the policies.
  - What is the problem with this approach?

Monte Carlo Evaluation: Basic Algorithm

For fixed policy π, how can we estimate $v_\pi(s_0)$ for fixed s_0?

Basic Algorithm:

- Input policy π to be evaluated
- Initialize estimate $V(s_0) = 0$ and Returns(s_0) ← empty list
- Repeat
  - Run episode with π beginning in state s_0: $S_0 = s_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  - Calculate return $G := R_1 + R_2 + \ldots + R_T$
  - Append G to Returns(s_0)
  - Set $V(s_0)$ = average (Returns(s_0))

- $V(s_0)$ converges to $v_\pi(s_0)$ as the number of episodes → ∞. Why?
- Using this approach, how would we determine $v_\pi(s)$ for all s?

How can we improve the basic algorithm?

- When we run an episode, we are implicitly obtaining returns for the states we pass through:
  - Suppose episode is S_0=7, A_0 =1, R_1=2, S_1=12, A_1=1, R_2= 5, S_2=20, A_2=0, R_3=20
  - Gives returns G(state 20) = 20, G(state 12) = 25, G(state 7) = 27
  - So can use this to improve our estimates not only for $v_\pi(7)$ but also for $v_\pi(12)$ and $v_\pi(20)$
- Obtain one return for each state visited in episode
- Note that G(state 12) = R_2 + G(state 20), G(state 7) = R_1 + G(state 12)
- So we can calculate the implicit returns by working backwards through the episode: start at state S_{T-1} and work backwards to S_0.

Improved Algo: Every visit Monte Carlo Evaluation

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, $t = T-1, T-2, \ldots, 0$:

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

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

Append G to $Returns(S_t)$

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

Example: Two episodes, $\mathcal{S} = \{1,2,3,\ldots,21\}$

- $S_0 = 7$, $A_0 = 1$, $R_1 = 2$, $S_1 = 12$, $A_1 = 1$, $R_2 = 5$, $S_2 = 20$, $A_2 = 0$, $R_3 = 20$
- $S_0 = 12$, $A_0 = 1$, $R_1 = 3$, $S_1 = 19$, $A_1 = 1$, $R_2 = 6$

- Quiz: What would be V(s), $s \in \mathcal{S}$, after the first episode? After the second episode? Assume V(s) is initialized to 0 for all $s \in \mathcal{S}$, and assume no discounting.

First-Visit Monte Carlo Evaluation

Input: a policy π to be evaluated

Initialize:

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

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

Loop forever (for each episode):

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

$\quad G \leftarrow 0$

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

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

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

$\quad\quad\quad$ Append G to $Returns(S_t)$

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

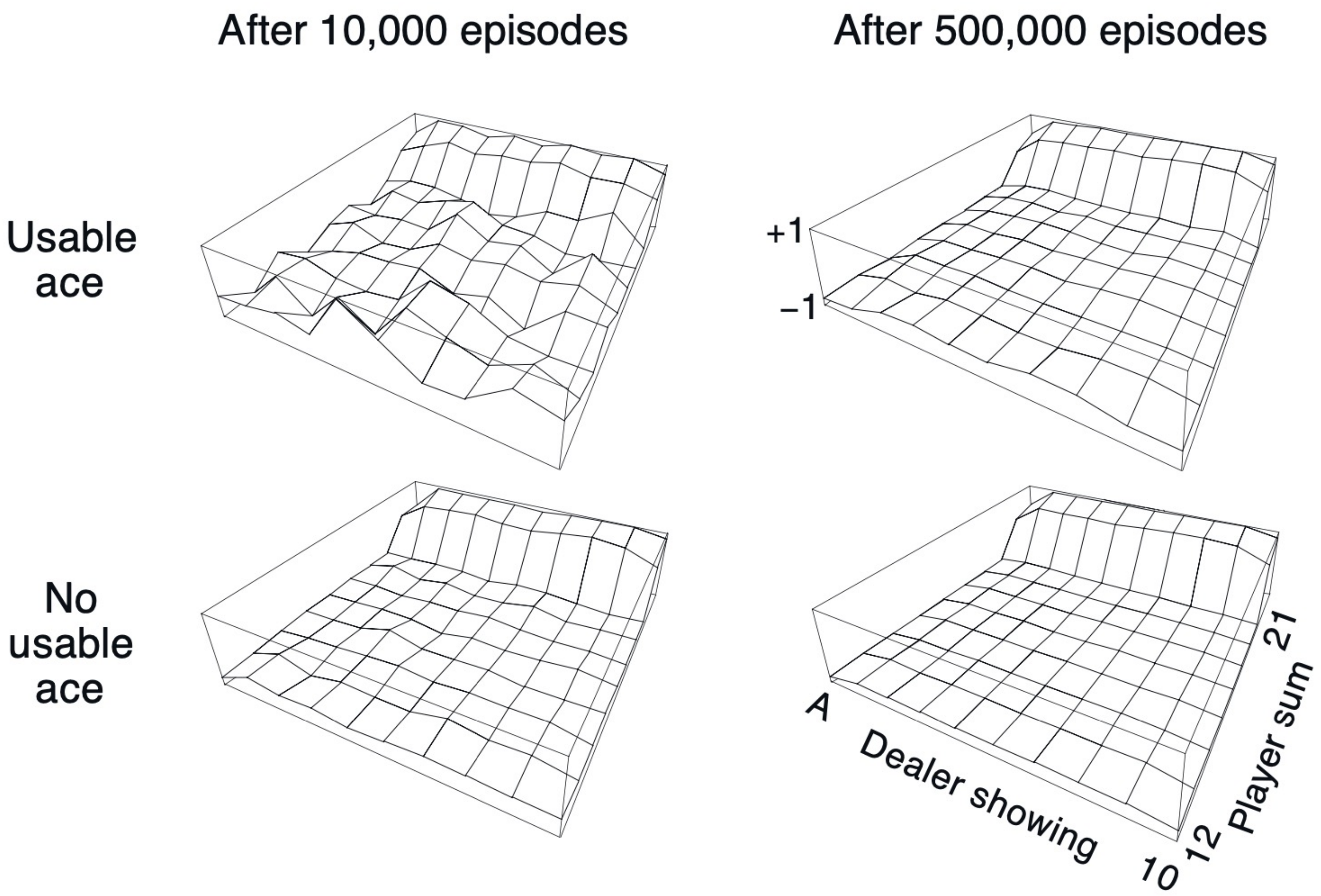
First visit versus every visit

- If the episode passes through the same state s twice, then can obtain two returns for that state, and average them both in for the estimate V(s).
- But these two returns are not independent. So can no longer use law of large numbers to analyze convergence.
- So for mathematical convenience, authors have line: Unless S_t appears in S_0, S_1,…,S_{t-1} (first-visit version of algorithm)
- With every-visit, we simply remove that line. Convergence proof becomes more challenging but is doable.

Real Blackjack

- Ace can count as 1 or 11
- Player will automatically hit if its sum is 11 or less. So player's total will range from 12 to 21 during game play.
- Dealer sticks if and only if sum of cards is 17 or more
- Player has "usable ace" if ace can count as 11 without going bust. Player may not hit with 10+3 but would likely hit with A+2
- State: current sum (12-21), has usable ace or not, dealer's up card
- 200 states altogether (10x10x2)
- Action space = {stick, hit}

Approximate value function for the policy player only sticks at 20 and 21



Exercise 5.1 Consider the diagrams on the right in Figure 5.1. Why does the estimated value function jump up for the last two rows in the rear? Why does it drop off for the whole last row on the left? Why are the frontmost values higher in the upper diagrams than in the lower? □

Exercise 5.2 Suppose every-visit MC was used instead of first-visit MC on the blackjack task. Would you expect the results to be very different? Why or why not? □

How do we use MC methods to find optimal policy?

- Recall policy iteration algorithm:

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

- Suppose that our V(s) from Monte Carlo evaluation are exactly equal to $v_\pi(s)$. Then we could improve with

$$\pi'(s) \leftarrow \text{argmax}_a\ r(s,a) + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')$$

or with

$$\pi'(s) = \text{argmax}_a\ q_\pi(s,a)$$

- In the RL setting, we do not know r(s,a) ad p(s'|s,a). So we use the latter approach: $\pi'(s) = \text{argmax}_a\ q_\pi(s,a)$

Two issues with policy iteration when MC is used for evaluation

1. The Monte Carlo evaluation algorithm estimates $v_\pi(s)$ not $q_\pi(s,a)$, so we need to adapt it to estimating $q_\pi(s,a)$.
2. $V(s)$ is only an estimate of $v_\pi(s)$. So will use Generalized Policy Iteration

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\quad \pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

$\quad Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

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

Loop forever (for each episode):

$\quad$ Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

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

$\quad G \leftarrow 0$

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

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

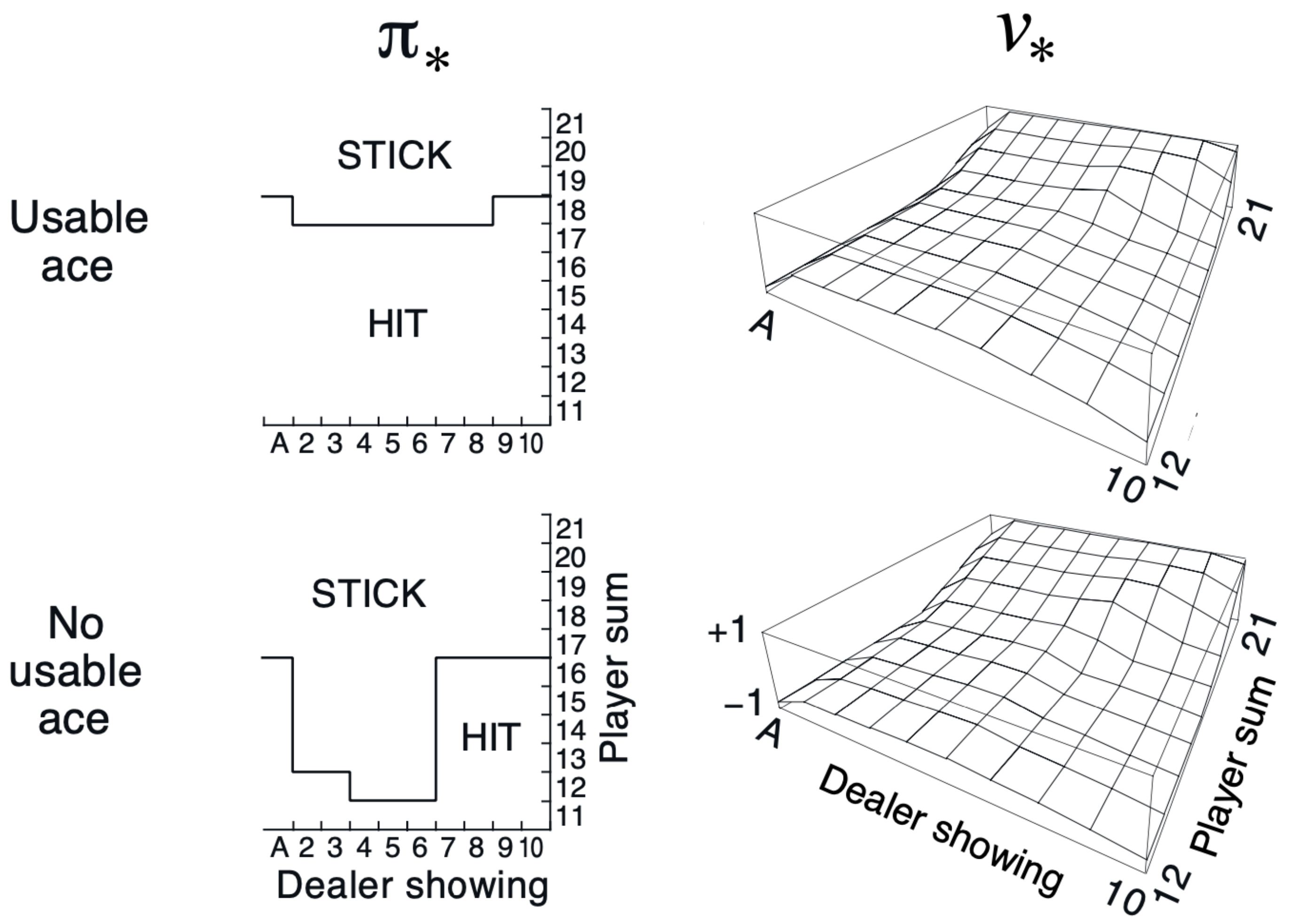
$\quad\quad$ ~~Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$~~:

$\quad\quad\quad$ Append G to $Returns(S_t, A_t)$

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

$\quad\quad\quad \pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

MC Exploring Starts applied to Blackjack



How would you program MC for blackjack?

- Function deal_card: takes no input and returns one of 13 different ranks, with each rank having the same probability.
- Have dictionary which maps card rank to values.
- Initialize Q(s,a) = 0 for all s,a. Initialize Returns(s,a). Initialize policy.
- For each episode:
    - Initialize episode by dealing one up card and one down card to dealer, and dealing cards to player until player's total is at least 12.
    - Determine initial state: (player's total, dealer's visible total, usable ace)
    - Note that initial state is random, and all initial states have positive probability. Choose first action randomly. Why?
    - Use current policy to play out episode according to rules of game.
        - First player draws cards according to current policy.
        - If player does not go bust, dealer then draws cards according to its fixed policy.
    - Determine return for episode.
    - Update Q(s,a) for all (s,a) visited in episode
    - Modify policy for all states visited in episode

Does MC exploring starts converge?

- “In our opinion, this is one of the fundamental most open problems in (tabular) reinforcement learning”.

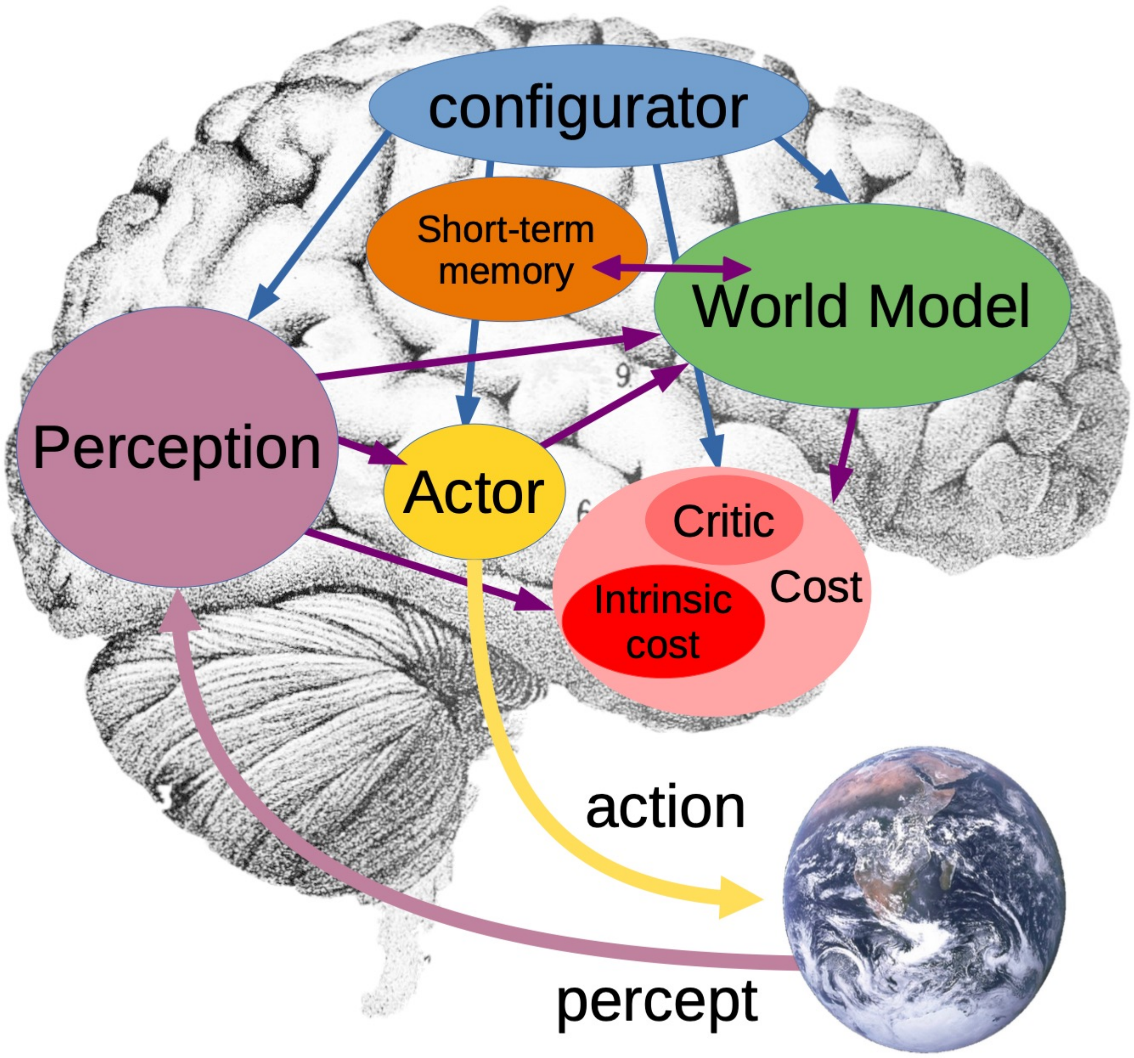
- Q(s,a) is only an estimate of $q_\pi(s,a)$. Policy improvement theorem requires exact value of $q_\pi(s,a)$

- See research paper by Wang et al.

Yann LeCun talk

- Goal is to build human level intelligence
- LLM’s are insufficient because (i) trained with text only; (ii) do not plan
- RL requires way to much environment interactions

Yann Lecun talk



World = environment
Perception = state (images,…)
Actor = policy
Critic = value estimate
World model = model for planning
Configurator = reward function
Short-term memory = buffer

Model-Based RL

- Goal: try to minimize interaction with environment
- Create approximate model of environment in computer. Use the model to simulate (partial episodes) with different sequences of actions.
- Choose the best first action and take that action in the actual environment. Move to new state. Repeat process.