

Monte- Carlo Methods

Objectives

- Understand how Monte-Carlo methods can be used to estimate value functions from sampled interaction
- Identify problems that can be solved using Monte-Carlo methods
- Use Monte-Carlo prediction to estimate the value function for a given policy.

Monte Carlo

- Monte Carlo methods are a class of computational algorithms that rely on random sampling to obtain numerical results.
- Monte Carlo methods are used to estimate value functions or policies by sampling trajectories (or episodes) of interaction between an agent and an environment

Monte Carlo

- Monte Carlo methods estimate values by averaging over a large number of random samples
- Example: roll 12 dice a few times and see the result. In this case, the average is 41.57, fairly close to the true

Sum of Faces	Probability
12	?
13	?
14	?
...	
71	?
72	?

Samples

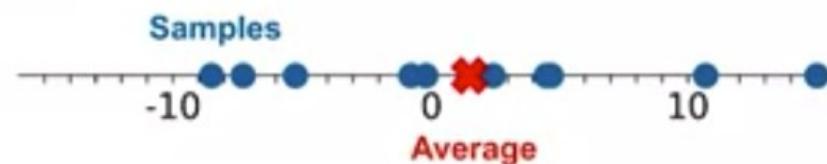
39 38 49 36 33 47 37 49 41 40 44 44
35 43 47 28 46 52 47 45 43 43 42 43
40 32 29 48 43 49 48 39 42 35 44 48
36 32 41 49 34 54 37 42 37 38 42 50
39 37 45 49 44 34 38 50 35 36 42 45

Monte Carlo

- Monte Carlo method for learning a value function would first observe multiple returns from the same state.
- It average those observed returns to estimate the exp

Recall that

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi} [G_t | S_t = s]$$

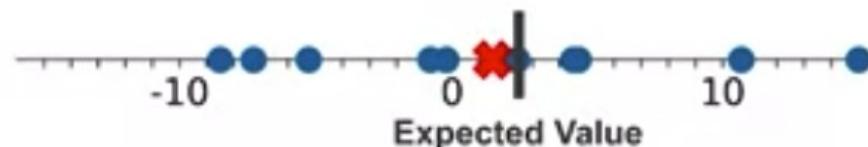


Monte Carlo

- The number of samples increases, the average tends to get closer and closer to the expected return.
- The more returns the agent observes from a state, the more likely it is that the sample average is close to

Recall that

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi} [G_t | S_t = s]$$

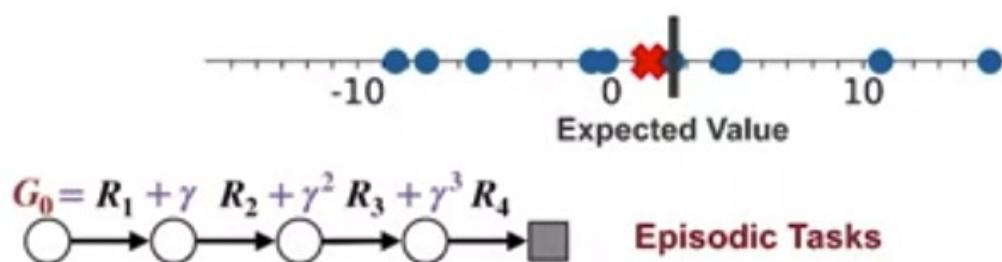


Monte Carlo

- These returns can only be observed at the end of an episode
- We will focus on Monte Carlo methods for episodic task

Recall that

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi} [G_t | S_t = s]$$



Monte Carlo

- In bandits the value of an arm is estimated using the average payoff sampled by pulling that arm.
- Monte Carlo methods consider policies instead of arms.
- The value state S under a given policy is estimated using the average that policy from S



Sample rewards

$$\frac{3 + 5 + 0 + 1}{4} = 1.5$$

Estimated value

Sample returns

$$\frac{3 + 5 + 0 + 1}{4} = 1.5$$

Estimated value

Monte-Carlo Methods

Monte Carlo

- Algorithm for estimating the state value function of a policy

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 : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

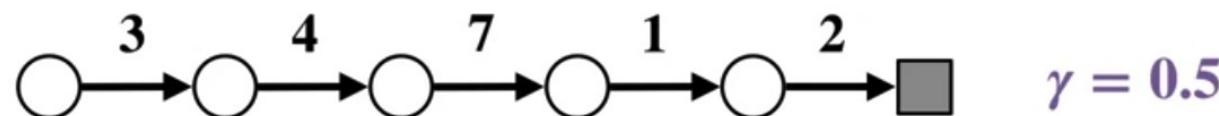
Append G to $Returns(S_t)$

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

Monte Carlo

- Suppose the discount factor is 0.05 and imagine an episode ending at time step five where the reward sequences is 3, 4, 7, 1, and two.
- Let's find each return G_0 to G_5 , starting by writing down the equation for each return.

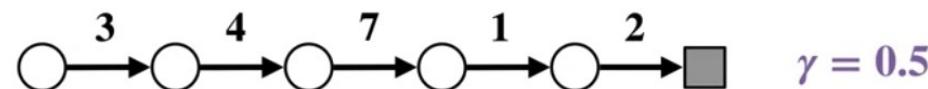
Computing returns efficiently



Monte Carlo

- Each return is included in the equation for the previous time steps return

Computing returns efficiently



$$G_0 = R_1 + \gamma G_1$$

$$G_1 = R_2 + \gamma G_2$$

$$G_2 = R_3 + \gamma G_3$$

$$G_3 = R_4 + \gamma G_4$$

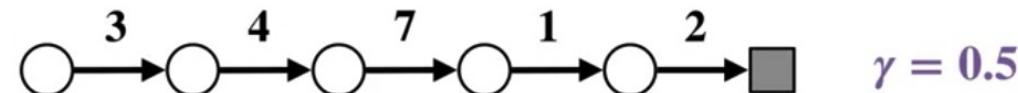
$$G_4 = R_5 + \gamma G_5$$

$$G_5 = 0$$

Monte Carlo

□ The results

Computing returns efficiently



$$G_0 = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \gamma^4 R_5 = 7$$

$$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \gamma^3 R_5 = 8$$

$$G_2 = R_3 + \gamma R_4 + \gamma^2 R_5 = 8$$

$$G_3 = R_4 + \gamma R_5 = 2$$

$$G_4 = R_5 = 2$$

$$G_5 = 0$$

Monte Carlo for Prediction

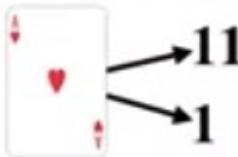
□ Example for Blackjack

Blackjack

Collect cards so that their sum is as large as possible without exceeding 21



$$= 10$$



□ How to win this game ?

Monte Carlo for Prediction

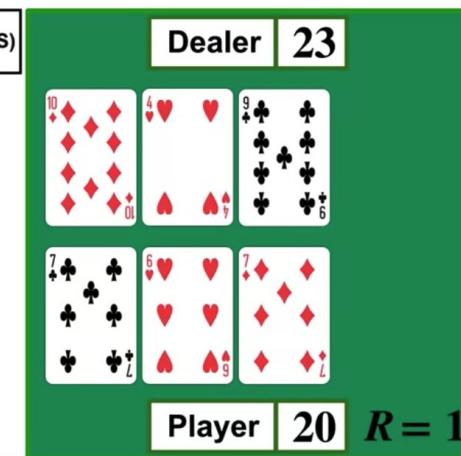
Problem formulate:

- **Undiscounted MDP where each game of blackjack corresponds to an episode**
- **Reward:** -1 for a loss, 0 for a draw, and 1 for a win
- **Actions:** Hit or Stick
- **States (200 in total):**
 - Whether the player has a usable ace (Yes or No)
 - The sum of the player's cards (12-21)
 - The card the dealer shows (Ace-10)
- Cards are dealt from a deck with replacement
- **Policy:** Stops requesting cards when the player's sum is 20 or 21

Monte Carlo for Prediction

- Suppose in the first state, the agents card shows a total of 13 with no usable ace and the dealers visible card shows 10.
- Since the agent's policy is fixed, it hits and gets a 7 moving it to 20.
- The agent sticks because its sum is 20 and it's now the dealers turn.
- The dealer c resulting in

S (Usable Ace, Sum, Dealer)	Returns(S)	V(S)
--------------------------------	------------	------



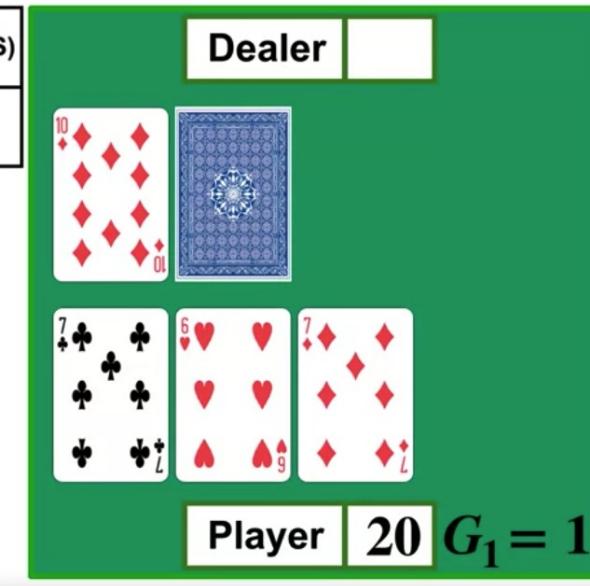
game and

Monte Carlo for Prediction

- In the last non-terminal state, the card sum was 20 with no usable ace and the dealer had a visible 10- state A.
- We add plus one to the list of returns for state A and set the value of state A equal to the average of the list

S (Usable Ace, Sum, Dealer)	Returns(S)	V(S)
A	Returns(A) = [1]	1

A = (NoAce, 20, 10)



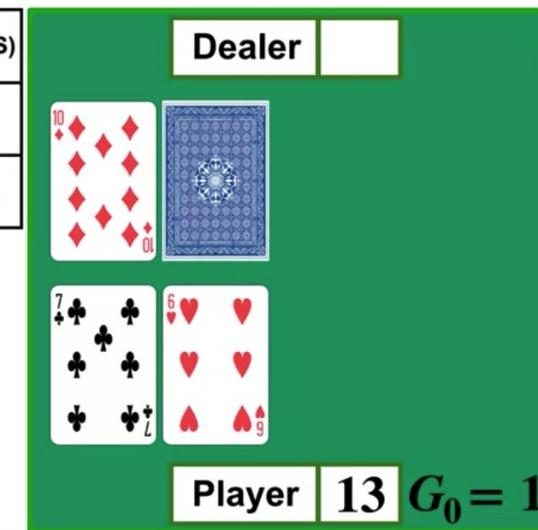
Monte Carlo for Prediction

- The second last state, state B, the agent shows 13 with no usable ace and the dealer has a visible 10.
- We add plus one to the list of returns now for state B and set the value of state B equal to the average of the list

S (Usable Ace, Sum, Dealer)	Returns(S)	V(S)
A	Returns(A) = [1]	1
B	Returns(B) = [1]	1

A = (NoAce, 20, 10)

B = (NoAce, 13, 10)



Monte Carlo for Prediction

□ Example

Suppose we have an agent navigating a gridworld environment, where the goal is to reach a certain terminal state while avoiding obstacles. We want to estimate the value function $V(s)$ for each state in the gridworld under a given policy.

Here's how we can use Monte Carlo for this prediction task:

1. **Initialization:** Initialize the value function $V(s)$ for all states randomly or with some initial guess.
2. **Episodic Interaction:** Have the agent interact with the environment by following the policy. This involves repeatedly generating episodes of interaction, where each episode consists of a sequence of state-action-reward transitions until the terminal state is reached.
3. **Episode Sampling:** For each episode, record the sequence of states visited s_1, s_2, \dots, s_T and the corresponding rewards obtained r_1, r_2, \dots, r_T , where T is the length of the episode.
4. **Return Calculation:** Compute the return G_t from each state s_t as the sum of discounted rewards obtained from that state onwards:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{T-t} r_T$$

Here, γ is the discount factor, which weights immediate rewards more heavily than future rewards.

5. **Value Estimation:** Update the value function $V(s)$ for each state s by averaging the returns obtained from all visits to that state across episodes:

$$V(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i(s)$$

where $N(s)$ is the number of visits to state s across all episodes, and $G_i(s)$ is the return obtained from the i th visit to state s .

6. **Iterative Improvement:** Repeat steps 2-5 for multiple episodes until the value function converges to a stable estimate.

Monte Carlo for Prediction

- Monte Carlo to estimate the state-value function for a simple grid world environment.

```
▶ 1 # 2.1 code demo for Monte Carlo- HoaDNT@fe.edu.vn
2 import numpy as np
3
4 class GridWorld:
5     def __init__(self):
6         self.grid_size = (3, 4)
7         self.num_actions = 4 # Up, Down, Left, Right
8         self.rewards = np.array([
9             [0, 0, 0, 0],
10            [0, 0, 0, 0],
11            [0, 0, 0, 1] # Reward of +1 in the bottom-right cell
12        ])
13         self.start_state = (2, 0)
14
15     def step(self, state, action):
16         # Define the dynamics of the environment
17         row, col = state
18         if action == 0: # Up
19             row = max(0, row - 1)
20         elif action == 1: # Down
21             row = min(self.grid_size[0] - 1, row + 1)
22         elif action == 2: # Left
23             col = max(0, col - 1)
24         elif action == 3: # Right
25             col = min(self.grid_size[1] - 1, col + 1)
26         next_state = (row, col)
27         reward = self.rewards[row, col]
28         return next_state, reward
29
```

Monte Carlo for Prediction

```

29
30 def monte_carlo(grid_world, num_episodes, gamma=1.0):
31     returns_sum = np.zeros(grid_world.grid_size)
32     returns_count = np.zeros(grid_world.grid_size)
33     V = np.zeros(grid_world.grid_size)
34
35     for _ in range(num_episodes):
36         episode = generate_episode(grid_world)
37         visited_states = set()
38         for t, (state, action, reward) in enumerate(episode):
39             if state not in visited_states:
40                 visited_states.add(state)
41                 G = sum([gamma**i * step[2] for i, step in enumerate(episode[t:])])
42                 returns_sum[state] += G
43                 returns_count[state] += 1
44                 V[state] = returns_sum[state] / returns_count[state]
45
46     return V
47
48 def generate_episode(grid_world):
49     episode = []
50     state = grid_world.start_state
51     while True:
52         action = np.random.choice(grid_world.num_actions)
53         next_state, reward = grid_world.step(state, action)
54         episode.append((state, action, reward))
55         if next_state == (2, 3): # Terminal state
56             break
57         state = next_state
58     return episode
59
60 # Create a grid world environment

```

Monte Carlo for Prediction

```
59
60 # Create a grid world environment
61 grid_world = GridWorld()
62
63 # Run Monte Carlo to estimate the state-value function
64 num_episodes = 1000
65 V = monte_carlo(grid_world, num_episodes)
66
67 # Print the estimated state-value function
68 print("Estimated State-Value Function:")
69 print(V)
70
```

Estimated State-Value Function:

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 0.]]
```

Summary

- Understand how Monte-Carlo methods can be used to estimate value functions from sampled interaction
- Identify problems that can be solved using Monte-Carlo methods
- Use Monte-Carlo prediction to estimate the value function for a given policy.



Q & A