

Reinforcement Learning (AI644)

Prepared by Dr. Ashraf Elsayed

Lecture 6



Reinforcement Learning

Lecture 6

Prepared by
Dr. Ashraf Elsayed

Sample-based Learning Methods

The objectives of this lecture is to:

- Understand Monte-Carlo methods
- 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.
- Estimate action-value functions using Monte-Carlo

Monte Carlo method: a definition

Let's suppose we have a problem in which we have to estimate a characteristics parameter of a given population. Monte Carlo method is a **numerical technique** that allows one to **generate a sample of the considered population** and **obtain a statistical estimation of required parameter**.

Monte Carlo method: brief history

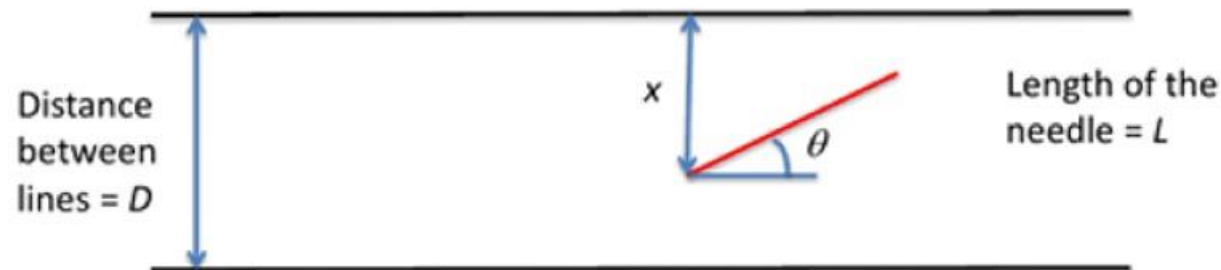
- **Comte Du Buffon (1777)**: needle tossing experiment to calculate π .
- **Laplace (1886)**: random points in a rectangle to calculate π .
- **Fermi (1930)**: random method to calculate the properties of the newly discovered neutron.
- **Manhattan project (40's)**: simulations during the initial development of the thermonuclear weapons. Von Neumann and Ulam coined the term "Monte Carlo".
- **Metropolis (1948)**: first actual Monte Carlo calculations using a computer (ENIAC) "*Equation of state Calculations by Fast Computing machines*".
- **Berger (1963)**: first complete coupled electron-photon transport code – ETRAN.
- **Exponential growth since the 1980's** with the availability of digital Computers.

Buffon's Needle

- One of the oldest problems in the field of geometrical probability, first stated in 1777.
- Drop a needle on a lined sheet of paper and determine the probability of the needle crossing one of the lines
- Remarkable result: probability is directly related to the value of π
- The needle will cross the line if $x \leq L \sin(\theta)$. Assuming $L \leq D$, how often will this occur?

$$P_{cut} = \int_0^\pi P_{cut}(\theta) \frac{d\theta}{\pi} = \int_0^\pi \frac{L \sin \theta}{D} \frac{d\theta}{\pi} = \frac{L}{\pi D} \int_0^\pi \sin \theta d\theta = \frac{2L}{\pi D}$$

- By sampling P_{cut} one can estimate π .

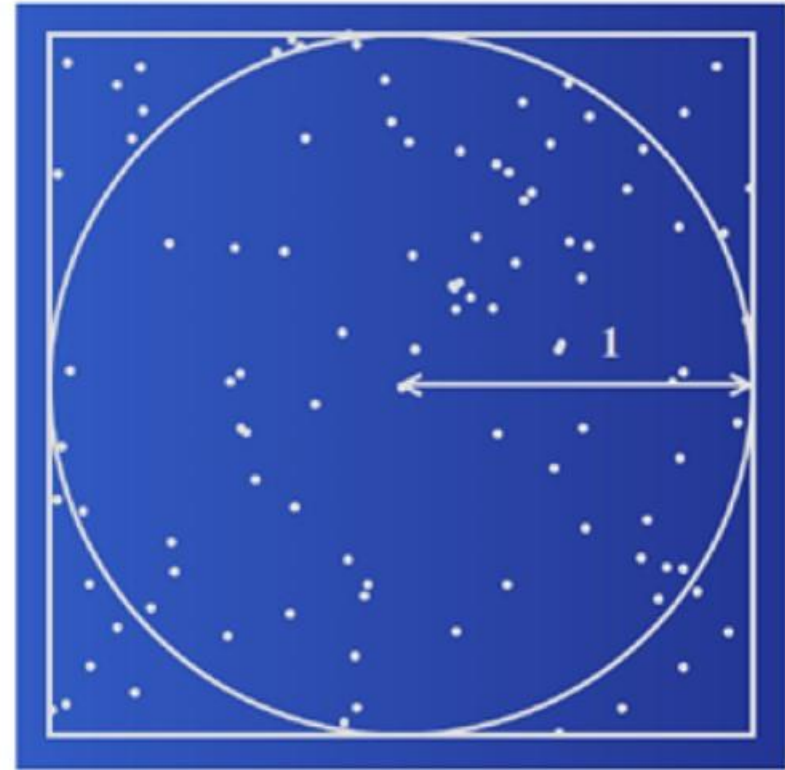


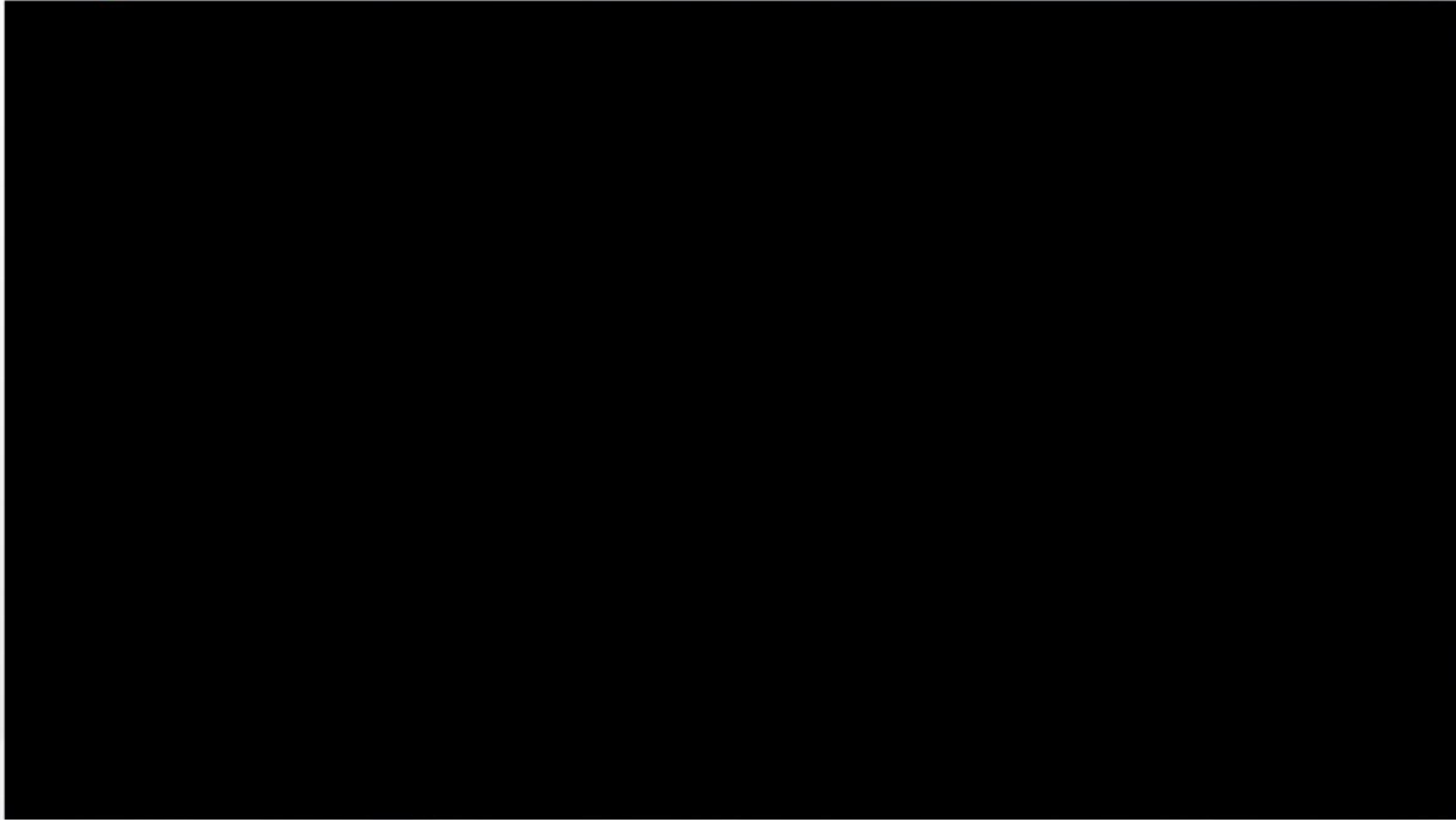
Laplace's method of calculating π (1886)

- Area of the square = 4
- Area of the circle = π
- Probability of random points inside the circle = $\pi / 4$

- Random points: N
- Random points inside circle: N_c

$$\pi \sim 4 N_c / N$$





Limitations of Dynamic Programming

- Dynamic programming algorithms use the Bellman equations to define iterative algorithms for both policy evaluation and control (finding an optimal policy).
- To use a pure Dynamic Programming approach, the agent needs to know the environments transition probabilities.
 - In some problems we do not know the environment transition probabilities
 - They involve operations over the entire state set of the MDP. If the state space is very large, this becomes computationally expensive.
- In general, the computation can be error-prone and tedious

Monte Carlo methods for prediction and control

- What is Monte Carlo?
 - The term Monte Carlo is often used more broadly for any **estimation** method that relies on **repeated random sampling**.
 - In RL Monte Carlo methods allow us to **estimate values** directly from **experience**, from **sequences of states, actions and rewards**.
 - Learning from experience is striking because the agent can accurately estimate a value function **without prior knowledge of the environment dynamics**.
- **For example**, imagine rolling a dice 12 times. With the help of Monte Carlo Methods, we can estimate values by averaging over a large number of random samples.

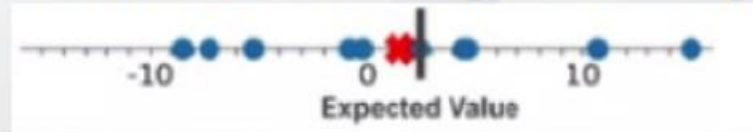
Sum of Faces	Probability	Samples
12	?	39 38 49 36 33 47 37 49 41 40 44 44
13	?	35 43 47 28 46 52 47 45 43 43 42 43
14	?	40 32 29 48 43 49 48 39 42 35 44 48
...		36 32 41 49 34 54 37 42 37 38 42 50
71	?	39 37 45 49 44 34 38 50 35 36 42 45
72	?	Average = 41.57

Monte Carlo Methods for Policy Evaluation

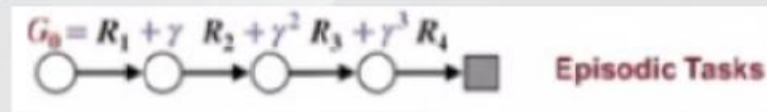
- In reinforcement learning we want to learn a value function.
- Recall that, value functions represent expected returns

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

- A Monte Carlo method for learning a value function would first observe multiple returns from the same state.
- Then, it average those observed returns to estimate the expected return from that state.
- As the number of samples increases, the average tends to get closer and closer to the expected return.



- These returns can only be observed at the end of an episode. So we will focus on Monte Carlo methods for episodic tasks.



Backup Diagram for Monte Carlo

- Entire rest of episode included
- **Only one choice** considered at each state (unlike DP)
 - thus, there will be an explore/exploit dilemma
- Does **not bootstrap** from successor state's values (unlike DP)
- Value is estimated by **mean return**
- State value estimates are independent, no bootstrapping



Monte Carlo Methods for Policy Evaluation

- Monte Carlo methods consider policies instead of arms.
- The value state S under a given policy is estimated using the average return sampled by following that policy from S to termination.
- The pseudo-code of the Monte Carlo prediction can be seen below

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

$G \leftarrow 0$

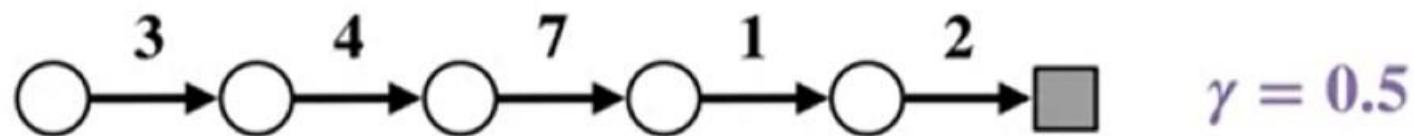
Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

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

Append G to $Returns(S_t)$

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

Efficient MC computation of returns



$$G_0 = R_1 + \gamma G_1$$

$$G_1 = R_2 + \gamma G_2$$

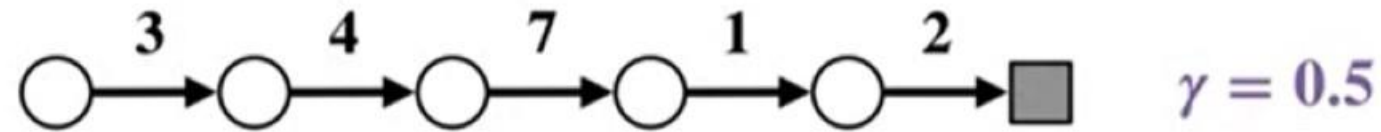
$$G_2 = R_3 + \gamma G_3$$

$$G_3 = R_4 + \gamma G_4$$

$$G_4 = R_5 + \gamma G_5$$

$$G_5 = 0$$

Efficient MC computation of returns



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

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

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

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

$$G_4 = R_5 = 2$$

$$G_5 = 0$$

Monte Carlo Methods for Policy Evaluation

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

$G \leftarrow 0$

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

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

Append G to $Returns(S_t)$

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

Calculating the Expected value
with Monte Carlo

Incremental Mean

- ▶ The mean $\mu_1, \mu_2, \dots$ of a sequence $x_1, x_2, \dots$ can be computed incrementally:

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$

Incremental Monte Carlo Updates

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, \dots, S_T$
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$

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

- In non-stationary problems, it can be useful to track a **running mean**, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

Example: Using Monte Carlo for Prediction

- Let's discuss the specifics of the Monte Carlo prediction algorithm using an example with a card game, **blackjack**



Example: Using Monte Carlo for Prediction

- Problem Formulation
 - Undiscounted MDP where each game of blackjack corresponds to an episode
 - -1 for a loss, 0 for a draw, and 1 for a win
 - 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

Example: Using Monte Carlo for Prediction

- 1st Episode

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

Dealer

23

10
♦

4
♥

9
♣

7
♣

6
♥

7
♦

Player

20

$R = 1$

Example: Using Monte Carlo for Prediction

- 1st Episode (working backwards)

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

A = (NoAce, 20, 10)



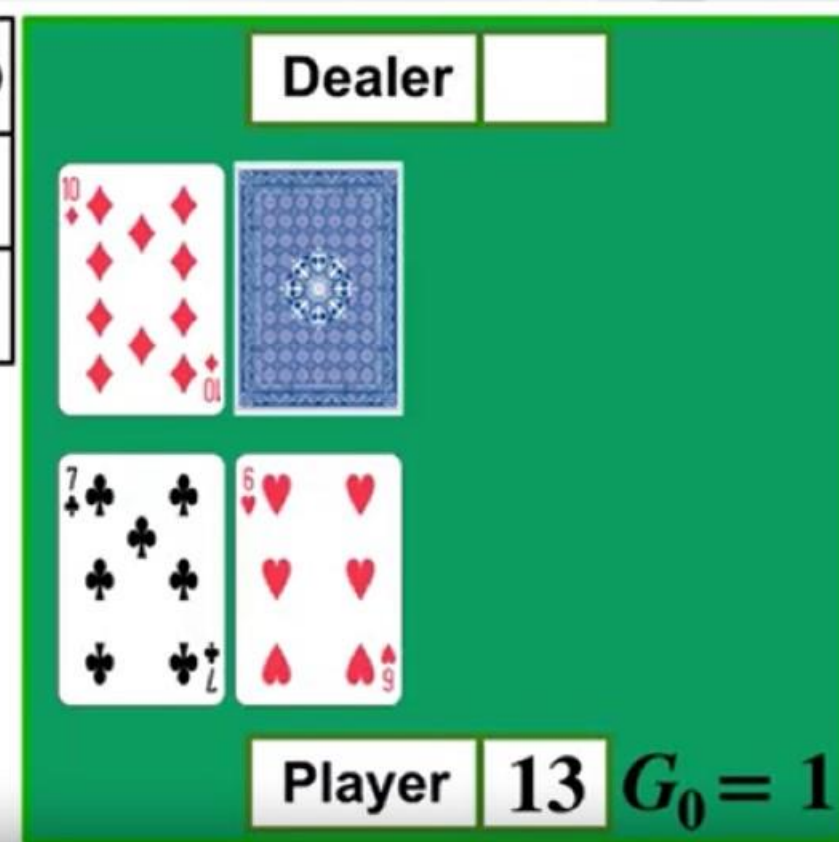
Example: Using Monte Carlo for Prediction

- 1st Episode (working backwards)

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 Policy Evaluation: Example

Dealer card:



Agent's hand:



State

Action

Next State

Reward

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



State

Action

Next State

Reward

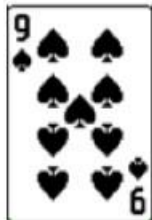
19, 10, no

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



Agent sum, dealer's card, ace?

State

Action

Next State

Reward

19, 10, no

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



Agent sum, dealer's card, ace?

State

Action

Next State

Reward

19, 10, no

HIT

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



Agent sum, dealer's card, ace?

<u>State</u>	<u>Action</u>	<u>Next State</u>	<u>Reward</u>
19, 10, no	HIT	22, 10, no	-1

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



Agent sum, dealer's card, ace?

Bust!
(reward = -1)

State

Action

Next State

Reward

19, 10, no

HIT

22, 10, no

-1

Monte Carlo Policy Evaluation: Example

<u>State</u>	<u>Action</u>	<u>Next State</u>	<u>Reward</u>
19, 10, no	HIT	22, 10, no	-1

Upon episode termination, make the following value function updates:

$$V((19, 10, no)) \leftarrow -1$$

Monte Carlo Policy Evaluation: Example

Next episode...

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



State

Action

Next State

Reward

13, 10, no

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



State

Action

Next State

Reward

13, 10, no

HIT

16, 10, no

0

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



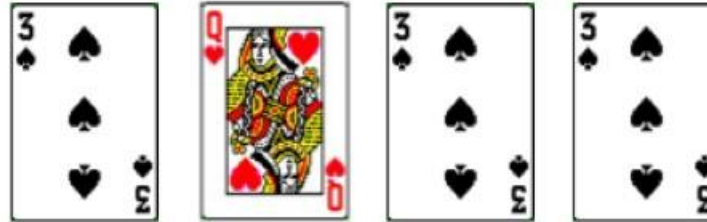
<u>State</u>	<u>Action</u>	<u>Next State</u>	<u>Reward</u>
13, 10, no	HIT	16, 10, no	0
16, 10, no			

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



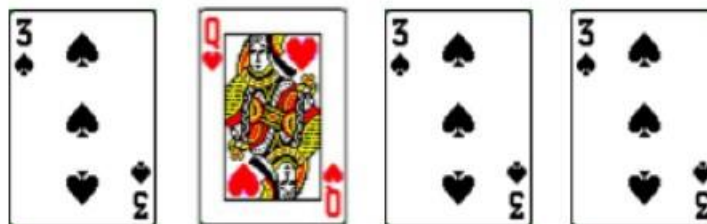
<u>State</u>	<u>Action</u>	<u>Next State</u>	<u>Reward</u>
13, 10, no	HIT	16, 10, no	0
16, 10, no	HIT	19, 10, no	0

Monte Carlo Policy Evaluation: Example

Dealer card:



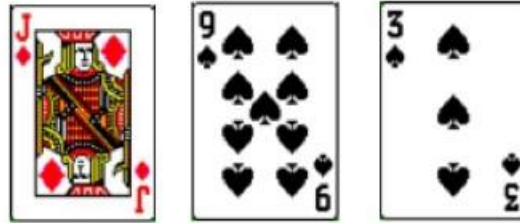
Agent's hand:



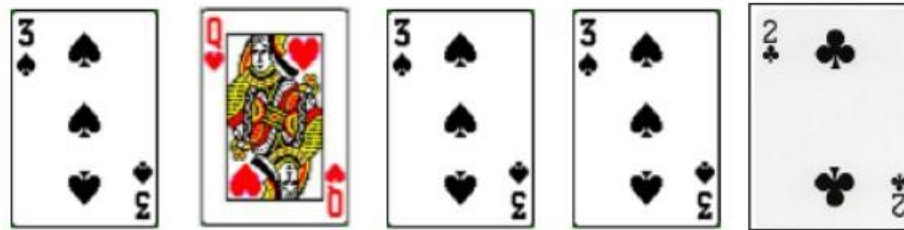
<u>State</u>	<u>Action</u>	<u>Next State</u>	<u>Reward</u>
13, 10, no	HIT	16, 10, no	0
16, 10, no	HIT	19, 10, no	0
19, 10, no			

Monte Carlo Policy Evaluation: Example

Dealer card:



Agent's hand:



<u>State</u>	<u>Action</u>	<u>Next State</u>	<u>Reward</u>
13, 10, no	HIT	16, 10, no	0
16, 10, no	HIT	19, 10, no	0
19, 10, no	HIT	21, 22, no	1

Monte Carlo Policy Evaluation: Example

<u>State</u>	<u>Action</u>	<u>Next State</u>	<u>Reward</u>
13, 10, no	HIT	16, 10, no	0
16, 10, no	HIT	19, 10, no	0
19, 10, no	HIT	21, 22, no	1

Upon episode termination, make the following value function updates:

$$V((13, 10, no)) \leftarrow 1$$

$$V((16, 10, no)) \leftarrow 1$$

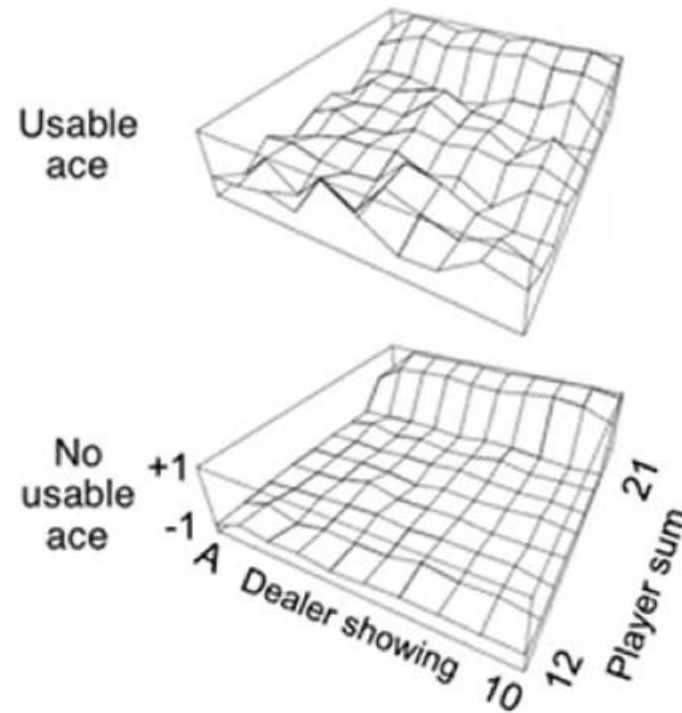
$$V((19, 10, no)) \leftarrow (-1 + 1)/2$$

Example: Using Monte Carlo for Prediction

- After 10000 Episodes

Approximate state-value functions

After 10,000 episodes

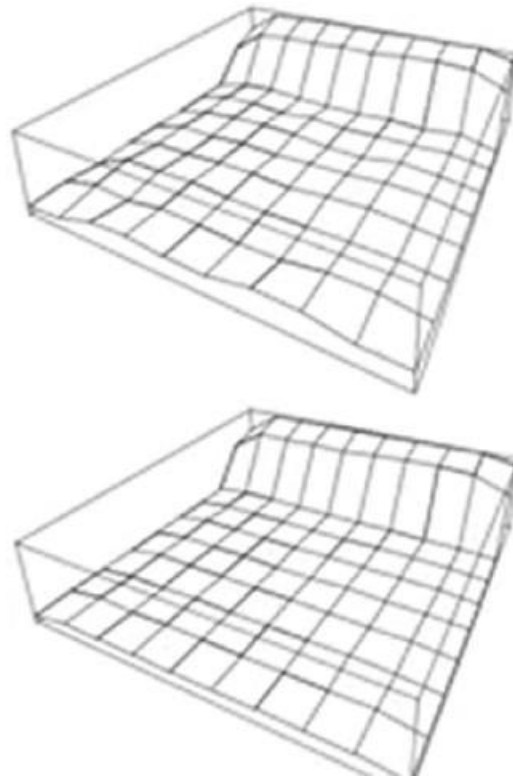


Example: Using Monte Carlo for Prediction

- After 500,000 Episodes

Approximate state-value functions

After 500,000 episodes



Implications of Monte Carlo Learning

- First, Monte Carlo learns directly from experience. So there's no need to keep a large model of the environment (Model-free RL).
- Monte Carlo methods can estimate the value of an individual state independently of the values of any other states. In dynamic programming, the value of each state depends on the values of other states.
- Finally, the computation needed to update the value of each state along the way doesn't depend in any way on the size of the MDP. Rather it depends on the length of the episode.

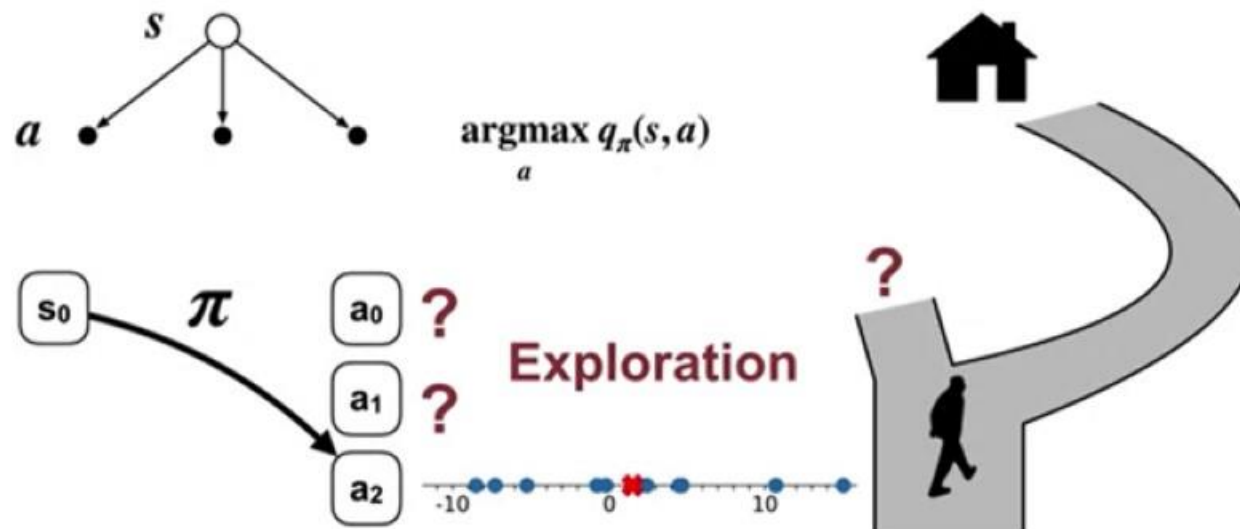
Using Monte Carlo for Action Values

- So far, we've talked about using Monte Carlo to learn **state-value functions** for a **fixed policy**.
- This time, we'll focus on learning **action values**.

- Recall that

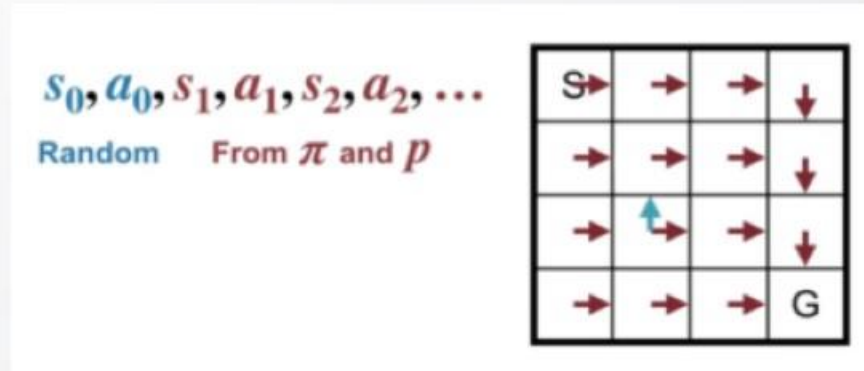
$$q_{\pi}(s, a) \doteq \mathbb{E}_{\pi} [G_t | S_t = s, A_t = a]$$

- **Action-values are important for learning a policy.**



Exploring Starts

- One way to maintain exploration is called **exploring starts**. In this case we must guarantee that episodes start in every state-action pair. Afterwards, the agent simply follows its policy.



- Exploring starts will randomly sample a state and action at the start of an episode shown in blue. After the initial action, the agent will follow the red policy until the episode ends.
- We must be able to set the start state in this way to evaluate a deterministic policy, like we can in this grid world.
- Other exploration strategies like **epsilon-greedy** can be used to evaluate stochastic policies.

Using Monte Carlo methods for generalized policy iteration

- General policy iteration (GPI) includes a policy evaluation and a policy improvement step.
- GPI algorithms produce sequences of policies that are at least as good as the policies before them.
- For the policy improvement step, we can make the policy greedy with respect to the agent's current action value estimates.
- For the policy evaluation step, we will use a Monte Carlo method to estimate the action values.

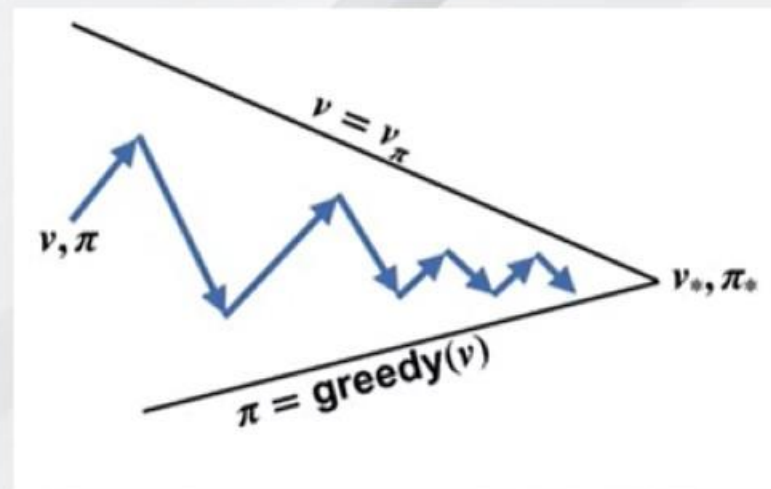
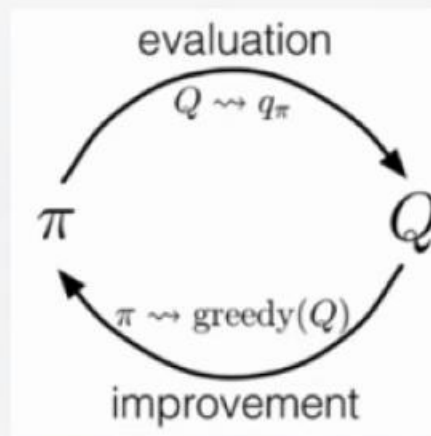
$$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \dots$$

Using Monte Carlo methods for generalized policy iteration

- **Improvement**

$$\pi_{k+1}(s) \doteq \operatorname{argmax}_a q_{\pi_k}(s, a)$$

- **Evaluation:** Monte Carlo Prediction



- Monte Carlo control methods combined **policy improvement** and **policy evaluation** on an episode-by-episode basis.

The MC generalized policy iteration algorithm

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

Initialize:

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

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

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

Loop forever (for each episode):

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

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

$G \leftarrow 0$

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

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

Append G to $Returns(S_t, A_t)$

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

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

Solving Blackjack Example

S,A	Returns(S,A)	Q(S,A)		$\pi(S)$
		Hit	Stick	



- Exploring starts requires that episodes begin with a random state and action.
- Luckily for us, our version of blackjack naturally starts in random states. Then, all we have to do is to randomly select the first action in each episode.

Solving Blackjack Example

S,A	Returns(S,A)	Q(S,A)		$\pi(S)$
		Hit	Stick	

Action: Hit



S,A	Returns(S,A)	Q(S,A)		$\pi(S)$
		Hit	Stick	

Action: Stick



Solving Blackjack Example

S,A	Returns(S,A)	Q(S,A)		$\pi(S)$
		Hit	Stick	
X,Stick	Returns(X,Stick) = [1]	0	1	

X = (NoAce, 20, 8)



Solving Blackjack Example

S,A	Returns(S,A)	Q(S,A)		$\pi(S)$
		Hit	Stick	
X,Stick	Returns(X,Stick) = [1]	0	1	Stick

$X = (\text{NoAce}, 20, 8)$



Solving Blackjack Example

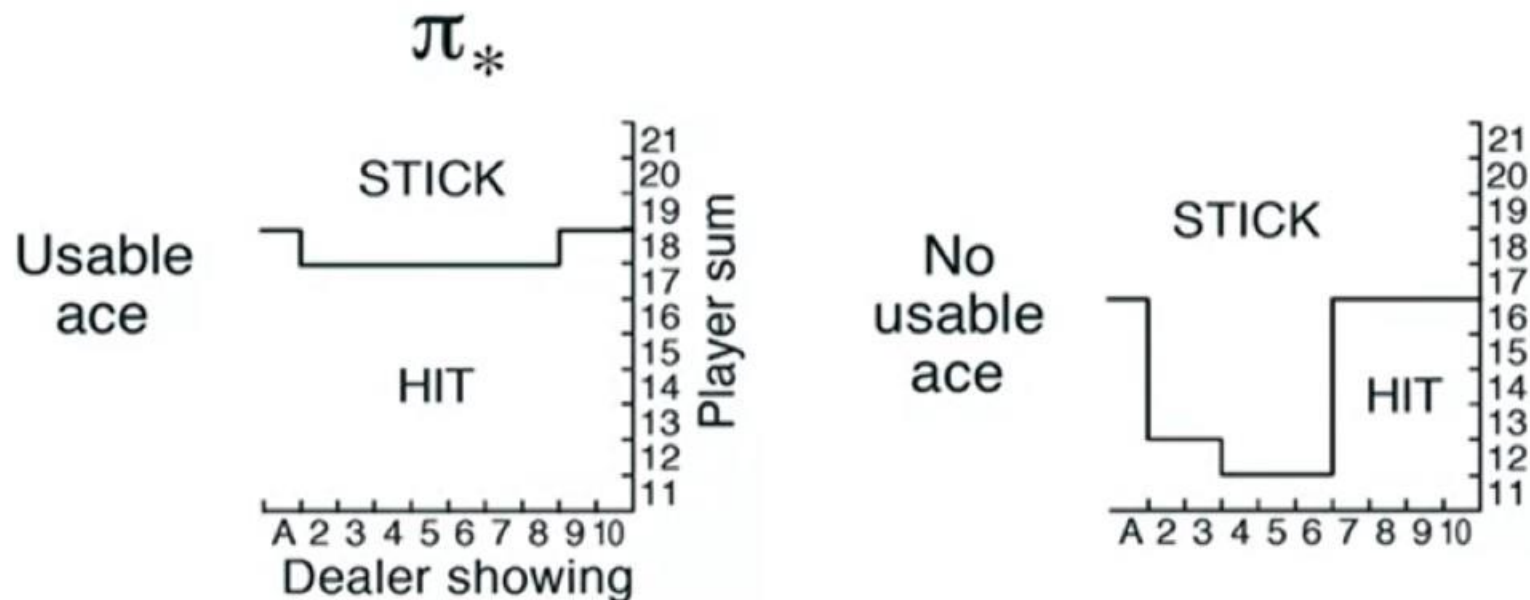
S,A	Returns(S,A)	Q(S,A)		$\pi(S)$
		Hit	Stick	
X,Stick	Returns(X,Stick) = [1]	0	1	Stick
Y,Hit	Returns(Y,Hit) = [1]	1	0	Hit

X = (NoAce, 20, 8)

Y = (NoAce, 13, 8)



Solving Blackjack Example



- When repeating this process over many episodes. The action values and the policy will approach their optimal values.
- Let's look at the optimal policy that the agent found after we ran it for a really long time. Notice how the agent plays when it has the usable ace. For most dealer cards, the agent hits until it has the sum near 19.
- With a usable ace, the agent has a lot more flexibility in calculating the sum of its cards, so the policy is much more aggressive.



Thank You