

# Reinforcement Learning (CS F317)



**BITS** Pilani  
Hyderabad Campus

Dr. Paresh Saxena  
Dpt. of Computer Science & Information Systems  
Email: psaxena@hyderabad.bits-pilani.ac.in

# Course Handout Discussion

**Lecturer:**

Parekh Saxena – psaxena@hyderabad.bits-pilani.ac.in, <https://psaxena86.github.io/>

### **Teaching Assistant:**

Nida Fatima - p20190504@hyderabad.bits-pilani.ac.in

### **Announcements:**

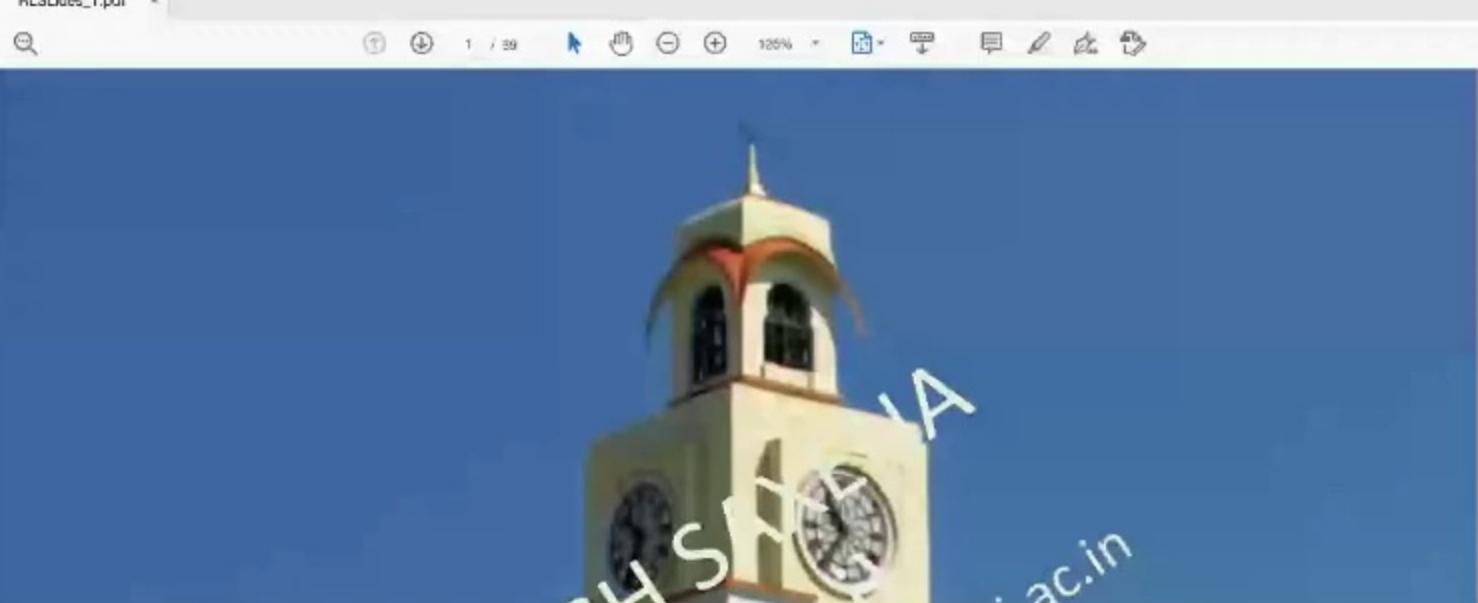
Via CMS

### **Text Book:**

---

Reinforcement Learning: An Introduction, Sutton and Barto, 2nd Edition.

- \* No Prerequisites on ML/DL required.
  - \* Brush up basics of probability and random variables !!



# Reinforcement Learning (CS F317)



**BITS** Pilani  
Hyderabad Campus

Dr. Paresh Saxena  
Dpt. of Computer Science & Information Systems  
Email: psaxena@hyderabad.bits-pilani.ac.in

# Course Handout Discussion

**Lecturer:**

Parekh Saxena – psaxena@hyderabad.bits-pilani.ac.in, <https://psaxena86.github.io/>

### **Teaching Assistant:**

Nida Fatima - p20190504@hyderabad.bits-pilani.ac.in

### **Announcements:**

Via CMS

### **Text Book:**

---

Reinforcement Learning: An Introduction, Sutton and Barto, 2nd Edition.

- \* No Prerequisites on ML/DL required.
  - \* Brush up basics of probability and random variables !!



# Course Evaluation

Component	Duration	Weightage	Date&Time	Mode
Mid-Term exam	90 Mins	35%	As announced in the Timetable	Open Book
Course Project (with final presentation/viva)	-	25% (5% will be evaluated before the mid-sem)	Details will be announced in September	Open Book
Comprehensive	120 Mins	40%	As announced in the Timetable	Closed Book

\* Lecture Videos will be shared.

\* Slides will not be shared.



# Course Topics

---

- Bandit Problems
- Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal Difference Learning
- Planning
- Function Approximation
- Multi-agent Learning

I

psaxena@hyderabad.bits-pilani.ac.in

# Reinforcement Learning

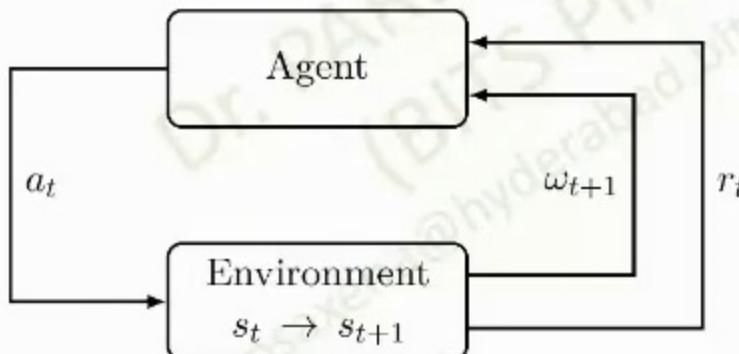
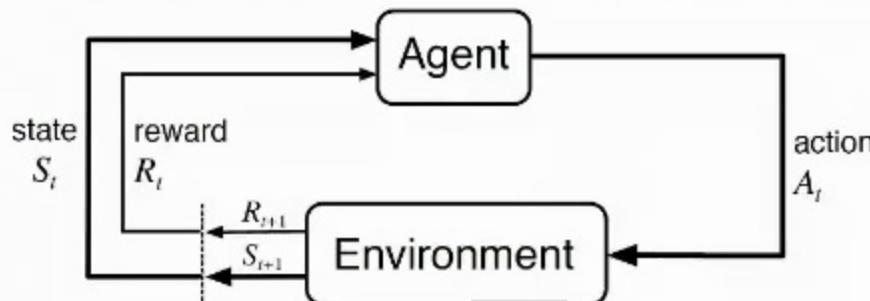
- Interaction and learning from the environment



Complete Independent Goal-Seeking Learners/Agents:

- What are rewards ?
- What are actions ?
- What are states ?

## Common RL state Diagrams



Adding observation “ $w_t$ ”

1 Markov Property

- $\mathbb{P}(\omega_{t+1} \mid \omega_t, a_t) = \mathbb{P}(\omega_{t+1} \mid \omega_t, a_t, \dots, \omega_0, a_0)$ , and
  - $\mathbb{P}(r_t \mid \omega_t, a_t) = \mathbb{P}(r_t \mid \omega_t, a_t, \dots, \omega_0, a_0)$ .

Markov Property: Future of the process  
only depends on the current observation.

**Fully observable MDP – observation is the same as the state,  $w_t = s_t$ .**



# xample

## Human Infant Learning



Agent: Baby

Environment: Workspace with things around

Actions: movement of arms, legs

Reward: curiosity, satisfaction

RL learning problem includes:

- What to do ?
- How to map situations to actions ?
- How to maximize a numerical reward signal ?

Learner/Agent is not told which actions to take.

Learner/Agent discovers the actions with the goal of maximizing the rewards.

Trail-and-Error Search

Possibility of delayed reward: sacrifice short-term gains for greater long-term benefits

Need to explore and exploit

Goal-directed agent interacting with an uncertain environment

---

# Reinforcement Learning – Key Features

- RL learning problem includes:
  - What to do ?
  - How to map situations to actions ?
  - How to maximize a numerical reward signal ?
- Learner/Agent is not told which actions to take.
- Learner/Agent discovers the actions with the goal of maximizing the rewards.
- Trail-and-Error Search
- Possibility of delayed reward: sacrifice short-term gains for greater long-term benefits
- Need to explore and exploit
- Goal-directed agent interacting with an uncertain environment



# Reinforcement Learning – Key Features

---

- RL learning problem includes:
  - What to do ?
  - How to map situations to actions ?
  - How to maximize a numerical reward signal ?
- Learner/Agent is not told which actions to take.
- Learner/Agent discovers the actions with the goal of maximizing the rewards.
- Trail-and-Error Search
- Possibility of delayed reward: sacrifice short-term gains for greater long-term benefits
- Need to explore and exploit
- Goal-directed agent interacting with an uncertain environment

# Reinforcement Learning - II

- How RL is different from Supervised learning and Unsupervised learning ?
  - RL not supervised: correct actions are not provided
  - RL not unsupervised: reward signal informs correct action

**RL is third category of ML – learning to act to maximize rewards.**

Innovate

achieve

lead

# RL in Real-time

Robot Learning to walk:

<https://www.youtube.com/watch?v=SBf5-eF-Elw>



# RL: Exploitation and Exploration

---

The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future.

- Exploitation: Agents trying to do the similar actions again that give them high rewards.
- But what if a higher reward is possible by trying a new action ? – Exploration.
- There should be a balance between exploitation and exploration.

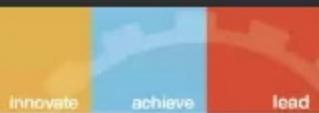
# Example: Exploitation vs Exploration



Given:

- Five doors
- Known: Behind door 3 is Rs. 100 and behind door 5 is Rs. 2000
- Not-known: What is behind door 1 and door 2 and door 4 ?
- You are given three chance to open doors to earn maximum money. You are allowed to open the same door again. What will you do ?

- You will earn Rs. 6000



## Example: Exploitation vs Exploration



Door 1



Door 2



Door 3



Door 4



Door 5

A different approach:

- Open Door 1
- Open Door 2
- Open Door 4
- You will earn a lot !!

# Example: Exploitation vs Exploration



Door 1



Door 2



Door 3



Door 4



Door 5

A different approach:

- Open Door 1
- Open Door 2
- Open Door 4
- You will earn a lot !!

# Exploitation vs Exploration



Door 1



Door 2



Door 3



Door 4



Door 5

A different approach:

- Open Door 1
- Open Door 2
- Open Door 4
- You will earn a lot !!

Innovate

achieve

lead

# RL: Key Elements - I

- Policy:
  - A policy defines the learning agent's way of behaving at a given time.
  - A policy is a mapping from perceived states of the environment to actions to be taken when in those states
  - They may be represented as a table, or involve some complex optimization
- Reward Signal:
  - A reward signal defines the goal in a reinforcement learning problem.
  - The agent's sole objective is to maximize the total reward it receives over the long run.

# Example: Exploitation vs Exploration



Door 1

Door 2

Door 3

Door 4

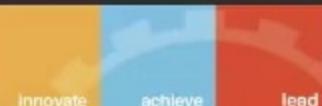
Door 5

Greedy Approach (Zero Exploration):

- Open Door 5
- Open Door 5
- Open Door 5
- You will earn Rs. 6000

# RL: Key Elements - II

- Value Function:
  - Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run.
  - The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- Model of the Environment:
  - Given a state and action, the model might predict the resultant next state and next reward.
  - Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced.



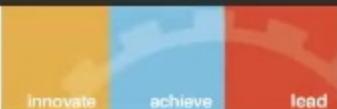
# RL Variants

---

- Methods to have exact solutions: typically smaller action and state spaces
- Methods to have approximate solutions: usually larger action and state spaces
- Single-state problems: Bandit Problems
- Multiple—states: Markov Decision Process

# Multi-arm Bandits

## Types of Feedback



# Types of Feedback

- Evaluative Feedback – based on the action taken (e.g., in RL)
- Instructive Feedback – independent of the action taken (e.g. basis of supervised learning)

# Bandit Problems: Origin



- Each slot machine can be thought as one-armed bandits.
- Which slot machine should we play each turn when their payoffs are unknown and not the same ?



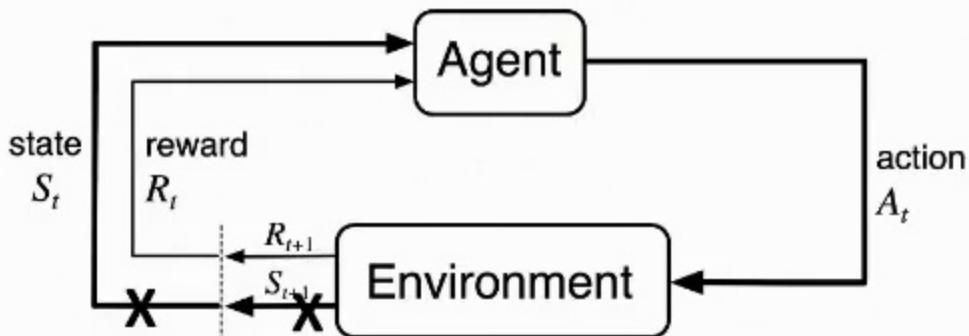
# Bandit Problems: Origin



- Each slot machine can be thought as one-armed bandits.
- Which slot machine should we play each turn when their payoffs are unknown and not the same ?

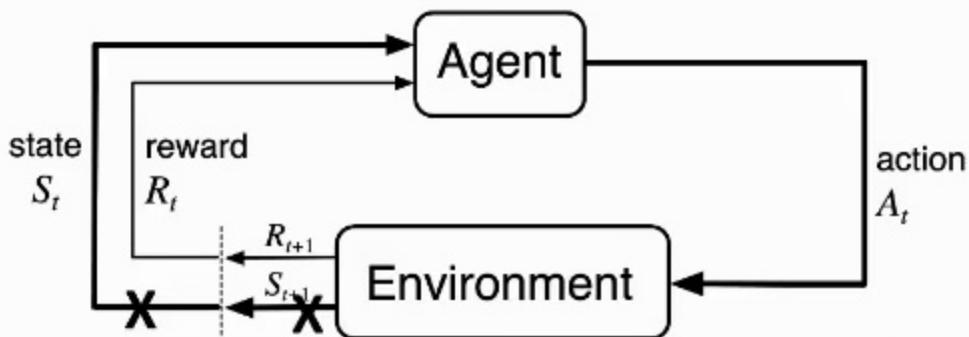
# Bandit Problems

- One-state problems
- Understanding of Exploration/Exploitation Tradeoff
- Set of actions (also known as arms)
- Space of Rewards (often rescaled to be  $(0, 1]$ )



# Bandit Problems

- One-state problems
- Understanding of Exploration/Exploitation Tradeoff
- Set of actions (also known as arms)
- Space of Rewards (often rescaled to be  $(0, 1]$ )



# Bandit Problems: Applications

- Clinical Trials – designing the medical drugs
  - Changing the drug component - separate action
  - Reward – patient's response
  - Take average result over several patients
- Online ad placement
- Web page personalization
- Games
- Networks

innovate

achieve

lead

# Online Ad Placement

The Hindu website interface showing various online ad placements:

- Top Right Banner:** A yellow, blue, and red horizontal bar with the words "innovate", "achieve", and "lead".
- Left Sidebar:**
  - Logo:** BITS Pilani logo.
  - Text:** One-year Post-Graduate Programme in Business Leadership @ IIM Kochi.
  - Section:** Highlights.
  - Text:** One year, full-time, residential programme.
  - Text:** Two weeks international immersion.
  - Text:** Renowned faculty. Happiness.
  - Text:** Join PGPBL @ IIMK.
- Main Content Area:**
  - Headline:** No immediate plan to ban 2 Hurriyat factions: official
  - Image:** Photo of a group of men.
  - Text:** SPECIAL CORRESPONDENT: JGU first became aware of NIA in March 2019.
  - Section:** NATIONAL.
  - Text:** INFRASTRUCTURE: InfraCo gets till September 15 to fix glitches in I-T portal.
  - Section:** NATIONAL.
  - Text:** PM Modi non-committal on Bihar team's demand for caste census.
  - Section:** CRICKET.
  - Text:** The summer of '71 that scripted a new chapter in Indian cricket.
  - Bottom Navigation:** A news summary section with numbered items from 1 to 11, each with a small thumbnail and a brief description.
- Right Sidebar:**
  - Section:** WORLDWIDE WITH SURESH KHANNA.
  - Text:** The latest optoelectronics technologies, applications, and market news.
  - Text:** Subscribe for free today.
  - Section:** Laser Focus World.
  - Text:** Most trusted resource for photonics technologies & solutions for technical...
  - Text:** Subscribe.
  - Image:** Photos of various optical components.
  - Section:** TOP-SELLING BOOKS.
  - Text:** AMAZING DEALS FROM 200+ STORES. Buy special for your...
  - Text:** Explore.
  - Image:** Book covers.
  - Section:** SAI University.
  - Text:** First university in India to offer Cognitive Neuroscience as an undergraduate major.
  - Text:** APPLY NOW.
  - Text:** LEARN MORE SCHOLARSHIPS. CELEBRATE TALENT AND ALL...
  - Image:** Brain circuit diagram.

Innovate

achieve

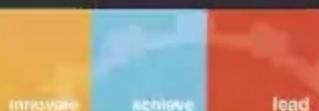
lead

# n-armed bandits

- n actions,  $a = [a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a_6 \ a_7 \ a_8 \ a_9 \ a_{10}]$
- On each time stamp,  $t = 1, 2, 3, \dots$ , choose action  $A_t = a$  (or  $a_i$ ) and receive a scalar reward sampled from some unknown random variable  $R_t$ , where:

$$q * (a) = \mathbb{E}[R_t | A_t = a]$$

- $R_t$  are iid (independently and identically distributed)

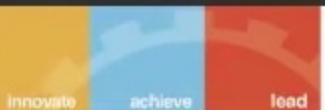


## n-armed bandits

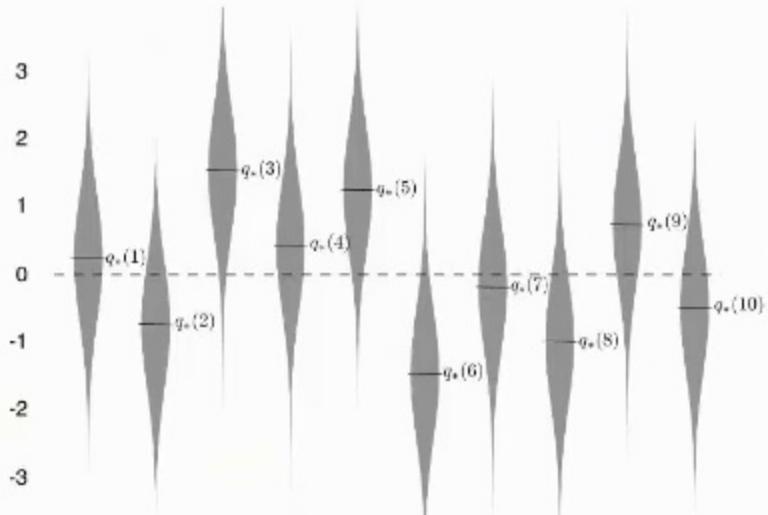
- n actions,  $\mathbf{a} = [a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a_6 \ a_7 \ a_8 \ a_9 \ a_{10}]$
- On each time stamp,  $t = 1, 2, 3, \dots$ , choose action  $A_t = a$  (or  $a_i$ ) and receive a scalar reward sampled from some unknown random variable  $R_t$ , where:

$$q^*(a) = \mathbb{E}[R_t | A_t = a]$$

- $R_t$  are iid (independently and identically distributed)

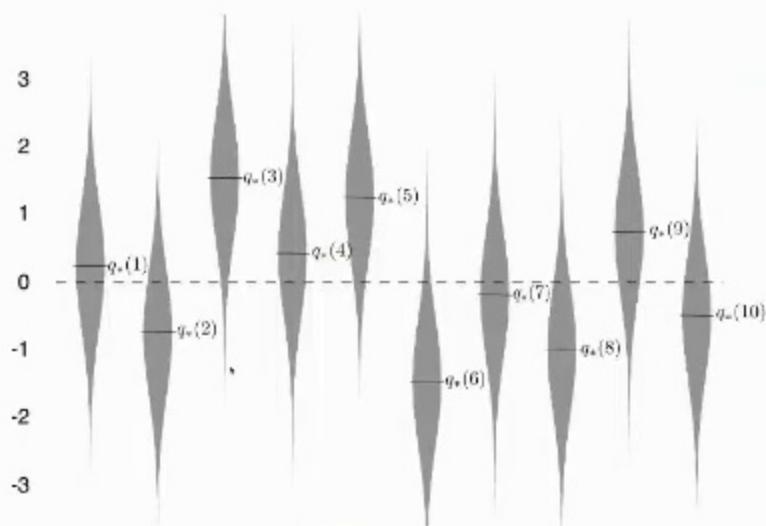


# 10-arm bandit example



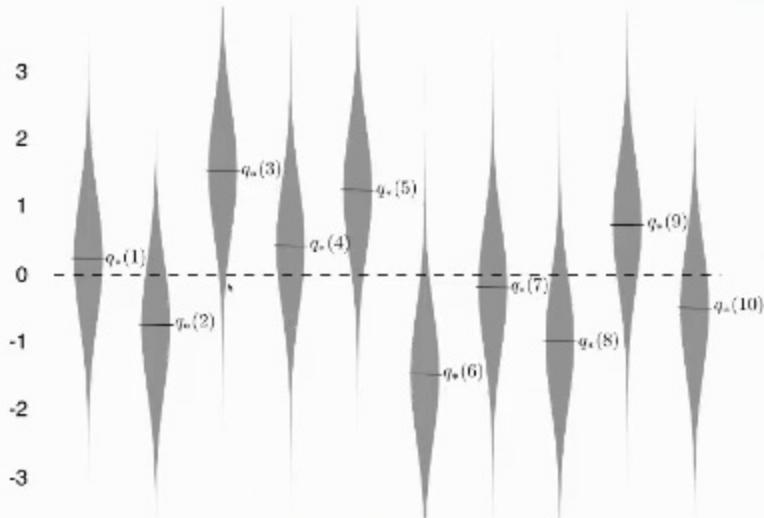
- What should we do when the distribution is known ? Which action will we take ?

# 10-arm bandit example



- If the distribution is known, we will choose the third action !!!!
- But in reality, these distributions are unknown.
- So, given these unknown distribution, what should we do ??

# 10-arm bandit example



- If the distribution is known, we will choose the third action !!!!
- But in reality, these distributions are unknown.
- So, given these unknown distribution, what should we do ??



# n-armed bandits

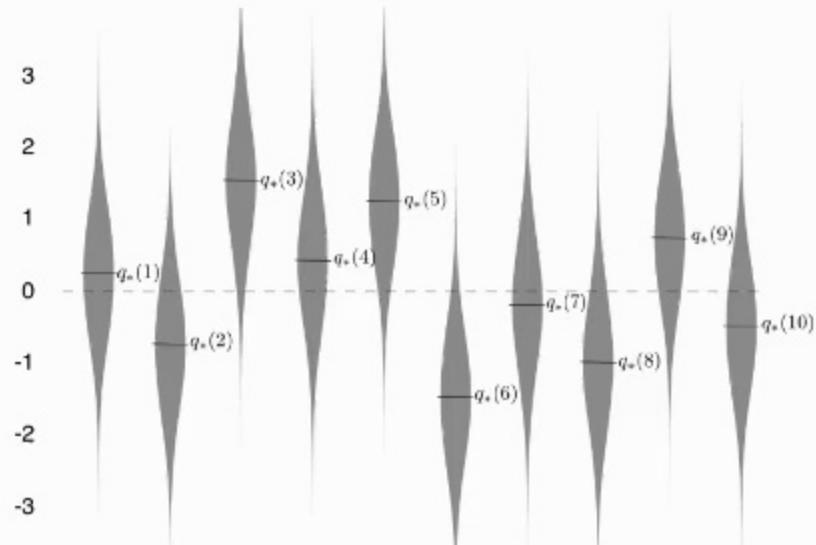
- n actions,  $a = [a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a_6 \ a_7 \ a_8 \ a_9 \ a_{10}]$
- On each time stamp,  $t = 1, 2, 3, \dots$ , choose action  $A_t = a$  (or  $a_i$ ) and receive a scalar reward sampled from some unknown random variable  $R_t$ , where:

$$q * (a) = \mathbb{E}[R_t | A_t = a]$$

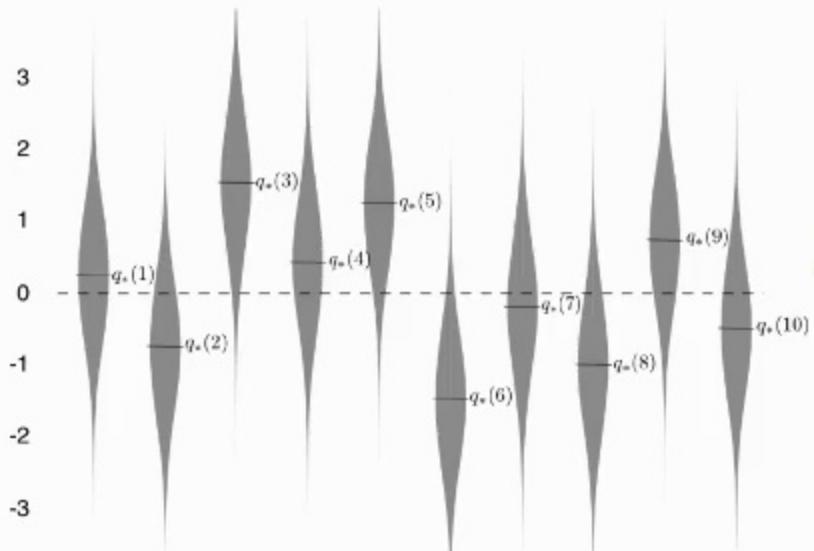
- $R_t$  are iid (independently and identically distributed)



# 10-arm bandit example



- What should we do when the distribution is known ? Which action will we take ?



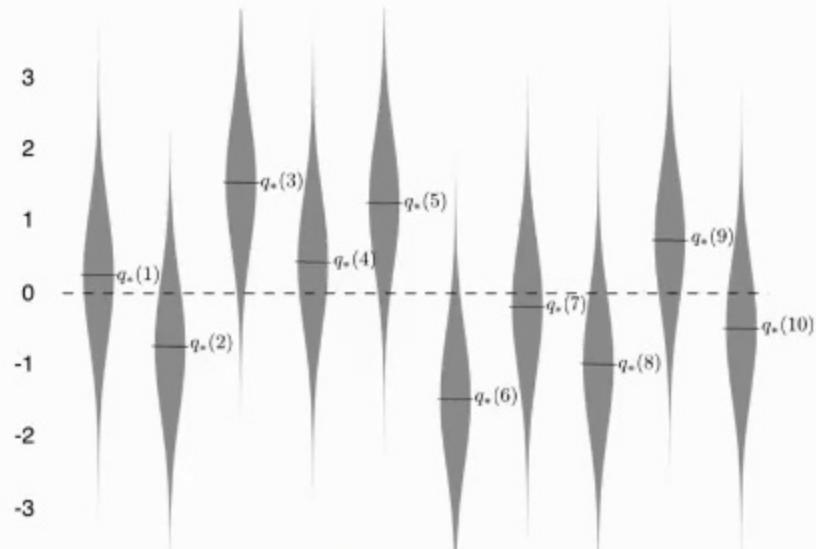
- If the distribution is known, we will choose the third action !!!!
- But in reality, these distributions are unknown.
- So, given these unknown distribution, what should we do ??

# 10-arm bandit example

innovate

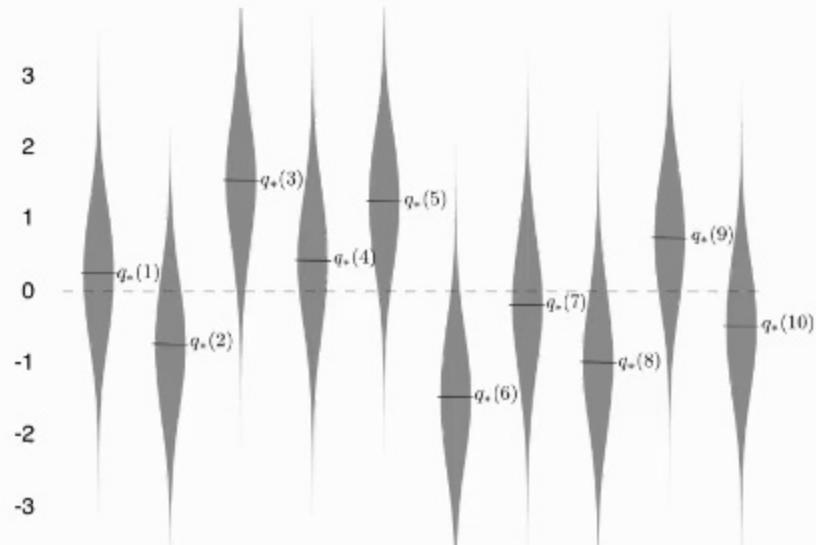
achieve

lead

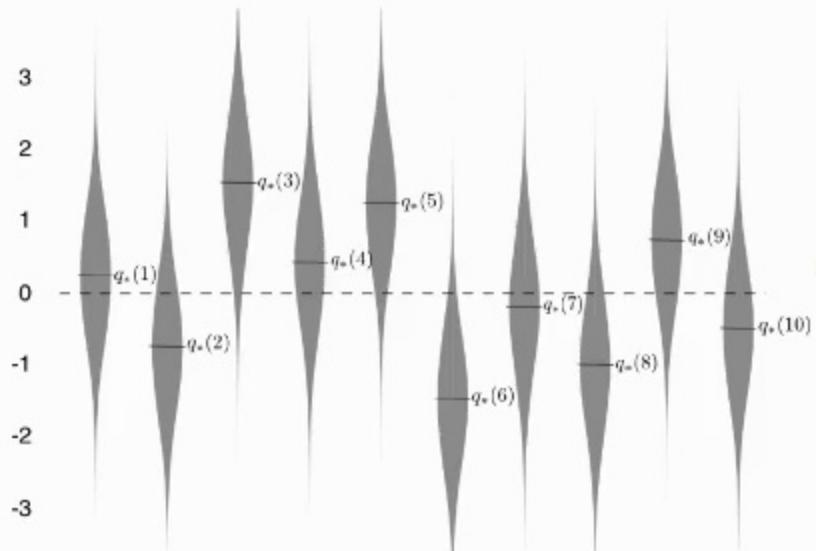


- If the distribution is known, we will choose the third action !!!!
- But in reality, these distributions are unknown.
- So, given these unknown distribution, what should we do ??

# 10-arm bandit example



- What should we do when the distribution is known ? Which action will we take ?



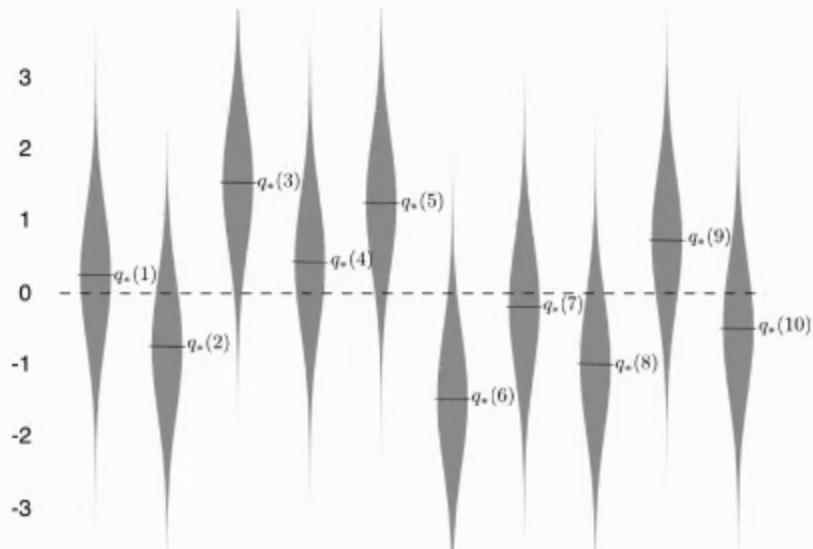
- If the distribution is known, we will choose the third action !!!!
- But in reality, these distributions are unknown.
- So, given these unknown distribution, what should we do ??

# 10-arm bandit example

innovate

achieve

lead



- If the distribution is known, we will choose the third action !!!!
- But in reality, these distributions are unknown.
- So, given these unknown distribution, what should we do ??

# Greedy Approach – Choose the action with best value estimate

- Lets first define the value estimate:
- At  $t^{\text{th}}$  time action  $a$  has been chosen  $N_t(a)$  prior to  $t$
- Rewards received by action  $a$  are:  $R_1, R_2, \dots, R_{N_t(a)}$
- Value Estimate is

$$Q_t(a) = \frac{R_1 + R_2 + \dots + R_{N_t(a)}}{N_t(a)}$$

- Greedy Approach – Choose the action that give the best value estimate, i.e.,

$$A_t = \operatorname{argmax}_a Q_t(a)$$

Is it the best approach ??

# Greedy Approach: Example

Value Estimate

	t=1
$Q_t(a_1)$	0
$Q_t(a_2)$	0
$Q_t(a_3)$	0
$Q_t(a_4)$	0
$Q_t(a_5)$	0
$Q_t(a_6)$	0
$Q_t(a_7)$	0
$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

If expected rewards are known, then we will always select action 1 with the maximum expected award ☺ but life is not that simple !!

Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0..99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

# Greedy Approach: Example (t=1)

innovate

achieve

lead

Value Estimate

	t=1
$Q_t(a_1)$	0
$Q_t(a_2)$	0
$Q_t(a_3)$	0
$Q_t(a_4)$	0
$Q_t(a_5)$	0
$Q_t(a_6)$	0
$Q_t(a_7)$	0
$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

- At t=1, all previous value estimates are zero (initialization)
- Select a random action
- Lets assume we assume action 3
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.22 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_6)$  0 $Q_t(a_7)$  0 $Q_t(a_8)$  0 $Q_t(a_9)$  0 $Q_t(a_{10})$  0

With mean = 0.55

- Let's assume we receive 0.22 as reward.
- Update now the value estimate (see next slide)

 $q * (a_6)$  0.21 $q * (a_7)$  0.80 $q * (a_8)$  1.30 $q * (a_9)$  0.79 $q * (a_{10})$  1.21

BITS Pilani, Hyderabad Campus

## Greedy Approach: Example (t=2)

Value Estimate

	t=1	t=2
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0	0.22
$Q_t(a_4)$	0	0

- $Q_t(a_3)$  is  $0.22/1 = 0.22$  where the number of occurrence of Action 3 is 1.
- At t=2, check the value estimates
- Select Action 3 with maximum

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86

# Greedy Approach. Example (t=1)

innovate

achieve

lead

Value Estimate

	t=1
$Q_t(a_1)$	0
$Q_t(a_2)$	0
$Q_t(a_3)$	0
$Q_t(a_4)$	0
$Q_t(a_5)$	0
$Q_t(a_6)$	0
$Q_t(a_7)$	0
$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

- At t=1, all previous value estimates are zero (initialization)
- Select a random action
- Lets assume we assume action 3
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.22 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0..99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

# Greedy Approach: Example (t=2)



Value Estimate

	t=1	t=2
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0	0.22
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $0.22/1 = 0.22$  where the number of occurrence of Action 3 is 1.
- At t=2, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

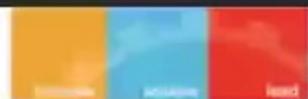
Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_7)$	0
$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

- reward.
- Update now the value estimate (see next slide)

$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=2)

Value Estimate

	t=1	t=2
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0	0.22
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

Expected Reward (Not Known)

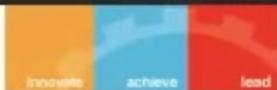
$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

- $Q_t(a_3)$  is  $0.22/1 = 0.22$  where the number of occurrence of Action 3 is 1.
- At t=2, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=3)

Value Estimate

	t=2	t=3
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0.22	.605
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $(0.22+0.99)/2 = 0.605$  where the number of occurrence of Action 3 is 2.
- At t=3, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive -1.38 as reward.
- Update now the value estimate (see next slide)

Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=3)

Value Estimate

	t=2	t=3
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0.22	.605
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $(0.22+0.99)/2 = 0.605$  where the number of occurrence of Action 3 is 2.
- At t=3, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive -1.38 as reward.
- Update now the value estimate (see next slide)

Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0..99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

## Greedy Approach: Example (t=3)



Value Estimate

	t=2	t=3
Q <sub>t</sub> (a <sub>1</sub> )	0	0
Q <sub>t</sub> (a <sub>2</sub> )	0	0
Q <sub>t</sub> (a <sub>3</sub> )	0.22	.605
Q <sub>t</sub> (a <sub>4</sub> )	0	0
Q <sub>t</sub> (a <sub>5</sub> )	0	0
Q <sub>t</sub> (a <sub>6</sub> )	0	0
Q <sub>t</sub> (a <sub>7</sub> )	0	0
Q <sub>t</sub> (a <sub>8</sub> )	0	0
Q <sub>t</sub> (a <sub>9</sub> )	0	0
Q <sub>t</sub> (a <sub>10</sub> )	0	0

- Q<sub>t</sub>(a<sub>3</sub>) is  $(0.22+0.99)/2 = 0.605$  where the number of occurrence of Action 3 is 2.
- At t=3, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive -1.38 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

q * (a <sub>1</sub> )	1.56
q * (a <sub>2</sub> )	-1.33
q * (a <sub>3</sub> )	-0.99
q * (a <sub>4</sub> )	0.86
q * (a <sub>5</sub> )	0.24
q * (a <sub>6</sub> )	0.21
q * (a <sub>7</sub> )	0.80
q * (a <sub>8</sub> )	1.30
q * (a <sub>9</sub> )	0.79
q * (a <sub>10</sub> )	1.21

## Greedy Approach: Example



$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

with mean -0.99

- Let's assume we receive -1.38 as reward.
- Update now the value estimate (see next slide)

$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

## Greedy Approach: Example (t=4)

Value Estimate

	t=3	t=4
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	.605	-0.056
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $(0.22+0.99-1.38)/3 = 0.056$  where the number of occurrence of Action 3 is 3.
- At t=4, check the value estimates
- Action 3 has the minimum reward now
- Other 9 have the same rewards, i.e., 0
- Select a random action out of the remaining 9, lets say we select 4
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive 1.03 as reward.

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=3)

Value Estimate

	t=2	t=3
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0.22	.605
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

- $Q_t(a_3)$  is  $(0.22+0.99)/2 = 0.605$  where the number of occurrence of Action 3 is 2.
- At t=3, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive -1.38 as reward.
- Update now the value estimate (see next slide)

# Greedy Approach: Example (t=1)



Value Estimate

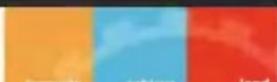
	t=1
$Q_t(a_1)$	0
$Q_t(a_2)$	0
$Q_t(a_3)$	0
$Q_t(a_4)$	0
$Q_t(a_5)$	0
$Q_t(a_6)$	0
$Q_t(a_7)$	0
$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

- At t=1, all previous value estimates are zero (initialization)
- Select a random action
- Lets assume we assume action 3
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.22 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

	Expected Reward
$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

# Greedy Approach: Example (t=2)



$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

- Update now the value estimate (see next slide)

$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

BITS Pilani, Hyderabad Campus



## Greedy Approach: Example (t=2)

Value Estimate

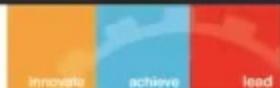
	t=1	t=2
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0	0.22
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $0.22/1 = 0.22$  where the number of occurrence of Action 3 is 1.
- At t=2, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

BITS Pilani, Hyderabad Campus



# Greedy Approach: Example (t=2)

Value Estimate

	t=1	t=2
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0	0.22
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $0.22/1 = 0.22$  where the number of occurrence of Action 3 is 1.
- At t=2, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

# Greedy Approach: Example (t=3)



$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Update now the value estimate (see next slide)

$q^*(a_9)$	0.79
$q^*(a_{10})$	1.21



## Greedy Approach: Example (t=3)

Value Estimate

	t=2	t=3
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0.22	.605
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $(0.22+0.99)/2 = 0.605$  where the number of occurrence of Action 3 is 2.
- At t=3, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive -1.38 as reward.
- Update now the value estimate (see next slide)

Expected Reward (Not Known)

$q^*(a_1)$	1.56
$q^*(a_2)$	-1.33
$q^*(a_3)$	-0.99
$q^*(a_4)$	0.86
$q^*(a_5)$	0.24
$q^*(a_6)$	0.21
$q^*(a_7)$	0.80
$q^*(a_8)$	1.30
$q^*(a_9)$	0.79
$q^*(a_{10})$	1.21

$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

- Update now the value estimate (see next slide)

$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=2)

Value Estimate

	t=1	t=2
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0	0.22
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $0.22/1 = 0.22$  where the number of occurrence of Action 3 is 1.
- At t=2, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=2)

Value Estimate

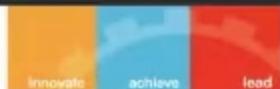
	t=1	t=2
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0	0.22
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $0.22/1 = 0.22$  where the number of occurrence of Action 3 is 1.
- At t=2, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

## Greedy Approach: Example (t=2)

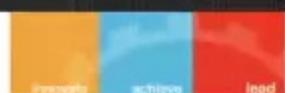


$Q_t(a_3)$	0	0.22
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- At t=2, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.99 as reward.
- Update now the value estimate (see next slide)

$q^*(a_3)$	0.55
$q^*(a_4)$	0.86
$q^*(a_5)$	0.24
$q^*(a_6)$	0.21
$q^*(a_7)$	0.80
$q^*(a_8)$	1.30
$q^*(a_9)$	0.79
$q^*(a_{10})$	1.21

BITS Pilani, Hyderabad Campus



## Greedy Approach: Example (t=3)

Value Estimate

	t=2	t=3
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0.22	.605
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0

- $Q_t(a_3)$  is  $(0.22+0.99)/2 = 0.605$  where the number of occurrence of Action 3 is 2.
- At t=3, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99

Expected Reward  
(Not Known)

$q^*(a_1)$	1.56
$q^*(a_2)$	-1.33
$q^*(a_3)$	-0.99
$q^*(a_4)$	0.86
$q^*(a_5)$	0.24
$q^*(a_6)$	0.21

$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Update now the value estimate (see next slide)

$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=3)

Value Estimate

	t=2	t=3
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	0.22	.605
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $(0.22+0.99)/2 = 0.605$  where the number of occurrence of Action 3 is 2.
- At t=3, check the value estimates
- Select Action 3 with maximum
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive -1.38 as reward.
- Update now the value estimate (see next slide)

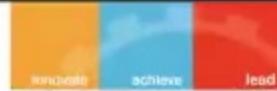
Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Update now the value estimate  
(see next slide)

$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=4)

Value Estimate

	t=3	t=4
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	.605	-0.056
$Q_t(a_4)$	0	0
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_3)$  is  $(0.22+0.99-1.38)/3 = 0.056$  where the number of occurrence of Action 3 is 3.
- At t=4, check the value estimates
- Action 3 has the minimum reward now
- Other 9 have the same rewards, i.e., 0
- Select a random action out of the remaining 9, lets say we select 4
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive 1.03 as reward.
- Update now the value estimate  
(see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_{10})$ 

0

reward.

- Update now the value estimate  
(see next slide)

 $q * (a_{10})$ 

1.21

## Greedy Approach: Example (t=5)

Value Estimate

	t=4	t=5
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-0.056	-0.056
$Q_t(a_4)$	0	1.03
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

Expected Reward  
(Not Known)

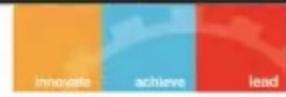
$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

- $Q_t(a_4)$  is  $(1.03)/1 = 1.03$  where the number of occurrence of Action 4 is 1.
- At t=5, check the value estimates
- Select Action 4 with maximum
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive -0.94 as reward.
- Update now the value estimate  
(see next slide)

$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Let's assume we receive -0.94 as reward.
- Update now the value estimate (see next slide)

$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=6)

Value Estimate

	t=5	t=6
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	1.03	.045
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0

- $Q_t(a_4)$  is  $(1.03 - 0.94)/2 = .045$  where the number of occurrence of Action 4 is 2.
- At t=6, check the value estimates
- Select Action 4 with maximum
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive 1.93 as reward.
- Update now the value estimate (see next slide)

Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79

$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Update now the value estimate (see next slide)

$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=6)

Value Estimate

	t=5	t=6
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	1.03	.045
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_4)$  is  $(1.03 - 0.94)/2 = .045$  where the number of occurrence of Action 4 is 2.
- At t=6, check the value estimates
- Select Action 4 with maximum
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive 1.93 as reward.
- Update now the value estimate (see next slide)

	Expected Reward (Not Known)
$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

reward.

- Update now the value estimate  
(see next slide)

$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=7)

Value Estimate

	t=6	t=7
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	.045	0.67
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

- $Q_t(a_4)$  is  $(1.03 - 0.94 + 1.93)/3 = .67$  where the number of occurrence of Action 4 is 3.
- At t=7, check the value estimates
- Select Action 4 with maximum
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive 1.20 as reward.
- Update now the value estimate  
(see next slide)

(t=7)

Value Estimate

	t=6	t=7
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	.045	0.67
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

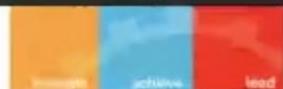
- $Q_t(a_4)$  is  $(1.03 - 0.94 + 1.93)/3 = .67$  where the number of occurrence of Action 4 is 3.
- At t=7, check the value estimates
- Select Action 4 with maximum
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive 1.20 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

BITS Pilani, Hyderabad Campus

## Greedy Approach: Example (t=8)



Value Estimate

	t=7	t=8
$Q_t(a_1)$	0	0

- $Q_t(a_4)$  is  $(1.03 - 0.94 + 1.93 + 1.2)/4$

Expected Reward  
(Not Known)

$q * (a_1)$	1.56

$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

(see next slide)

$q * (a_9)$	0.79
$q * (a_{10})$	1.21

BITS Pilani, Hyderabad Campus



## Greedy Approach: Example (t=8)

Value Estimate

	t=7	t=8
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	0.67	0.805
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_4)$  is  $(1.03 - 0.94 + 1.93 + 1.2) / 4 = .67$  where the number of occurrence of Action 4 is 4.
- Let's stop at t=8 now.

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

BITS Pilani, Hyderabad Campus

$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

• Update now the value estimate  
(see next slide)

$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

BITS Pilani, Hyderabad Campus

## Greedy Approach: Example (t=8)



Value Estimate

	t=7	t=8
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	0.67	0.805
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_4)$  is  $(1.03 - 0.94 + 1.93 + 1.2) / 4 = .67$  where the number of occurrence of Action 4 is 4.
- Let's stop at t=8 now.

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0..99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

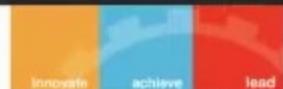
BITS Pilani, Hyderabad Campus

$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- with mean 0.86
- Let's assume we receive 1.20 as reward.
  - Update now the value estimate (see next slide)

$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

BITS Pilani, Hyderabad Campus



## Greedy Approach: Example (t=8)

Value Estimate

	t=7	t=8
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	0.67	0.805
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0

- $Q_t(a_4)$  is  $(1.03-0.94+1.93+1.2)/4 = .67$  where the number of occurrence of Action 4 is 4.
- Let's stop at t=8 now.

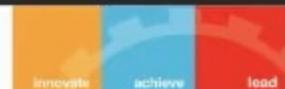
Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79

$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Update now the value estimate  
(see next slide)

$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=8)

Value Estimate

	t=7	t=8
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	0.67	0.805
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_4)$  is  $(1.03-0.94+1.93+1.2)/4 = .67$  where the number of occurrence of Action 4 is 4.
- Let's stop at t=8 now.

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_1)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

reward.

- Update now the value estimate (see next slide)

$q * (a_1)$	0.86
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=6)

Value Estimate

	t=5	t=6
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-0.056	-0.056
$Q_t(a_4)$	1.03	.045
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

- $Q_t(a_4)$  is  $(1.03 - 0.94)/2 = .045$  where the number of occurrence of Action 4 is 2.
- At t=6, check the value estimates
- Select Action 4 with maximum
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive 1.93 as reward.
- Update now the value estimate (see next slide)



## Greedy Approach: Example (t=5)

Value Estimate

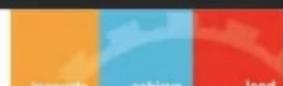
	t=4	t=5
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-0.056	-0.056
$Q_t(a_4)$	0	1.03
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_4)$  is  $(1.03)/1 = 1.03$  where the number of occurrence of Action 4 is 1.
- At t=5, check the value estimates
- Select Action 4 with maximum
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive -0.94 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

## Greedy Approach: Example (t=6)





## Greedy Approach: Example (t=5)

Value Estimate

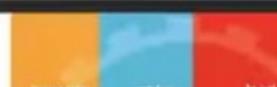
	t=4	t=5
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-0.056	-0.056
$Q_t(a_4)$	0	1.03
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_4)$  is  $(1.03)/1 = 1.03$  where the number of occurrence of Action 4 is 1.
- At t=5, check the value estimates
- Select Action 4 with maximum
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive -0.94 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

## Greedy Approach: Example (t=6)



## Greedy Approach: Example (t=7)



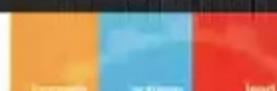
Value Estimate

	t=6	t=7
Q <sub>t</sub> (a <sub>1</sub> )	0	0
Q <sub>t</sub> (a <sub>2</sub> )	0	0
Q <sub>t</sub> (a <sub>3</sub> )	-.056	-.056
Q <sub>t</sub> (a <sub>4</sub> )	.045	0.67
Q <sub>t</sub> (a <sub>5</sub> )	0	0
Q <sub>t</sub> (a <sub>6</sub> )	0	0
Q <sub>t</sub> (a <sub>7</sub> )	0	0
Q <sub>t</sub> (a <sub>8</sub> )	0	0
Q <sub>t</sub> (a <sub>9</sub> )	0	0
Q <sub>t</sub> (a <sub>10</sub> )	0	0

- Q<sub>t</sub>(a<sub>4</sub>) is (1.03 - 0.94 + 1.93) / 3 = .67 where the number of occurrence of Action 4 is 3.
- At t=7, check the value estimates
- Select Action 4 with maximum
- A reward will be a random value taken from a random variable with mean 0.86
- Let's assume we receive 1.20 as reward.
- Update now the value estimate (see next slide)

	Expected Reward (Not Known)
q * (a <sub>1</sub> )	1.56
q * (a <sub>2</sub> )	-1.33
q * (a <sub>3</sub> )	-0.99
q * (a <sub>4</sub> )	0.86
q * (a <sub>5</sub> )	0.24
q * (a <sub>6</sub> )	0.21
q * (a <sub>7</sub> )	0.80
q * (a <sub>8</sub> )	1.30
q * (a <sub>9</sub> )	0.79
q * (a <sub>10</sub> )	1.21

## Greedy Approach: Example





## Greedy Approach: Example (t=8)

Value Estimate

	t=7	t=8
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	0.67	0.805
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- $Q_t(a_4)$  is  $(1.03 - 0.94 + 1.93 + 1.2) / 4 = .67$  where the number of occurrence of Action 4 is 4.
- Let's stop at t=8 now.

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=8)

Value Estimate

	t=8
$Q_t(a_1)$	0
$Q_t(a_2)$	0
$Q_t(a_3)$	-0.056
$Q_t(a_4)$	0.805
$Q_t(a_5)$	0
$Q_t(a_6)$	0
$Q_t(a_7)$	0
$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

### Key takeaways after t=8:

- The initial selected action (Action 3) is no more the best action
- We are stuck with Action 4 !!
- Action 4 is suboptimal (Best is Action 1)
- We never arrived at Action 1
- $Q_8(a_4) = 0.805$  starts getting converging to  $q * (a_4) = 0.86$  meaning that we may stuck with Action 4 forever since the estimate might not go less than 0.

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_9)$  0 0 $q^*(a_9)$  0.79 $Q_t(a_{10})$  0 0 $q^*(a_{10})$  1.21

## Greedy Approach: Example (t=8)

Value Estimate

	t=8
$Q_t(a_1)$	0
$Q_t(a_2)$	0
$Q_t(a_3)$	-0.056
$Q_t(a_4)$	0.805
$Q_t(a_5)$	0
$Q_t(a_6)$	0
$Q_t(a_7)$	0
$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

### Key takeaways after t=8:

- The initial selected action (Action 3) is no more the best action
- We are stuck with Action 4 !!
- Action 4 is suboptimal (Best is Action 1)
- We never arrived at Action 1
- $Q_8(a_4) = 0.805$  starts getting converging to  $q^*(a_4) = 0.86$  meaning that we may stuck with Action 4 forever since the estimate might not go less than 0.

Expected Reward  
(Not Known)

$q^*(a_1)$	1.56
$q^*(a_2)$	-1.33
$q^*(a_3)$	-0.99
$q^*(a_4)$	0.86
$q^*(a_5)$	0.24
$q^*(a_6)$	0.21
$q^*(a_7)$	0.80
$q^*(a_8)$	1.30
$q^*(a_9)$	0.79
$q^*(a_{10})$	1.21

$Q_t(a_9)$  0

$Q_t(a_{10})$  0

estimate might not go less than 0.

$q * (a_9)$  0.79

$q * (a_{10})$  1.21

BITS Pilani, Hyderabad Campus



## $\epsilon$ -Greedy Approach

- Be greedy most of the time
- But select a random action for  $\epsilon$  fraction of the time, i.e., at each step select a random action with probability  $\epsilon$ .
- Now let's check again the previous example with this new approach.

BITS Pilani, Hyderabad Campus



## $\epsilon$ -Greedy Approach: Example (t=8)

Value Estimate

	t=8
$Q_t(a_1)$	0
$Q_t(a_2)$	0
$Q_t(a_3)$	-.056
$Q_t(a_4)$	0.805
$Q_t(a_5)$	0
$Q_t(a_6)$	0
$Q_t(a_7)$	0
$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

Assume that  $\epsilon = 1/9$  and till t=8, we are being greedy.

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

# $\epsilon$ -Greedy Approach: Example (t=8)

innovate

achieve

lead

Value Estimate

	t=8
$Q_t(a_1)$	0
$Q_t(a_2)$	0
$Q_t(a_3)$	-0.056
$Q_t(a_4)$	0.805
$Q_t(a_5)$	0
$Q_t(a_6)$	0
$Q_t(a_7)$	0
$Q_t(a_8)$	0
$Q_t(a_9)$	0
$Q_t(a_{10})$	0

Assume that  $\epsilon = 1/9$  and till t=8, we are being greedy.

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

BITS Pilani, Hyderabad Campus

# $\epsilon$ -Greedy Approach: Example (t=9)

innovate

achieve

lead

Value Estimate

	t=8	t=9
--	-----	-----

Expected Reward  
(Not Known)

$Q_t(a_9)$	0	$q * (a_9)$	0.79
$Q_t(a_{10})$	0	$q * (a_{10})$	1.21

## $\epsilon$ -Greedy Approach: Example (t=9)



Value Estimate

	t=8	t=9
$Q_t(a_1)$	0	0
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-.056	-.056
$Q_t(a_4)$	0.805	0.805
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

- Let's be non-greedy and choose a random action
- Just by chance, assume that you choose Action 1.
- A reward will be a random value taken from a random variable with mean 1.56
- Let's assume we receive 1.4 as reward.
- Update now the value estimate (see next slide)

	Expected Reward (Not Known)
$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## $\epsilon$ -Greedy Approach: Example (t=10)

Value Estimate

	t=9	t=10
$Q_t(a_1)$	0	1.4
$Q_t(a_2)$	0	0
$Q_t(a_3)$	-0.056	-0.056
$Q_t(a_4)$	0.805	0.805
$Q_t(a_5)$	0	0
$Q_t(a_6)$	0	0
$Q_t(a_7)$	0	0
$Q_t(a_8)$	0	0
$Q_t(a_9)$	0	0
$Q_t(a_{10})$	0	0

Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

- $Q_t(a_1)$  is  $(1.4)/1 = 1.4$  where the number of occurrence of Action 1 is 1.
- and so on !!



## $\epsilon$ -Greedy Approach

- Be greedy most of the time
- But select a random action for  $\epsilon$  fraction of the time, i.e., at each step select a random action with probability  $\epsilon$ .
- Now let's check again the previous example with this new approach.

□



$\epsilon$ -Greedy Approach – Can we FORCE initial exploration without changing Epsilon Value ?

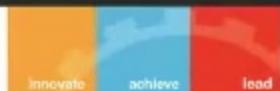
5

BITS Pilani, Hyderabad Campus

**Greedy Approach:**  
**Example with a different initial values**



Expected Reward



## Greedy Approach: Example with a different initial values

Value Estimate

	t=1
$Q_t(a_1)$	5
$Q_t(a_2)$	5
$Q_t(a_3)$	5
$Q_t(a_4)$	5
$Q_t(a_5)$	5
$Q_t(a_6)$	5
$Q_t(a_7)$	5
$Q_t(a_8)$	5
$Q_t(a_9)$	5
$Q_t(a_{10})$	5

Expected Reward  
(Not Known)

	Expected Reward
$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_8)$	5
$Q_t(a_9)$	5
$Q_t(a_{10})$	5

$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=1) - initial values

Value Estimate

	t=1
$Q_t(a_1)$	5
$Q_t(a_2)$	5
$Q_t(a_3)$	5
$Q_t(a_4)$	5
$Q_t(a_5)$	5
$Q_t(a_6)$	5
$Q_t(a_7)$	5
$Q_t(a_8)$	5
$Q_t(a_9)$	5
$Q_t(a_{10})$	5

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

- At t=1, all previous value estimates are zero (initialization)
- Select a random action
- Lets assume we assume action 3
- A reward will be a random value taken from a random variable with mean -0.99
- Let's assume we receive 0.22 as reward.
- Update now the value estimate (see next slide)

$Q_t(a_9)$  5 $Q_t(a_{10})$  5

(see next slide)

 $q * (a_9)$  0.79 $q * (a_{10})$  1.21

## Greedy Approach: Example (t=2) - initial values

Value Estimate

	t=1	t=2
$Q_t(a_1)$	5	5
$Q_t(a_2)$	5	5
$Q_t(a_3)$	5	0.22
$Q_t(a_4)$	5	5
$Q_t(a_5)$	5	5
$Q_t(a_6)$	5	5
$Q_t(a_7)$	5	5
$Q_t(a_8)$	5	5
$Q_t(a_9)$	5	5
$Q_t(a_{10})$	5	5

- $Q_t(a_3)$  is  $0.22/1 = 0.22$  where the number of occurrence of Action 3 is 1.
- At t=2, check the value estimates
- Action 3 is minimum. Other have same. Select randomly one from others.
- Lets assume we select 1.
- A reward will be a random value taken from a random variable with mean 1.56
- Let's assume we receive 1.2 as reward.
- Update now the value estimate (see next slide)

Expected Reward  
(Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_8)$	5	5
$Q_t(a_9)$	5	5
$Q_t(a_{10})$	5	5

- Let's assume we receive 1.2 as reward.
- Update now the value estimate (see next slide)

$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=3) - initial values

Value Estimate

	t=2	t=3
$Q_t(a_1)$	5	1.2
$Q_t(a_2)$	5	5
$Q_t(a_3)$	5	0.22
$Q_t(a_4)$	5	5
$Q_t(a_5)$	5	5
$Q_t(a_6)$	5	5
$Q_t(a_7)$	5	5
$Q_t(a_8)$	5	5
$Q_t(a_9)$	5	5
$Q_t(a_{10})$	5	5

- $Q_t(a_1)$  is  $1.2/1 = 1.2$  where the number of occurrence of Action 1 is 1.
- At t=3, check the value estimates
- Action 1 and 3 have lower values. Other have same. Select randomly one from others.
- Lets assume we select 10.
- A reward will be a random value taken from a random variable with mean 1.21
- Let's assume we receive 1.4 as reward.
- Update now the value estimate (see next slide)

Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_9)$	5	5
$Q_t(a_{10})$	5	5

- Update now the value estimate (see next slide)

$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=4) - initial values

Value Estimate

	t=3	t=4
$Q_t(a_1)$	5	1.2
$Q_t(a_2)$	5	5
$Q_t(a_3)$	5	0.22
$Q_t(a_4)$	5	5
$Q_t(a_5)$	5	5
$Q_t(a_6)$	5	5
$Q_t(a_7)$	5	5
$Q_t(a_8)$	5	5
$Q_t(a_9)$	5	5
$Q_t(a_{10})$	5	1.4

- $Q_t(a_{10})$  is  $1.4/1 = 1.4$  where the number of occurrence of Action 10 is 1.
- At t=4, check the value estimates
- Action 1, 3 and 10 have lower values. Other have same. Select randomly one from others.
- Lets assume we select 5.
- A reward will be a random value taken from a random variable with mean 0.24
- Let's assume we receive .11 as reward.
- Update now the value estimate (see next slide)

Expected Reward (Not Known)

$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

$Q_t(a_8)$	5	5
$Q_t(a_9)$	5	5
$Q_t(a_{10})$	5	1.4

- Let's assume we receive .11 as reward.
- Update now the value estimate (see next slide)

$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21



## Greedy Approach: Example (t=5)

Value Estimate

	t=3	t=4
$Q_t(a_1)$	5	1.2
$Q_t(a_2)$	5	5
$Q_t(a_3)$	5	0.22
$Q_t(a_4)$	5	5
$Q_t(a_5)$	5	0.11
$Q_t(a_6)$	5	5
$Q_t(a_7)$	5	5
$Q_t(a_8)$	5	5
$Q_t(a_9)$	5	5
$Q_t(a_{10})$	5	1.4

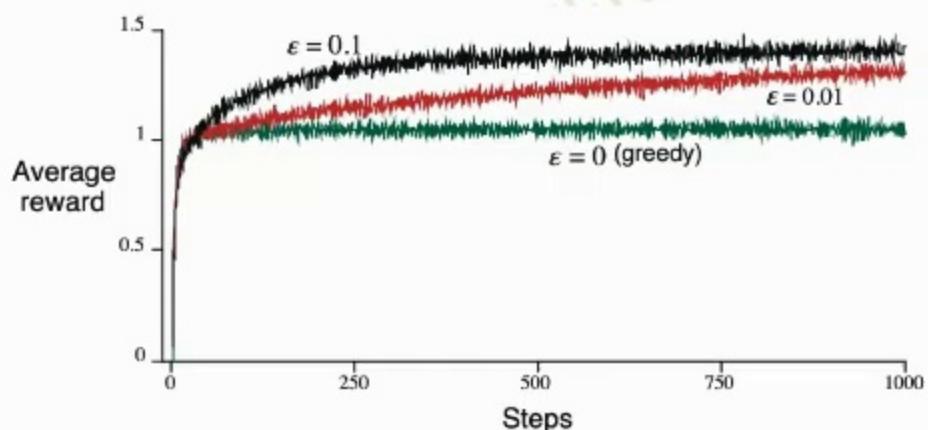
Key Points:

- At each timestamp a separate action is being taken
- High Exploration
- By the end of 10<sup>th</sup> time stamp, most likely all the actions are explored

Expected Reward  
(Not Known)

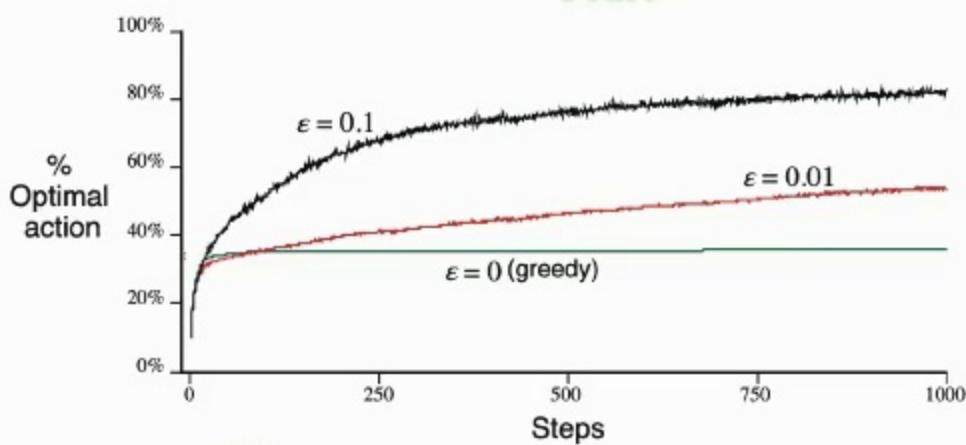
$q * (a_1)$	1.56
$q * (a_2)$	-1.33
$q * (a_3)$	-0.99
$q * (a_4)$	0.86
$q * (a_5)$	0.24
$q * (a_6)$	0.21
$q * (a_7)$	0.80
$q * (a_8)$	1.30
$q * (a_9)$	0.79
$q * (a_{10})$	1.21

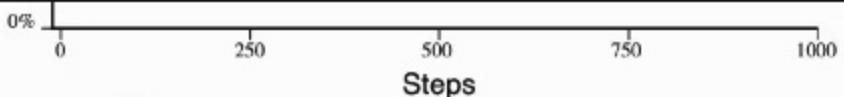
## Results from the Book - I





## Results from the Book - II



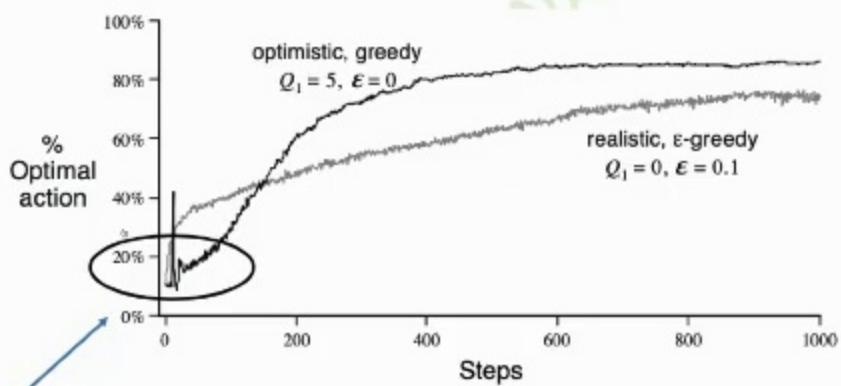


PSA

BITS Pilani, Hyderabad Campus



## Results from the Book (with high Initial Values)



Why Oscillations ?

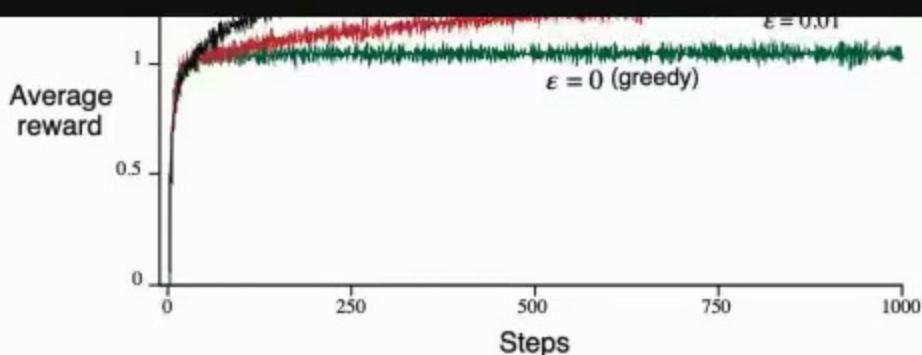
- All actions are explored even few times
- Results in oscillations till overall reward stabilizes.



## Other Related Topics

---

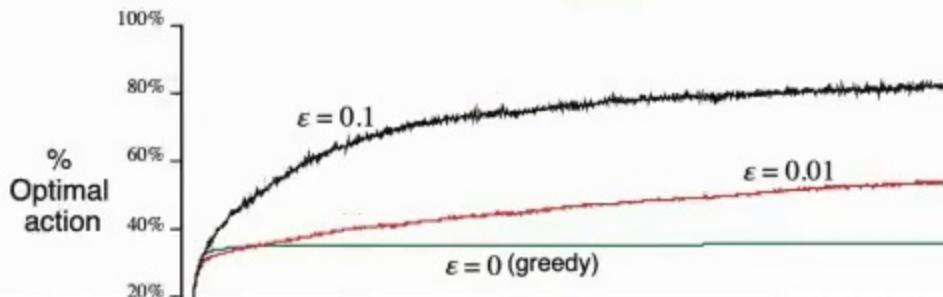
- Upper-Confidence-Bound Action Selection
  - Reduce Exploration Over Time
  - Refer to text book for more details
- Non-Stationary Bandits
  - Expected value of rewards vary over time
  - Constant Step size and weighted average



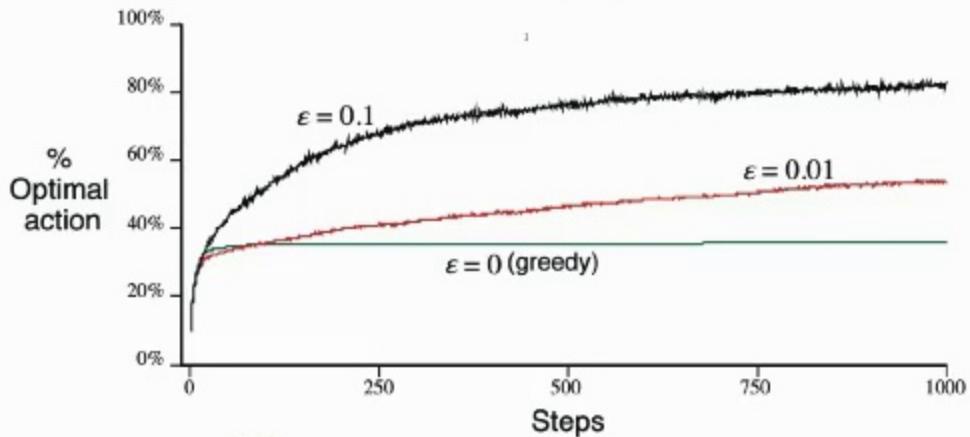
BITS Pilani, Hyderabad Campus



## Results from the Book - II



## Results from the Book - II





meet.google.com/brc-ssbd-pz?pli=1&authuser=1

(ct) - nida... Gmail YouTube Maps YouTube bitspilani

You have extensions installed that may affect the quality of your call

You're presenting to everyone

The screenshot shows a Google Meet session with nine participants. On the left, a dark overlay displays a presentation slide with a grid of icons and text. A blue button at the bottom of this overlay says "Stop presenting". Below it is an "Ignore" button. To the right of the presentation area, participant cards are arranged in a grid:

- Row 1: paresh saxena (profile picture), Aditya Chopra (profile picture), BHARAT (profile picture)
- Row 2: SHUBHAM PRIYANK (profile picture), ATISHAY JAIN (profile picture), SHREYA (profile picture)
- Row 3: NEIL PARESH MEHTA (profile picture), KENUTE (profile picture), RUBAN S (profile picture)

Each participant card includes a microphone icon and a video camera icon. In the bottom right corner of the participant area, there are two messages from users: "yes" and "yes ma'am".

ring, don't share your entire screen or browser window. Share just a  
ent window instead.

Stop presenting

Ignore

meet.google.com is sharing your screen.

Stop sharing

Hide

ssbd-pz]

to search



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | +

lab.research.google.com/drive/1whm6Rn07\_y5HpzbKrNYiWivQSfjLDg#scrollTo=016Nlk6zoc5v2

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb

View Insert Runtime Tools Help Last saved at 11:11 AM

ext

Comment

RAM Disk

## Multi-Armed Bandits with Epsilon Greedy Action Selection

### Summary of the concepts learnt:

**Multi-Armed bandit:** An agent chooses between k different actions and receives a reward based on the chosen action. The goal of the agent is to identify which action to choose to get the maximum reward.

**Exploration:** An Agent learns more about the environment.

**Exploitation:** An Agent uses current knowledge to obtain the best results.

**Exploration-Exploitation Dilemma:** Exploration sacrifice the chance of getting a known reward. Exploitation risks acting sub-optimally by not exploring enough.

**Epsilon Greedy Action Selection:** It is method to balance exploration and exploitation.

Epsilon refers to the probability of choosing to explore. The value for epsilon is selected in between 0 and 1. If epsilon is 0, then it always exploits. If epsilon is 1, then it always explores. If epsilon is 0.1, then 10% of the time it explores and 90% it exploits.

## Greedy Action Selection

Install the libraries

meet.google.com is sharing your screen. Stop sharing Hide

to search



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | +

lab.research.google.com/drive/1whm6Rn07\_ySHpzbKrNYIiWivQSfjLDg#scrollTo=LLAGhdtWcfc-

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb

View Insert Runtime Tools Help Last saved at 11:11 AM

RAM Disk

## Summary of the concepts learnt:

**Multi-Armed Bandit:** An agent chooses between k different actions and receives a reward based on the chosen action. The goal of the agent is to identify which action to choose to get the maximum reward.

**Exploration:** An Agent learns more about the environment.

**Exploitation:** An Agent uses current knowledge to obtain the best results.

**Exploration-Exploitation Dilemma:** Exploration sacrifice the chance of getting a known reward. Exploitation risks acting sub-optimally by not exploring enough.

**Epsilon Greedy Action Selection:** It is method to balance exploration and exploitation.

Epsilon refers to the probability of choosing to explore. The value for epsilon is selected in between 0 and 1. If epsilon is 0, then it always exploits. If epsilon is 1, then it always explores. If epsilon is 0.1, then 10% of the time it explores and 90% it exploits.

## Greedy Action Selection

### Importing the libraries

```
numpy as np  
matplotlib.pyplot as plt  
otlib inline
```

meet.google.com is sharing your screen. Stop sharing Hide

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | +

lab.research.google.com/drive/1whm6Rn07\_ySHpzbKrNYIiWivQSfjLDg#scrollTo=LLAGhdtWcfc-

(ect) - nida... Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help Last saved at 11:11 AM

RAM Disk

**Armed bandit:** An agent chooses between k different actions and receives a reward based on the chosen action. The goal of the is to identify which action to choose to get the maximum reward.

**Exploration:** An Agent learns more about the environment.

**Exploitation:** An Agent uses current knowledge to obtain the best results.

**Exploration-Exploitation Dilemma:** Exploration sacrifice the chance of getting a known reward. Exploitation risks acting sub-optimally by knowing enough.

**Epsilon Greedy Action Selection:** It is method to balance exploration and exploitation.

Epsilon refers to the probability of choosing to explore. The value for epsilon is selected in between 0 and 1. If epsilon is 0, then it always exploits. If epsilon is 1, then it always explores. If epsilon is 0.1, then 10% of the time it explores and 90% it exploits.

## Greedy Action Selection

ng the libraries

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

meet.google.com is sharing your screen. Stop sharing Hide

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_ySHpzbKrNYjIWivQSfjLDg#scrollTo=LLAGhdtWcfc-

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb

View Insert Runtime Tools Help Last saved at 11:11 AM

RAM Disk

## of the concepts learnt:

**Armed bandit:** An agent chooses between  $k$  different actions and receives a reward based on the chosen action. The goal of the agent is to identify which action to choose to get the maximum reward.

**Exploration:** An Agent learns more about the environment.

**Exploitation:** An Agent uses current knowledge to obtain the best results.

**Exploration-Exploitation Dilemma:** Exploration sacrifice the chance of getting a known reward. Exploitation risks acting sub-optimally by not exploring enough.

**Epsilon Greedy Action Selection:** It is method to balance exploration and exploitation.

Epsilon refers to the probability of choosing to explore. The value for epsilon is selected in between 0 and 1. If epsilon is 0, then it always exploits. If epsilon is 1, then it always explores. If epsilon is 0.1, then 10% of the time it explores and 90% it exploits.

## Greedy Action Selection

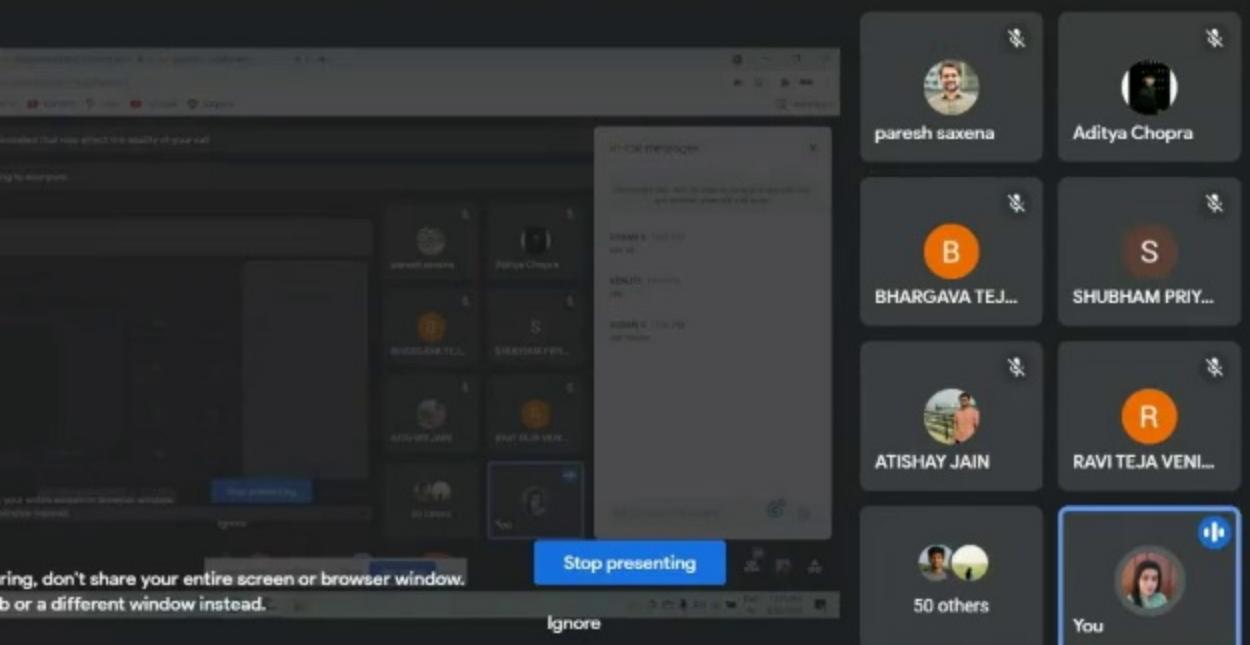
ng the libraries

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

meet.google.com is sharing your screen. Stop sharing Hide

have extensions installed that may affect the quality of your call.

You're presenting to everyone



## In-call messages

Messages can only be seen by people who are deleted when the owner

RUBAN S 12:03 PM  
yes sir

KENUTE 12:06 PM  
yes

RUBAN S 12:06 PM  
yes ma'am

Send a message to everyone

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | +

lab.research.google.com/drive/1whm6Rn07\_y5HpbKtNYIiWivQSfjLDg#scrollTo=LLAGhdtWcfc-

Gmail YouTube Maps YouTube bitsplani

MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help Last saved at 11:11 AM

RAM Disk

## Summary of the concepts learnt:

**Multi-Armed Bandit:** An agent chooses between k different actions and receives a reward based on the chosen action. The goal of the agent is to identify which action to choose to get the maximum reward.

**Exploration:** An Agent learns more about the environment.

**Exploitation:** An Agent uses current knowledge to obtain the best results.

**Exploration-Exploitation Dilemma:** Exploration sacrifice the chance of getting a known reward. Exploitation risks acting sub-optimally by not exploring enough.

**Epsilon Greedy Action Selection:** It is method to balance exploration and exploitation.

Epsilon refers to the probability of choosing to explore. The value for epsilon is selected in between 0 and 1. If epsilon is 0, then it always exploits. If epsilon is 1, then it always explores. If epsilon is 0.1, then 10% of the time it explores and 90% it exploits.

## Greedy Action Selection

Importing the libraries

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

meet.google.com is sharing your screen. Stop sharing Hide

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x +

lab.research.google.com/drive/1iGICF0zNQF\_ayF\_j8\_6-2Xu4Axz-BXXR#scrollTo=R1qc-8-xrf5I

Gmail YouTube Maps YouTube bitspilani

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

## Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

## Plotting the plots

```
def plot_history(history):
    rewards = history["rewards"]
    cum_rewards = history["cum_rewards"]
    chosen_arms = history["arms"]

    plt.figure(figsize=[30,8])

    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cumulative Rewards")

    fig.add_subplot(122)
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
    set_title("Chosen Actions")
```

## Environment

meet.google.com is sharing your screen. Stop sharing Hide



## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help Last saved at 11:11 AM

Comment



RAM Disk

### Summary of the concepts learnt:

**Multi-Armed Bandit:** An agent chooses between k different actions and receives a reward based on the chosen action. The goal of the agent is to identify which action to choose to get the maximum reward.

**Exploration:** An Agent learns more about the environment.

**Exploitation:** An Agent uses current knowledge to obtain the best results.

**Exploration-Exploitation Dilemma:** Exploration sacrifice the chance of getting a known reward. Exploitation risks acting sub-optimally by not exploring enough.

**Epsilon Greedy Action Selection:** It is method to balance exploration and exploitation.

Epsilon refers to the probability of choosing to explore. The value for epsilon is selected in between 0 and 1. If epsilon is 0, then it always exploits. If epsilon is 1, then it always explores. If epsilon is 0.1, then 10% of the time it explores and 90% it exploits.

## Greedy Action Selection

Importing the libraries

```
numpy as np  
matplotlib.pyplot as plt  
otlib inline
```

meet.google.com is sharing your screen. Stop sharing Hide

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x +

lab.research.google.com/drive/1IGICF0zNQF\_ayF\_j8\_6-2Xu4Axz-BXXR#scrollTo=R1qc-8-xrf51

ect) - nida... Gmail YouTube Maps YouTube bitspilani

All changes saved

Comment

RAM Disk

ating the environment

ing a selective action

ng Exploration and Exploitation with epsilon greedy algorithm

enting Epsilon greedy algorithm

g plot for Epsilon greedy algorithm

meet.google.com is sharing your screen. Stop sharing Hide

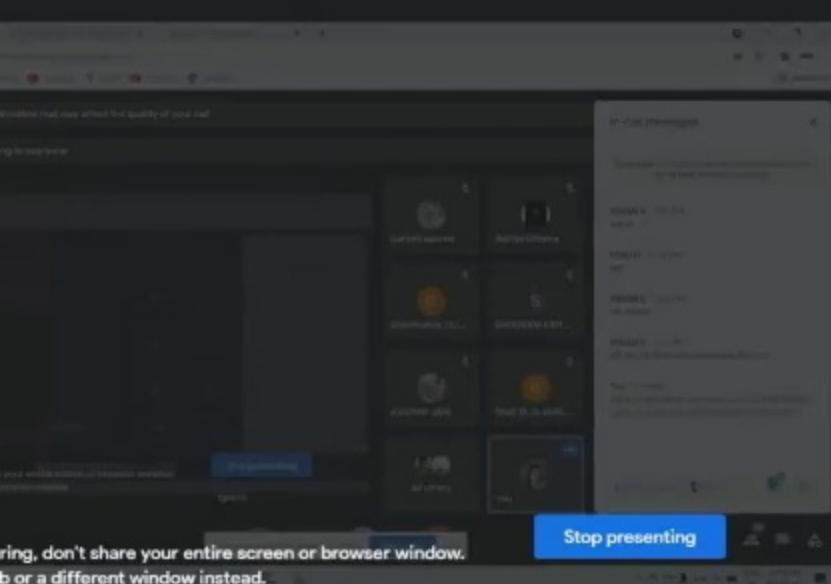
to search

meet.google.com/brc-ssbd-pzj?pli=1&authuser=1

Gmail YouTube Maps YouTube bitspilani

You have extensions installed that may affect the quality of your call

You're presenting to everyone



### In-call messages

Messages can only be seen by people in the call. They are deleted when the call ends.

RUBAN S 12:03 PM  
yes sir

KENUTE 12:06 PM  
yes

RUBAN S 12:06 PM  
yes ma'am

RUBAN S 12:15 PM  
will you be sharing this notebook with me

You 12:18 PM  
[https://colab.research.google.com/F\\_ayF\\_I8\\_6-2Xu4Axz-BXXR#scrollTo](https://colab.research.google.com/F_ayF_I8_6-2Xu4Axz-BXXR#scrollTo)

Send a message to everyone

bcd-pzj

MultiArmedBandit\_EpsilonGreedy

Lesson1 - Colaboratory

lab.research.google.com/drive/1iGICF0zNQF\_ayF\_j8\_6-2Xu4Axz-BXXR#scrollTo=R1qc-8-xf5I

nida... Gmail YouTube Maps YouTube bitspilani

All changes saved

RAM Disk

```
ext
plt.figure(figsize=[30,8])
fig.add_subplot(121)
plot(cum_rewards, label="avg rewards")
set_title("Cumulative Rewards")

fig.add_subplot(122)
bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
set_title("Chosen Actions")
```

## Environment

ating the environment

ing a selective action

meet.google.com is sharing your screen. Stop sharing Hide

of Lesson1 ☆

File Insert Runtime Tools Help Last saved at 12:19 PM

Comment



ext

Connect



## Importing the libraries

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

## Plotting the plots

```
def plot_history(history):
    rewards = history["rewards"]
    cum_rewards = history["cum_rewards"]
    chosen_arms = history["arms"]

    fig = plt.figure(figsize=[30,8])
    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cumulative Rewards")

    fig.add_subplot(122)
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
    set_title("Chosen Actions")
```

Environment

meet.google.com is sharing your screen. Stop sharing Hide

to search



### **Using the libraries**

```
numpy as np  
matplotlib.pyplot as plt  
%matplotlib inline
```

the plots

```

ot_history(history):
rds = history["rewards"]
rewards = history["cum_rewards"]
en_arms = history["arms"]

= plt.figure(figsize=[30,8])

= fig.add_subplot(121)
plot(cum_rewards, label="avg rewards")
set_title("Cumulative Rewards")

= fig.add_subplot(122)
bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
set_title("Chosen Actions")

```

## Environment

meet.google.com is sharing your screen.

## of Lesson1

Comment

File Insert Runtime Tools Help Last saved at 12:19 PM

```
ext
cum_rewards = history["cum_rewards"]
chosen_arms = history["arms"]

plt.figure(figsize=[30,8])

fig.add_subplot(121)
plot(cum_rewards, label="avg rewards")
set_title("Cumulative Rewards")

fig.add_subplot(122)
bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
set_title("Chosen Actions")
```

## Environment

ating the environment

ing a selective action

meet.google.com is sharing your screen. Stop sharing Hide

MultiArmedBandit\_EpsilonGreedy.ipynb

Lesson1 - Colaboratory

Copy of Lesson1 - Colaboratory

MultiArmedBandit\_EpsilonGreedy.ipynb

- Colaboratory

colab.research.google.com

of Lesson1

View Insert

ext

ng the libraries

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

ng the plots

```
def plot_history(history):
    rewards = history["rewards"]
    cum_rewards = history["cum_rewards"]
    chosen_arms = history["arms"]

    fig = plt.figure(figsize=[30,8])

    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cumulative Rewards")

    fig.add_subplot(122)
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
    set_title("Chosen Actions")
```

Environment

meet.google.com is sharing your screen.

Stop sharing Hide

o H e A G F

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1iGICF0zNQF\_ayFj8\_6-2Xu4Axz-BXXR#scrollTo=R1qc-8-xrf5I

Gmail YouTube Maps YouTube bitspilani

1 ★

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

## Importing the libraries

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

## Plotting the plots

```
def plot_history(history):
    rewards = history["rewards"]
    cum_rewards = history["cum_rewards"]
    chosen_arms = history["arms"]

    plt.figure(figsize=[30,8])

    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cumulative Rewards")

    fig.add_subplot(122)
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
    set_title("Chosen Actions")
```

## Environment

meet.google.com is sharing your screen. Stop sharing Hide

meet.google.com/birc-ssbd-pzj?pli=1&authuser=1

Lesson1 - Colaboratory

Copy of Lesson1 - Colaboratory

You're presenting to everyone

In-call messages

Aditya Chopra 12:19 PM  
++ in share you can turn off the edit would have to copy it to make edits

RUBAN S 12:19 PM  
there's a way to make the notebook think

BHARGAVA TEJA U 12:19 PM  
it will still be viewable but not edita viewable\*

SANATH SALIL 12:20 PM  
you can restrict editing and allow v

Samarth Soni 12:20 PM  
ma'am click on share and over there the option

SANATH SALIL 12:20 PM  
by sharing a link specific to viewing

Stop presenting

Ignore

Stop sharing Hide

meet.google.com is sharing your screen.

48 others

You

Send a message to everyone

meet.google.com/birc-ssbd-pzj?pli=1&authuser=1

Lesson1 - Colaboratory

colab.research.google.com

have extensions installed that may affect the quality

You're presenting to everyone

In-call messages

Aditya Chopra 12:19 PM  
++ in share you can turn off the edit would have to copy it to make edits

RUBAN S 12:19 PM  
there's a way to make the notebook think

BHARGAVA TEJA U 12:19 PM  
it will still be viewable but not edita viewable\*

SANATH SALIL 12:20 PM  
you can restrict editing and allow v

Samarth Soni 12:20 PM  
ma'am click on share and over there the option

SANATH SALIL 12:20 PM  
by sharing a link specific to viewing

Stop presenting

Ignore

Stop sharing Hide

meet.google.com is sharing your screen.

48 others

You

Send a message to everyone

meet.google.com/birc-ssbd-pzj?pli=1&authuser=1

Lesson1 - Colaboratory

colab.research.google.com

have extensions installed that may affect the quality

You're presenting to everyone

In-call messages

Aditya Chopra 12:19 PM  
++ in share you can turn off the edit would have to copy it to make edits

RUBAN S 12:19 PM  
there's a way to make the notebook think

BHARGAVA TEJA U 12:19 PM  
it will still be viewable but not edita viewable\*

SANATH SALIL 12:20 PM  
you can restrict editing and allow v

Samarth Soni 12:20 PM  
ma'am click on share and over there the option

SANATH SALIL 12:20 PM  
by sharing a link specific to viewing

Stop presenting

Ignore

Stop sharing Hide

meet.google.com is sharing your screen.

48 others

You

Send a message to everyone

Share with people and groups

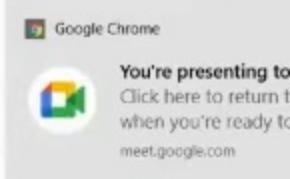
Add people and groups

**Nida** Nida Fatima (you)  
nidafatima127@gmail.com Owner

[Send feedback to Google](#) [Done](#)

Get link

Anyone on the internet with this link can view [Change](#) [Copy link](#)



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1lGICF0zNQF\_ayF\_j8\_6-2Xu4Axz-BXXR#scrollTo=R1qc-8-xrf5I

nidaf... Gmail YouTube Maps YouTube bitspilani

All changes saved

Comment RAM Disk

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

the plots

```
def plot_history(history):
    rewards = history["rewards"]
    cum_rewards = history["cum_rewards"]
    chosen_arms = history["arms"]

    plt.figure(figsize=[30,8])

    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cummulative Rewards")

    fig.add_subplot(122)
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
    set_title("Chosen Actions")
```

Environment

meet.google.com is sharing your screen. Stop sharing Hide

to search

meet.google.com/brc-ssbd-pzj?pli=1&authuser=1

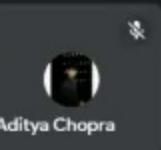
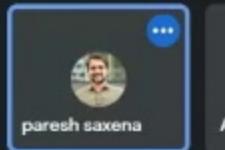
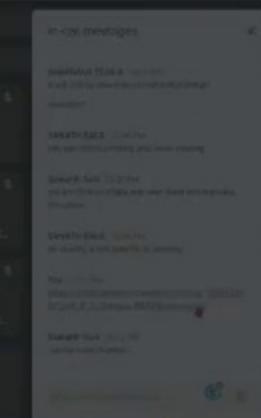
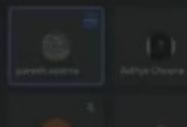
Gmail YouTube Maps YouTube bitspilani

You have extensions installed that may affect the quality of your call

You're presenting to everyone

extensions that may affect the quality of your call

ING TO EVERYONE



### In-call messages

BHARGAVA TEJA U 12:19 PM  
it will still be viewable but not edita  
viewable\*

SANATH SALIL 12:20 PM  
you can restrict editing and allow v

Samarth Soni 12:20 PM  
ma'am click on share and over ther  
the option

SANATH SALIL 12:20 PM  
by sharing a link specific to viewing

You 12:21 PM  
[https://colab.research.google.com/QF\\_ayF\\_iB\\_6-2Xu4Axz-BXXR?usp=sharing](https://colab.research.google.com/QF_ayF_iB_6-2Xu4Axz-BXXR?usp=sharing)

Samarth Soni 12:21 PM  
cannot save changes

Send a message to everyone

ring, don't share your entire screen or browser window.  
b or a different window instead.

Stop presenting

Ignore

meet.google.com is sharing your screen.

Stop sharing

Hide

ebd-pz]

e.com/dmve/1iGICFOzNQF\_myF\_iB\_6-2Xu4Axz-BXXR?usp=sharing...

to search



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1iGICF0zNQF\_ayFjB\_6-2Xu4Axz-BXXR#scrollTo=R1qc-8-xrf5I

Gmail YouTube Maps YouTube bitspilani

File Insert Runtime Tools Help All changes saved

Comment RAM Disk

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

the plots

```
def plot_history(history):
    rewards = history["rewards"]
    cum_rewards = history["cum_rewards"]
    chosen_arms = history["arms"]

    plt.figure(figsize=[30,8])

    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cummulative Rewards")

    fig.add_subplot(122)
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
    set_title("Chosen Actions")
```

Environment

meet.google.com is sharing your screen. Stop sharing Hide

Google Chrome

You're presenting to **meet.google.com**. Click here to return to the presentation when you're ready to meet.google.com

MultiArmedBandit\_EpsilonGreedy | Lesson1 - Colaboratory | Copy of Lesson1 - Colaboratory +

lab.research.google.com/drive/1zURgoSloY-yTBEgWTqluz1RxKLdhuc5C

Gmail YouTube Maps YouTube bitspilani

## of Lesson1

View Insert Runtime Tools Help Last saved at 12:19 PM

Comment Connect

### Importing the libraries

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

### Plotting the plots

```
def plot_history(history):
    rewards = history["rewards"]
    cum_rewards = history["cum_rewards"]
    chosen_arms = history["arms"]

    fig = plt.figure(figsize=[30,8])

    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cumulative Rewards")

    fig.add_subplot(122)
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
    set_title("Chosen Actions")
```

Environment

meet.google.com is sharing your screen. Stop sharing Hide

to search

## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help Last saved at 11:11 AM

Comment



ext

Reconnect

Epsilon refers to the probability of choosing to explore. The value for epsilon is selected in between 0 and 1. If epsilon is 0, then it always exploits. If epsilon is 1, then it always explores. If epsilon is 0.1, then 10% of the time it explores and 90% it exploits.

## Greedy Action Selection

ng the libraries

```
numpy as np  
matplotlib.pyplot as plt  
otlib inline
```

plots

```
ot_history(history):  
rds = history["rewards"]  
ewards = history["cum_rewards"]  
en_arms = history["arms"]  
  
= plt.figure(figsize=[30,8])  
= fig.add_subplot(121)
```

### Runtime disconnected

Your runtime has been disconnected due to inactivity or reaching its maximum duration. [Learn more](#)  
If you are interested in longer runtimes with more lenient timeouts, you may want to check out [Colab Pro](#).

[Close](#)

[Reconnect](#)

## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help Last saved at 11:11 AM

Comment

...

ext

exploits.

RAM Disk

## Greedy Action Selection

Importing the libraries

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

Creating plots

```
def plot_history(history):
    rewards = history["rewards"]
    cum_rewards = history["cum_rewards"]
    chosen_arms = history["arms"]

    fig = plt.figure(figsize=[30,8])

    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cumulative Rewards")
```

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_y5HpzbKrNYiWivQSFjLDg#scrollTo=LLAGhdtWcfc-

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help Last saved at 11:11 AM

ext Comment RAM Disk

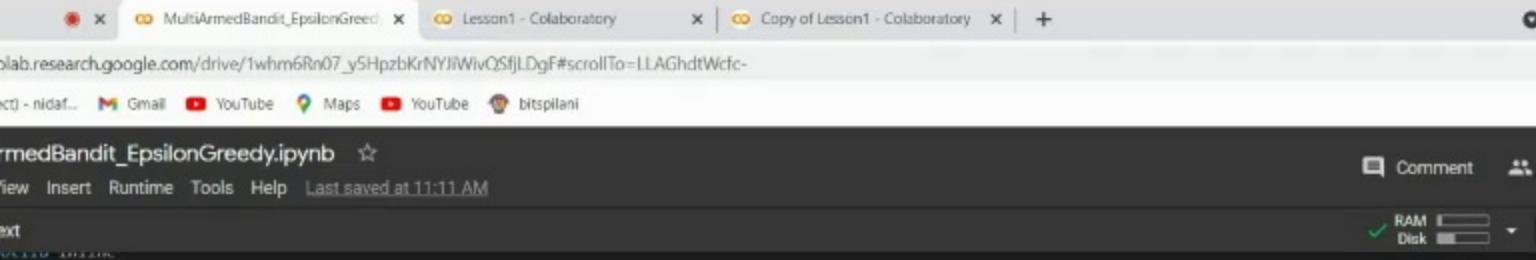
## Greedy Action Selection

Importing the libraries

```
 Loading...  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline
```

Creating plots

```
def plot_history(history):  
    rewards = history["rewards"]  
    cum_rewards = history["cum_rewards"]  
    chosen_arms = history["arms"]  
  
    fig = plt.figure(figsize=[30,8])  
  
    fig.add_subplot(121)  
    plot(cum_rewards, label="avg rewards")  
    set_title("Cumulative Rewards")  
  
    fig.add_subplot(122)  
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")  
    set_title("Chosen Actions")
```



## plots

```
def plot_history(history):
    rewards = history["rewards"]
    cum_rewards = history["cum_rewards"]
    chosen_arms = history["arms"]

    plt.figure(figsize=[30,8])

    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cumulative Rewards")

    fig.add_subplot(122)
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
    set_title("Chosen Actions")
```

## an environment

```
Env(object):

    setting all the parameters as member variables of class
    __init__(self, reward_probabilities, rewards):
        self.reward_probabilities = reward_probabilities # probability of receiving reward from that arm
        self.rewards = rewards # acquired reward from that arm
```

## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help Last saved at 11:11 AM

Comment



```
ext
rds = history["rewards"]
rewards = history["cum_rewards"]
chosen_arms = history["arms"]
```

```
= plt.figure(figsize=[30,8])
```

```
= fig.add_subplot(121)
plot(cum_rewards, label="avg rewards")
set_title("cumulative Rewards")
```

```
= fig.add_subplot(122)
bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
set_title("Chosen Arms")
```

Loading...

an environment

Env(object):

```
etting all the parameters as member variables of class
__init__(self, reward_probabilities, rewards):
    self.reward_probabilities = reward_probabilities # probability of receiving reward from that arm
    self.rewards = rewards # acquired reward from that arm
    self.k_arms = len(rewards) # Number of arms
```

```
choose_arm(self, arm):
    np.random.random() < self.reward_probabilities[arm]:
        return self.rewards[arm]
```

## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help Last saved at 11:11 AM

Comment



RAM



Disk



## Creating an environment

Env(object):

```
        setting all the parameters as member variables of class
    def __init__(self, reward_probabilities, rewards):
        self.reward_probabilities = reward_probabilities # probability of receiving reward from that arm
        self.rewards = rewards # acquired reward from that arm
        self.k_arms = len(rewards) # Number of arms
```

```
choose_arm(self, arm):
    np.random.random() < self.reward_probabilities[arm]:
        return self.rewards[arm]
    else:
        return 0.0
```

## Creating the variables and the environment

```
reward_probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0];
rewards=[95.0, 0.0, 25.5, 10.05, 5.45, 2.50];
environment = Env(reward_probabilities, rewards);

print("K_arms \t\t: {environment.k_arms}")
print("Reward probabilities\t: {environment.reward_probabilities}")
```

## MultiArmedBandit\_EpsilonGreedy.ipynb

View Insert Runtime Tools Help Last saved at 11:11 AM

Comment



ext

RAM Disk

```
etting all the parameters as member variables of class
_init_(self, reward_probabilities, rewards):
lf.reward_probabilities = reward_probabilities # probability of receiving reward from that arm
lf.rewards = rewards # acquired reward from that arm
lf.k_arms = len(rewards) # Number of arms

choose_arm(self, arm):
np.random.random() < self.reward_probabilities[arm]:
return self.rewards[arm]
se:
return 0.0
```

## Creating the variables and the environment

```
_probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0];
s=[95.0, 0.0, 25.5, 10.05, 5.45, 2.5];
nvironment = Env(reward_probabilities, rewards);

f"K_arms \t\t\t: {environment.k_arms}"
f"Reward probabilities\t: {environment.reward_probabilities}"
f"Rewards \t\t\t: {environment.rewards}"

probabilities : 6
probabilities : [0.01, 1.0, 0.75, 0.99, 0.65, 1.0]
s : [95.0, 0.0, 25.5, 10.05, 5.45, 2.5]
```

View Insert Runtime Tools Help All changes saved

next

```
lf.k_arms = len(rewards) # Number of arms
```

```
choose_arm(self, arm):
    np.random.random() < self.reward_probabilities[arm]
    return self.rewards[arm]
else:
    return 0.0
```

#### **Rating the variables and the environment**

probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0]: 75 out of 100 actions give me 25.5

$s=[95.0, 0.0, 25.5, 10.05, 5.45, 2.50]$ :

gment = Env(reward probabilities, rewards);

```
F"\_arms \t\t\t: {environment.k_arms}")
```

```
f"Reward probabilities\t: {environment.reward_probabilities}")
```

```
f"Rewards \t\t: {environment.rewards}")
```

probabilities : [0.01, 1.0, 0.75, 0.99, 0.65, 1.0]  
r : [95.0, 9.0, 25.5, 19.25, 5.45, 2.5]

ing selective action

lab.research.google.com/drive/1whm6Rn07\_y5HpbKrnNYjIWivQSfjLDg#scrollTo=0MbY9fHLjrO

ect) - nida... Gmail YouTube Maps YouTube bitspilani

## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



```
ext
se:
return 0.0
```

✓ RAM
✗ Disk

## Creating the variables and the environment

↑ ↓ ↶ ↷

```
_probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0]; 75 out of 100 actions give me 25.5
s=[95.0, 0.0, 25.5, 10.05, 5.45, 2.50];
environment = Env(reward_probabilities, rewards);

#<__main__.Env object at 0x0000000003E8A0C>
#> arms : {environment.k_arms}
#> Reward probabilities : {environment.reward_probabilities}
#> Rewards : {environment.rewards}
```

```
probabilities : 6
: [0.01, 1.0, 0.75, 0.99, 0.65, 1.0]
s : [95.0, 0.0, 25.5, 10.05, 5.45, 2.5]
```

## Taking selective action

```
comment.choose_arm(3) for _ in range(10)]
```

```
[10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05]
```

## Using Exploration and Exploitation with Epsilon greedy algorithm

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_y5HpbKtNYIiWivQSfjLDg#scrollTo=8\_sbHr1gMrU5

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb

Comment

RAM Disk

View Insert Runtime Tools Help

ext

## Exploring Exploration and Exploitation with Epsilon greedy algorithm

```
EpsilonGreedyAgent(object):
    # Setting all the parameters as member variables of class
    def __init__(self, env, max_iterations=200, epsilon=0.1):
        if env is None:
            raise ValueError("Environment must be provided")
        self.env = env
        self.max_iterations = max_iterations
        self.epsilon = epsilon

    # Function that let the agent act within the environment
    def act(self):
        values = np.zeros(self.env.k_arms)      # Payoff of each arm is set to a
        total_rewards = np.zeros(self.env.k_arms) # Total rewards of each arm is set to 0
        n_counts = np.zeros(self.env.k_arms)     # Number of times each arm is pulled

        rewards = []                          # List to store the actual rewards that agent takes
        avg_rewards = []                     # Average of all the rewards

        for i in range(1, self.max_iterations + 1): # Implementing the epsilon-greedy algorithm
            if np.random.random() < self.epsilon: # random action/exploitation
                arm = np.random.choice(self.env.k_arms)
            else:                                # greedy action/exploitation
                arm = np.argmax(q_values)          # argmax has a property that if 2 index has same value then it chooses the lower index always

            reward = self.env.step(arm)           # Environment step function returns a tuple of reward and next state
            total_rewards[arm] += reward         # Updating total rewards for each arm
            n_counts[arm] += 1                  # Incrementing the count of times each arm is pulled
            avg_rewards.append(total_rewards / n_counts)

        return avg_rewards
```

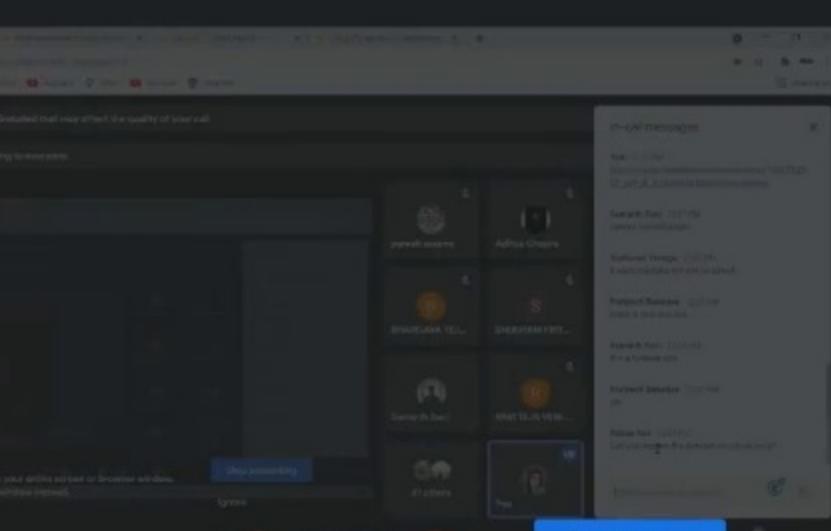
0s completed at 12:28 PM

sheet.google.com/brc-ssbd-pz?pli=1&authuser=1

Gmail YouTube Maps YouTube bitspilani

You have extensions installed that may affect the quality of your call

You're presenting to everyone



iring, don't share your entire screen or browser window.

or a different window instead.



to search

### In-call messages

You 12:21 PM  
[https://colab.research.google.com/QE\\_ayE\\_i8\\_6-2Xu4Xz-BXXR?usp=...](https://colab.research.google.com/QE_ayE_i8_6-2Xu4Xz-BXXR?usp=...)

Samarth Soni 12:21 PM  
cannot save changes

Yashaswi Yenugu 12:21 PM  
it says changes will not be saved

Pratyush Banerjee 12:21 PM  
there is only one link

Samarth Soni 12:21 PM  
it is a viewer link

Pratyush Banerjee 12:22 PM  
yes

Rohan Rao 12:23 PM  
Can you explain the dataset structure

Send a message to everyone

## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



RAM Disk

```
ext
bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
set_title("Chosen Actions")
```

an environment

Env(object):

```
etting all the parameters as member variables of class
__init__(self, reward_probabilities, rewards):
    self.reward_probabilities = reward_probabilities # probability of receiving reward from that arm
    self.rewards = rewards # acquired reward from that arm
    self.k_arms = len(rewards) # Number of arms
```

```
choose_arm(self, arm):
    np.random.random() < self.reward_probabilities[arm]:
        return self.rewards[arm]
    else:
        return 0.0
```

ating the variables and the environment

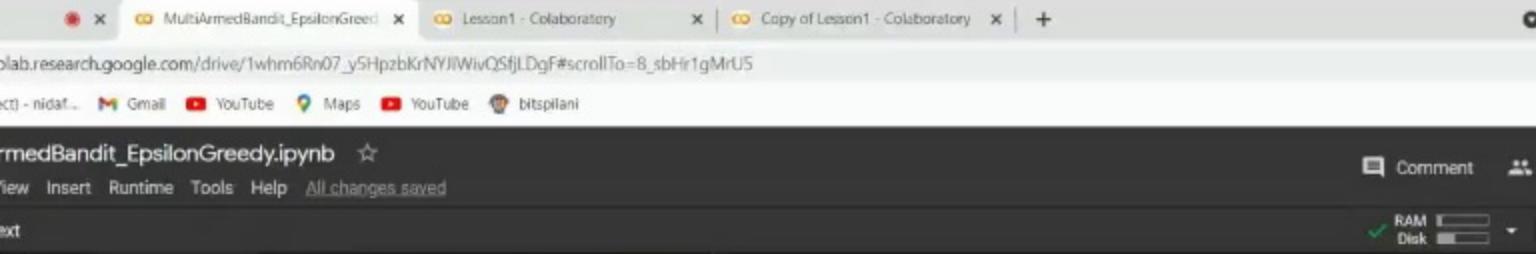
```
_probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0]; 75 out of 100 actions give me 25.5
s=[95.0, 0.0, 25.5, 10.05, 5.45, 2.50];
environment = Env(reward_probabilities, rewards);
```

✓ 0s completed at 12:28 PM

Google Chrome



You're presenting to  
Click here to return to  
when you're ready to  
meet.google.com



## Creating the environment

```
_probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0]; 75 out of 100 actions give me 25.5
s=[95.0, 0.0, 25.5, 10.05, 5.45, 2.50];
environment = Env(reward_probabilities, rewards);

f"X_arms \t\t:t: {environment.k_arms}"
```

## Creating the variables and the environment

```
_probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0]; 75 out of 100 actions give me 25.5
s=[95.0, 0.0, 25.5, 10.05, 5.45, 2.50];
environment = Env(reward_probabilities, rewards);

f"X_arms \t\t:t: {environment.k_arms}"
```

✓ 0s completed at 12:28 PM

to search



You have extensions installed that may affect the quality of your call

You're presenting to everyone

A screenshot of a video call interface. At the top, it says "You're presenting to everyone". Below that, there's a grid of participant icons. A modal window titled "In-call messages" is open, listing messages from various participants. One message from "Samarth Soni" says "cannot save changes". Another from "Yashaswi Yenugu" says "it says changes will not be saved". A third from "Pratyush Banerjee" says "there is only one link". A fourth from "Samarth Soni" says "it is a bviewer link". A fifth from "Pratyush Banerjee" says "yes". A sixth from "Rohan Rao" says "Can you explain the dataset struc...". A seventh from "Rohan Rao" says "No it's clear now". At the bottom of the interface, there's a blue button labeled "Stop presenting".

In-call messages

Samarth Soni 12:21 PM cannot save changes

Yashaswi Yenugu 12:21 PM it says changes will not be saved

Pratyush Banerjee 12:21 PM there is only one link

Samarth Soni 12:21 PM it is a bviewer link

Pratyush Banerjee 12:22 PM yes

Rohan Rao 12:23 PM Can you explain the dataset struc...

Rohan Rao 12:29 PM No it's clear now

Send a message to everyone



## Bandit\_EpsilonGreedy.ipynb ☆

Insert Runtime Tools Help All changes saved

Comment

RAM Disk

```
onGreedyAgent(object):
    all the parameters as member variables of class
    t_(self, env, max_iterations=200, epsilon=0.1):
        v = env
        iterations = max_iterations
        epsilon = epsilon

    that let the agent act within the environment
    elif:
        s = np.zeros(self.env.k_arms) # Payout of each arm is set to 0
        rewards = np.zeros(self.env.k_arms) # Total rewards of each arm is set to 0
        counts = np.zeros(self.env.k_arms) # Number of times each arm is pulled
        rs = []
        rewards = [] # List to store the actual rewards that agent makes
        avg_rewards = [] # Average of all the rewards

    for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm
        if random.random() < self.epsilon: # random action/exploration
            a = np.random.choice(self.env.k_arms)

        else: # greedy action/exploitation
            a = np.argmax(q_values) # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

        d = self.env.choose_arm(a)

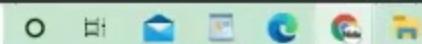
        rewards[a] += reward # Update the values
        counts[a] += 1
        q_values[a] = ave_rewards[a]/ave_counts[a]

        rs.append(reward) # append the values in list
        rewards.append(sum(rewards)/ len(rewards))

    {"counts": ave_counts, "rewards": rewards, "cum_rewards": cum_rewards}
```

ng Epsilon greedy algorithm

0s completed at 12:28 PM



## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

ext

Comment



RAM Disk

## Using Exploration and Exploitation with Epsilon greedy algorithm

```
EpsilonGreedyAgent(object):
    # Setting all the parameters as member variables of class
    def __init__(self, env, max_iterations=200, epsilon=0.1):
        self.env = env
        self.iterations = max_iterations
        self.epsilon = epsilon

    # Method that let the agent act within the environment
    def act(self):
        values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
        total_rewards = np.zeros(self.env.k_arms) # Total rewards of each arm is set to 0
        arm_counts = np.zeros(self.env.k_arms)   # Number of times each arm is pulled

        rewards = []                          # list to store the actual rewards that agent makes
        average_rewards = []                 # Average of all the rewards

        for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm
            if np.random.random() < self.epsilon: # random action/exploration
                arm = np.random.choice(self.env.k_arms)

            else:                            # greedy action/exploitation
                arm = np.argmax(q_values)      # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always
```

✓ 0s completed at 12:28 PM

armedBandit EpsilonGreedy.ipynb

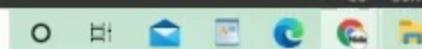
View Insert Runtime Tools Help All changes saved

240

Comment

RAM Disk

✓ Os completed at 12:28 PM



## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



ext

RAM Disk

```
method that let the agent act within the environment
act(self):
    values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
    arm_rewards = np.zeros(self.env.k_arms)   # Total rewards of each arm is set to 0
    arm_counts = np.zeros(self.env.k_arms)     # Number of times each arm is pulled

    rewards = []                                # list to store the actual rewards that agent makes
    cum_rewards = []                            # Average of all the rewards

for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

    if np.random.random() < self.epsilon: # random action/exploration
        arm = np.random.choice(self.env.k_arms)

    else:                               # greedy action/exploitation
        arm = np.argmax(q_values)       # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

    reward = self.env.choose_arm(arm)

    arm_rewards[arm] += reward           # Update the values
    arm_counts[arm] += 1
    q_values[arm] = arm_rewards[arm]/arm_counts[arm]

    rewards.append(reward)              # append the values in list
    cum_rewards.append(sum(rewards)/ len(rewards))

return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}
```

## Introducing Epsilon Greedy algorithm

✓ 0s completed at 12:28 PM

## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



```

ext
cum_rewards = []
# Average of all the rewards

for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

    if np.random.random() < self.epsilon: # random action/exploration
        arm = np.random.choice(self.env.k_arms)

    else:                                # greedy action/exploitation
        arm = np.argmax(q_values)          # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

    reward = self.env.choose_arm(arm)

    arm_rewards[arm] += reward           # Update the values
    arm_counts[arm] += 1
    q_values[arm] = arm_rewards[arm]/arm_counts[arm]

    rewards.append(reward)              # append the values in list
    cum_rewards.append(sum(rewards)/ len(rewards))

return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}

```

✓ RAM 
  
✓ Disk 

## Implementing Epsilon greedy algorithm

```

egy_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=0.01) # instantiate the class
history = egreedy_agent.act() # make the agent to act
print("TOTAL REWARD : {sum(eg_history['rewards'])}")

```

✓ 0s completed at 12:28 PM

## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

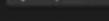
File Insert Runtime Tools Help All changes saved

Comment



ext

RAM Disk



```
EpsilonGreedyAgent(object):
    # Setting all the parameters as member variables of class
    def __init__(self, env, max_iterations=200, epsilon=0.1):
        self.env = env
        self.iterations = max_iterations
        self.epsilon = epsilon

    # Method that let the agent act within the environment
    def act(self):
        values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
        arm_rewards = np.zeros(self.env.k_arms)  # Total rewards of each arm is set to 0
        arm_counts = np.zeros(self.env.k_arms)   # Number of times each arm is pulled

        rewards = []                          # list to store the actual rewards that agent makes
        avg_rewards = []                     # Average of all the rewards

        for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

            if np.random.random() < self.epsilon: # random action/exploration
                arm = np.random.choice(self.env.k_arms)

            else:                                # greedy action/exploitation
                arm = np.argmax(values)           # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

            reward = self.env.choose_arm(arm)

            arm_rewards[arm] += reward          # update the values
            arm_counts[arm] += 1
```

✓ 0s completed at 12:28 PM



## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help All changes saved

Comment



```

method that let the agent act within the environment
def act(self):
    values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
    arm_rewards = np.zeros(self.env.k_arms)  # Total rewards of each arm is set to 0
    arm_counts = np.zeros(self.env.k_arms)   # Number of times each arm is pulled

    rewards = []                            # list to store the actual rewards that agent makes
    cum_rewards = []                        # Average of all the rewards

    for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

        if np.random.random() < self.epsilon: # random action/exploration
            arm = np.random.choice(self.env.k_arms)

        else:                                # greedy action/exploitation
            arm = np.argmax(q_values)          # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

        reward = self.env.choose_arm(arm)

        arm_rewards[arm] += reward           # update the values
        arm_counts[arm] += 1
        q_values[arm] = arm_rewards[arm]/arm_counts[arm]

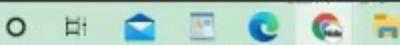
        rewards.append(reward)              # append the values in list
        cum_rewards.append(sum(rewards)/ len(rewards))

    return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}

```

## Introducing Epsilon Greedy algorithm

✓ 0s completed at 12:28 PM



## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



```

    ext
    arm_counts = np.zeros(self.env.k_arms) # Number of times each arm is pulled
    rewards = [] # List to store the action rewards that agent makes
    cum_rewards = [] # Average of all the rewards

    for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

        if np.random.random() < self.epsilon: # random action/exploration
            arm = np.random.choice(self.env.k_arms)

        else: # greedy action/exploitation
            arm = np.argmax(q_values)
            # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

        reward = self.env.choose_arm(arm)

        arm_rewards[arm] += reward # Update the values
        arm_counts[arm] += 1
        q_values[arm] = arm_rewards[arm]/arm_counts[arm]

        rewards.append(reward) # append the values in list
        cum_rewards.append(sum(rewards)/len(rewards))

    return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}

```

 ✓ RAM
   
 ✓ Disk
 

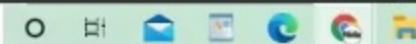
## Implementing Epsilon greedy algorithm

```

    my_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=0.01) # instantiate the class
    trajectory = egreedy_agent.act() # make the agent to act

```

✓ 0s completed at 12:28 PM



## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help All changes saved

Comment



```

ext

etting all the parameters as member variables of class
__init__(self, env, max_iterations=200, epsilon=0.1):
    self.env = env
    self.iterations = max_iterations
    self.epsilon = epsilon

method that let the agent act within the environment
act(self):
    values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
    arm_rewards = np.zeros(self.env.k_arms)  # Total rewards of each arm is set to 0
    arm_counts = np.zeros(self.env.k_arms)   # Number of times each arm is pulled

    rewards = []                          # list to store the actual rewards that agent makes
    avg_rewards = []                      # Average of all the rewards

for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

    if np.random.random() < self.epsilon: # random action/exploration
        arm = np.random.choice(self.env.k_arms)

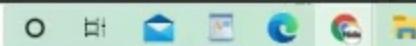
    else:                                # greedy action/exploitation
        arm = np.argmax(q_values)          # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

    reward = self.env.choose_arm(arm)

    arm_rewards[arm] += reward           # update the values
    arm_counts[arm] += 1
    q_values[arm] = arm_rewards[arm]/arm_counts[arm]

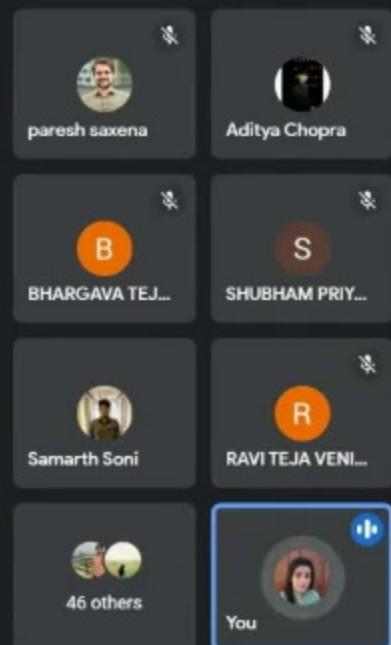
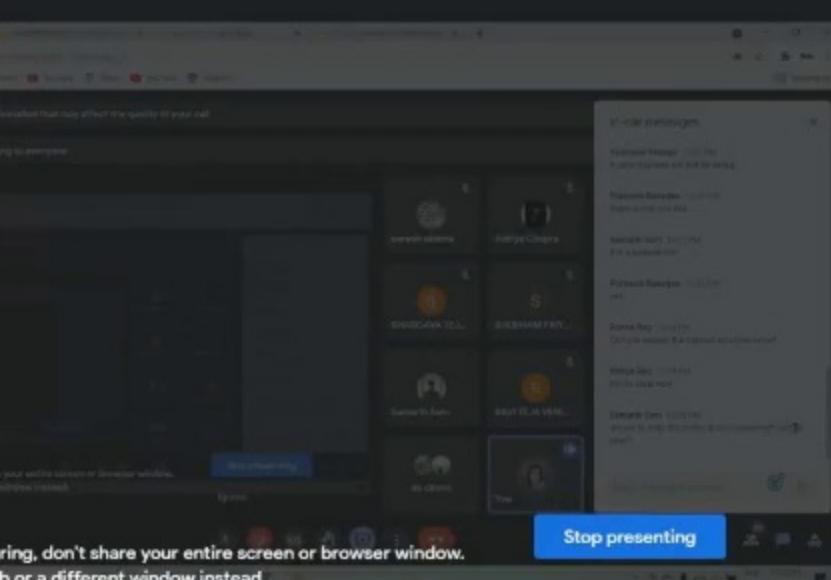
```

✓ 0s completed at 12:28 PM



You have extensions installed that may affect the quality of your call

You're presenting to everyone



### In-call messages

Yashaswi Yenugu 12:21 PM  
it says changes will not be saved

Pratyush Banerjee 12:21 PM  
there is only one link

Samarth Soni 12:21 PM  
It is a bviewer link

Pratyush Banerjee 12:22 PM  
yes

Rohan Rao 12:23 PM  
Can you explain the dataset structure

Rohan Rao 12:29 PM  
No it's clear now

Samarth Soni 12:29 PM  
are we to write the codes in our colab later?

Send a message to everyone

## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



```
ext
def update_qs(arm, reward):
    arm_counts[arm] += 1
    q_values[arm] = arm_rewards[arm]/arm_counts[arm]

rewards.append(reward)           # append the values in list
cum_rewards.append(sum(rewards)/ len(rewards))

return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}
```

✓ RAM
Disk

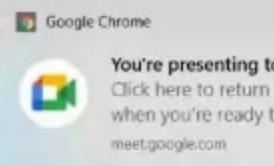
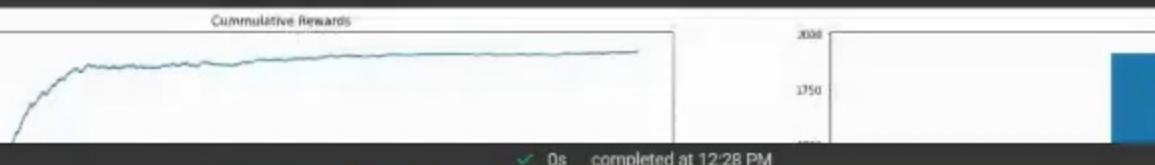
## Presenting Epsilon greedy algorithm

```
eg_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=0.01)      # instantiate the class
history = eg_agent.act()               # make the agent to act
f"TOTAL REWARD : {sum(eg_history['rewards'])})")
```

REWARD : 36817.399999999994

## Plot for Epsilon greedy algorithm

history(eg\_history)



## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment

...

ext

RAM Disk

```
egy_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=0.01)    # instantiate the class
history = egreedy_agent.act()                                                    # make the agent to act
print("TOTAL REWARD : {sum(eg_history['rewards'])})")
```

REWARD : 36817.39999999994

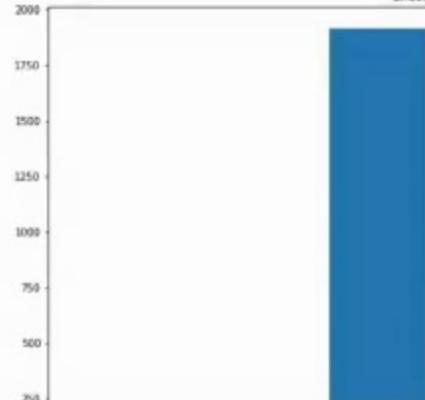
## Plot for Epsilon greedy algorithm

history(eg\_history)

Cumulative Rewards



Chosen Actions



0s completed at 12:28 PM

## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

ext

```
rewards.append(reward) # append the values in list
cum_rewards.append(sum(rewards)/ len(rewards))

return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}
```

Comment



RAM Disk

## Implementing Epsilon greedy algorithm

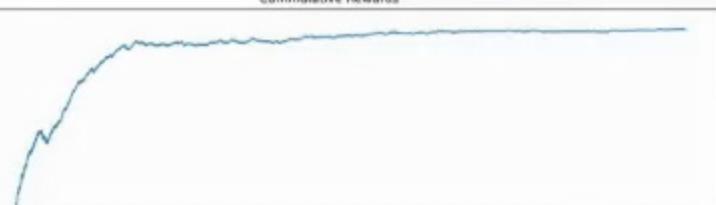
```
eg_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=0.01) # instantiate the class
history = eg_agent.act() # make the agent to act
print("TOTAL REWARD : {sum(eg_history['rewards'])})")
```

REWARD : 36817.39999999994

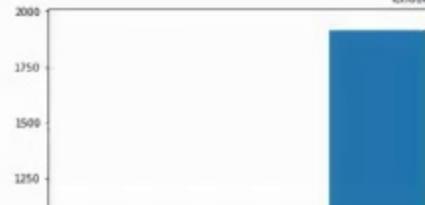
## Plot for Epsilon greedy algorithm

history(eg\_history)

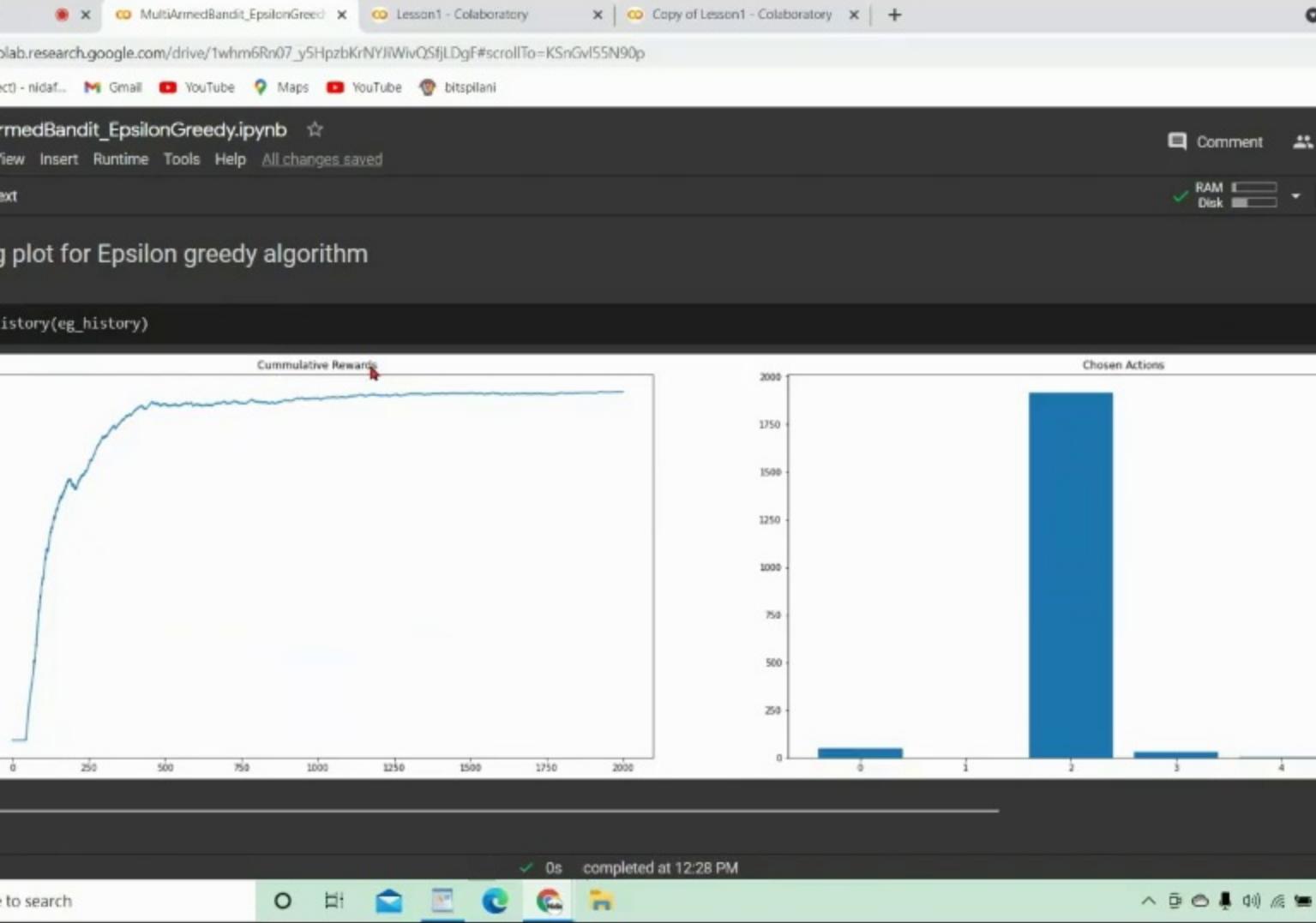
Cummulative Rewards



Chosen Actions



✓ 0s completed at 12:28 PM



## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

View Insert Runtime Tools Help All changes saved

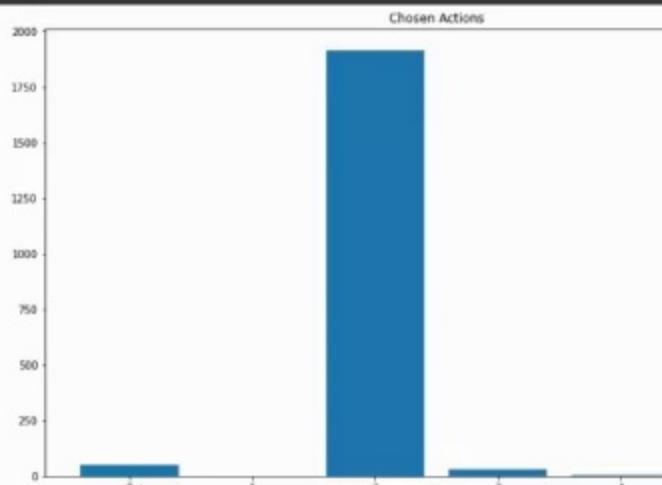
Comment



## Plot for epsilon greedy algorithm

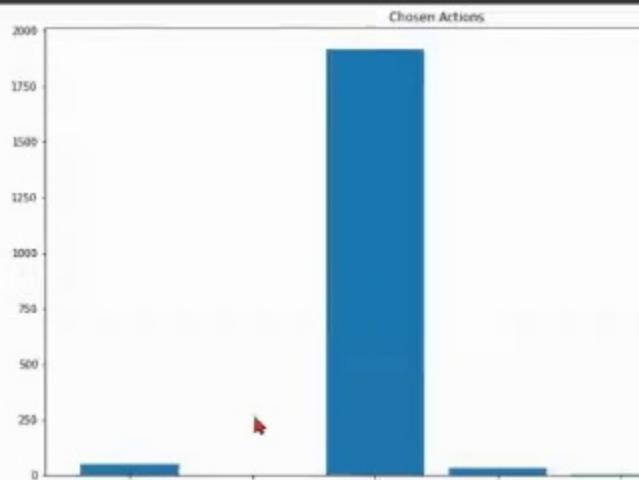
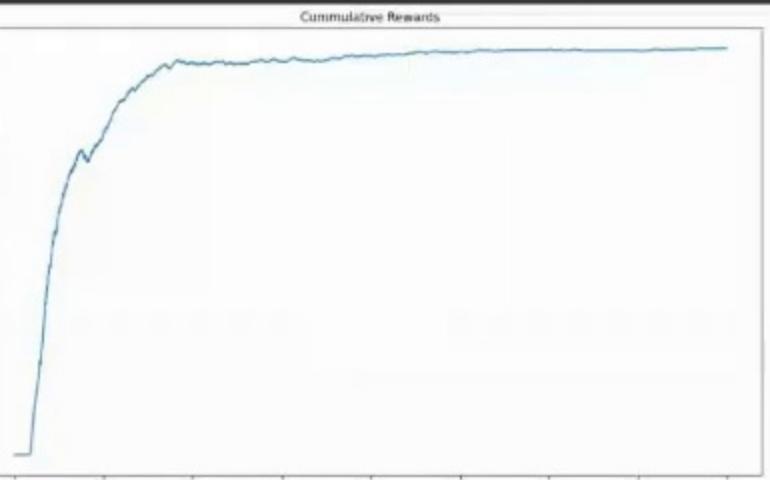
RAM Disk

history(eg\_history)



## Plot for Epsilon greedy algorithm

istory(eg\_history)



## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help All changes saved

Comment

...

```

ext
    if np.random.uniform() < self.epsilon:      # Exploitation
        arm = np.random.choice(self.env.k_arms)
    else:                                     # Greedy action/exploitation
        arm = np.argmax(q_values)             # argmax has a property that if 2 index has same qvalue then it chooses the lower index always
    reward = self.env.choose_arm(arm)

    arm_rewards[arm] += reward            # Update the values
    arm_counts[arm] += 1
    q_values[arm] = arm_rewards[arm]/arm_counts[arm]

    rewards.append(reward)                # append the values in list
    cum_rewards.append(sum(rewards)/len(rewards))

return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}

```

## Implementing Epsilon greedy algorithm

```

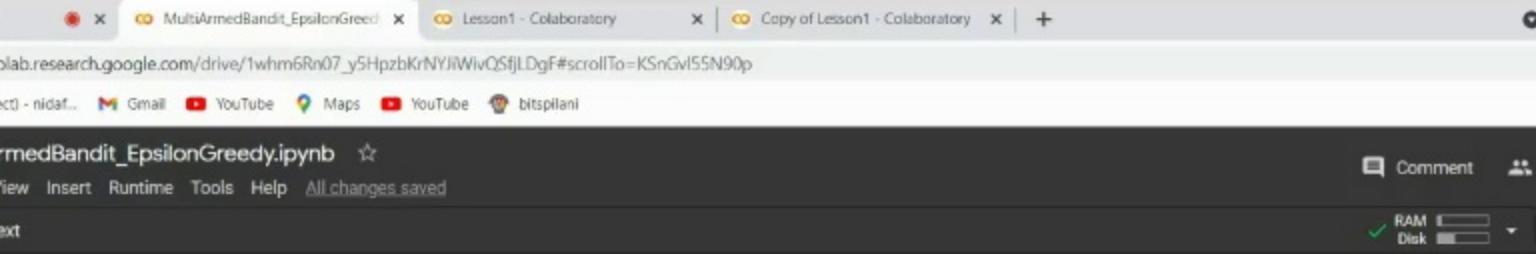
y_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=0.01)    # instantiate the class
history = egreedy_agent.act()                                              # make the agent to act
f"TOTAL REWARD : {sum(eg_history['rewards'])}"

```

REWARD : 36817.39999999994

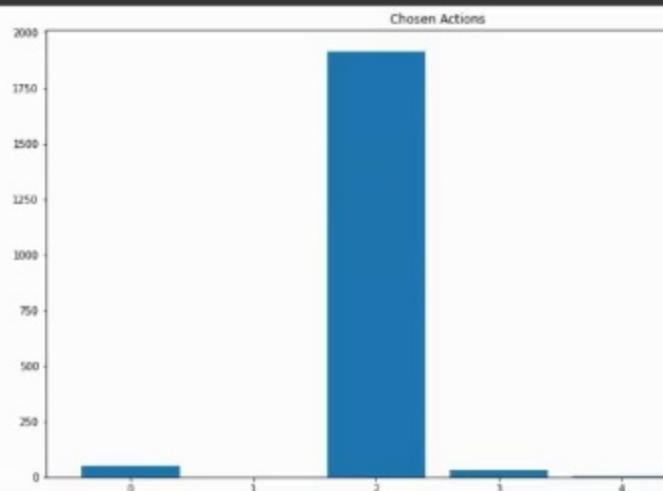
## Plot for Epsilon greedy algorithm

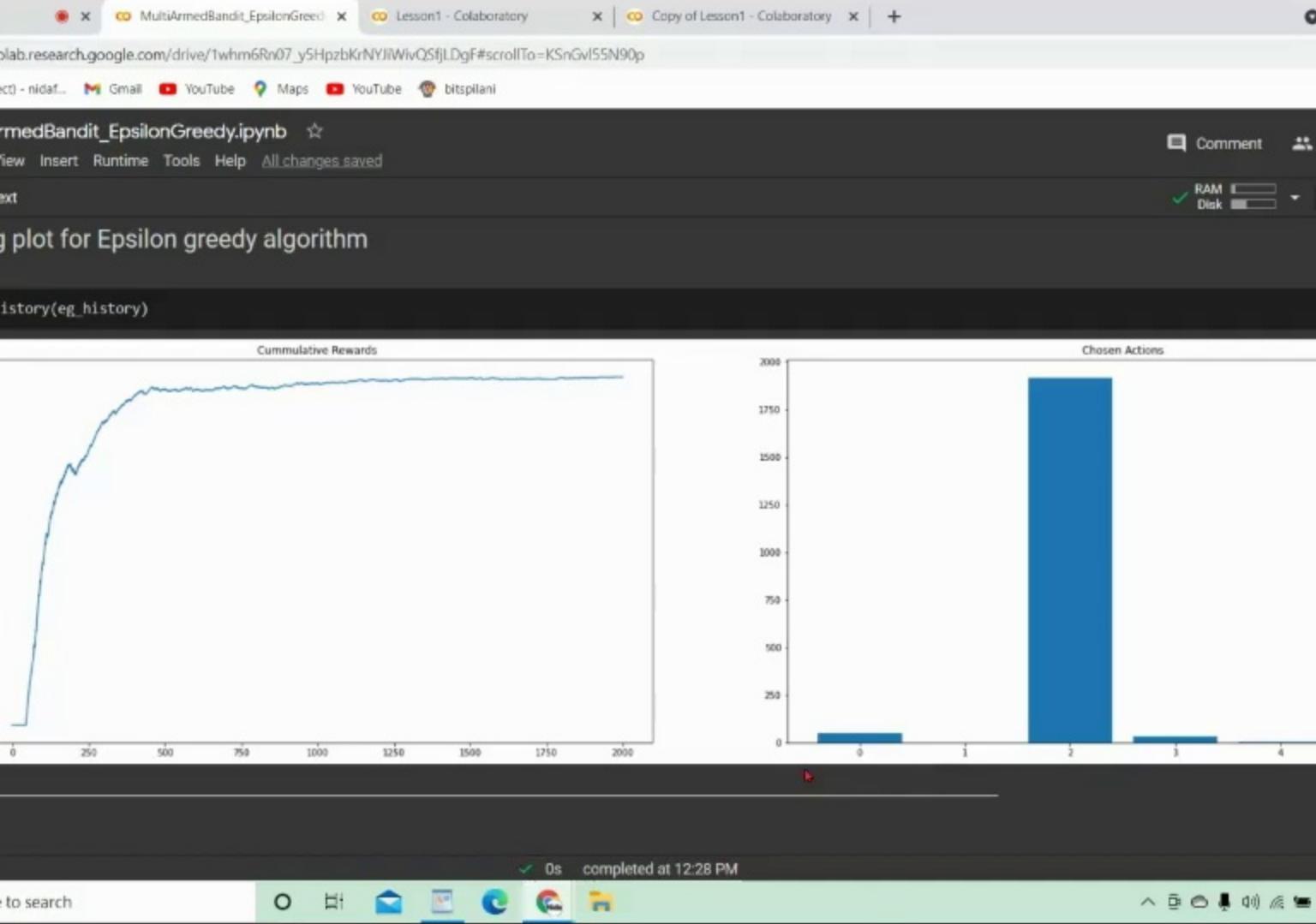
Completed at 12:28 PM



## Plot for Epsilon greedy algorithm

history(eg\_history)





MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_ySHpzbKrNYiWivQSfjLDg#scrollTo=KSnGvI55N90p

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb

View Insert Runtime Tools Help All changes saved

ext

turn {"arms": arm\_counts, "rewards": rewards, "cum\_rewards": cum\_rewards}

enting Epsilon greedy algorithm

```
y_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=0.01) # instantiate the class
tory = egreedy_agent.act() # make the agent to act
+"TOTAL REWARD : {sum(eg_history['rewards'])})"}
```

REWARD : 36817.39999999994

g plot for Epsilon greedy algorithm

```
istory(eg_history)
```

Cumulative Rewards

Chosen Actions

0s completed at 12:28 PM

## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help

Comment



RAM Disk

```
probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0]; 75 out of 100 actions give me 25.5
s=[95.0, 0.0, 25.5, 10.05, 5.45, 2.58];
environment = Env(reward_probabilities, rewards);

# arms \t\t:t: {environment.k_arms}
# Reward probabilities\t: {environment.reward_probabilities}
# Rewards \t\t:t: {environment.rewards}

<ipython-input-16-4e4eab007f>, line 2
reward_probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0]; 75 out of 100 actions give me 25.5
^
Error: invalid syntax
```

Stack Overflow

ing selective action

onment.choose\_arm(3) for \_ in range(10)]

ng Exploration and Exploitation with Epsilon greedy algorithm

## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help

Comment

...

ext

RAM Disk

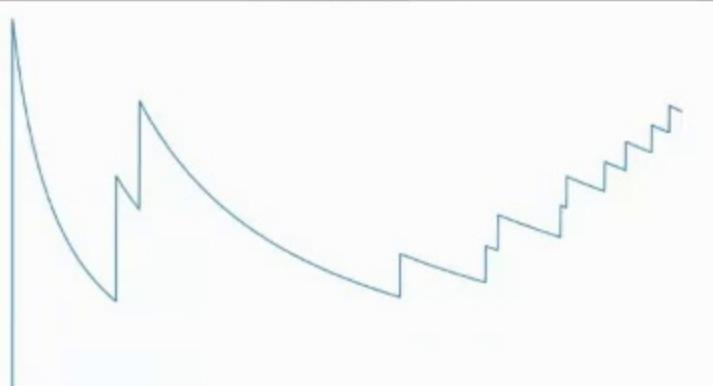
```
egy_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=0.00)      # instantiate the class
tory = egreedy_agent.act()                                                       # make the agent to act
f"TOTAL REWARD : {sum(eg_history['rewards'])}"
```

REWARD : 1140.0

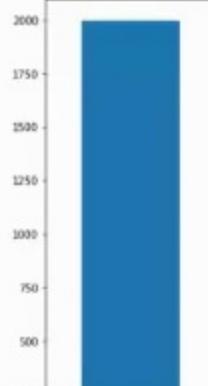
### Plot for Epsilon greedy algorithm

istory(eg\_history)

Cumulative Rewards



Chosen Actions



✓ 0s completed at 12:41 PM

to search



## edBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help

Comment

RAM Disk

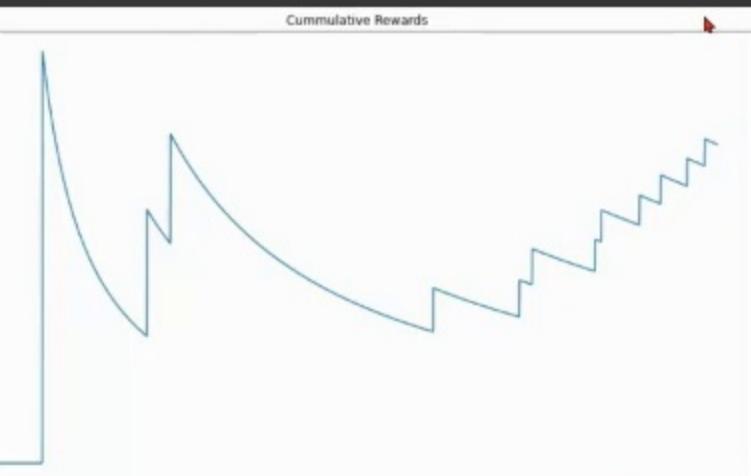
```
agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=0.00) # instantiate the class
history = egreedy_agent.act() # make the agent to act
TOTAL REWARD : (sum(eg_history['rewards']))"
```

TOTAL REWARD : 1140.0

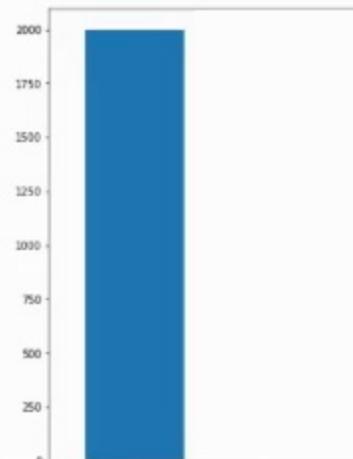
## Plot for Epsilon greedy algorithm

plot(eg\_history)

Cumulative Rewards



Chosen Actions



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_y5HpzbKrNYIiWivQSfjLDgF#scrollTo=0MbY94HLjrO

ect) - nida... Gmail YouTube Maps YouTube bitspilani

edBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

ry = egreedy\_agent.act()  
TOTAL REWARD : {sum(eg\_history['rewards'])}"

WARD : 1140.0

plot for Epsilon greedy algorithm

```
story(eg_history)
```

Cummulative Rewards

Chosen Actions

Action	Value
0	~2000
1	~10
2	~10
3	~10

0s completed at 12:41 PM

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_y5HpbzKrNYIWivQSfjLDg#scrollTo=0MbY94HLjrO

ect) - nida... Gmail YouTube Maps YouTube bitspilani

edBandit\_EpsilonGreedy.ipynb ☆

Insert Runtime Tools Help All changes saved

Comment RAM Disk

ng the variables and the environment

```
robabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0];
[95.0, 0.0, 25.5, 10.05, 5.45, 2.58];
env = Env(reward_probabilities, rewards);

K_arms : t:t: {environment.k_arms}
Reward_probabilities:t: {environment.reward_probabilities}
Rewards :t:t: {environment.rewards}

probabilities : 6
: [0.01, 1.0, 0.75, 0.99, 0.65, 1.0]
: [95.0, 0.0, 25.5, 10.05, 5.45, 2.5]
```

g selective action

```
agent.choose_arm(3) for _ in range(10)]
```

[10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05]

Exploration and Exploitation with Epsilon greedy algorithm

```
EpsilonGreedyAgent(object):
```

ing all the parameters as member variables of class

```
init_(self, env, max_iterations=200, epsilon=0.1):
```

0s completed at 12:41 PM

to search



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_y5HpbKtNYiWlvQSfjLDg#scrollTo=0MbY9f4HLjrO

ect) - nida... Gmail YouTube Maps YouTube bitspilani

edBandit\_EpsilonGreedy.ipynb ☆

Insert Runtime Tools Help All changes saved

Comment RAM Disk

```
return self.rewards[arm]
:
return 0.0
```

ng the variables and the environment

```
probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0];
[95.0, 0.0, 25.5, 10.05, 5.45, 2.50];
env = Env(reward_probabilities, rewards);

K_arms \t\l\t: {environment.k_arms}")
Reward probabilities\t: {environment.reward_probabilities}")
Rewards \t\l\t: {environment.rewards}")

probabilities : 6
: [0.01, 1.0, 0.75, 0.99, 0.65, 1.0]
: [95.0, 0.0, 25.5, 10.05, 5.45, 2.5]
```

g selective action

```
ment.choose_arm(3) for _ in range(10)]

[10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05]
```

Exploration and Exploitation with Epsilon greedy algorithm

```
EpsilonGreedyAgent(object):
    0s completed at 12:41 PM
```

to search

## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help All changes saved

```
ext
np.random.random() < self.reward_probabilities[arm]:
    return self.rewards[arm]
else:
    return 0.0
```

Comment



RAM Disk

## Creating the variables and the environment

```
_probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0];
s=[95.0, 0.0, 25.5, 10.05, 5.45, 2.50];
environment = Env(reward_probabilities, rewards);

f"K_arms \t\t: {environment.k_arms}"
f"Reward probabilities\t: {environment.reward_probabilities}"
f"Rewards \t\t: {environment.rewards}"
```

```
probabilities      : 6
probabilities      : [0.01, 1.0, 0.75, 0.99, 0.65, 1.0]
s                  : [95.0, 0.0, 25.5, 10.05, 5.45, 2.5]
```

## Choosing selective action

```
environment.choose_arm(3) for _ in range(10)]
```

```
, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05]
```

✓ 0s completed at 12:41 PM

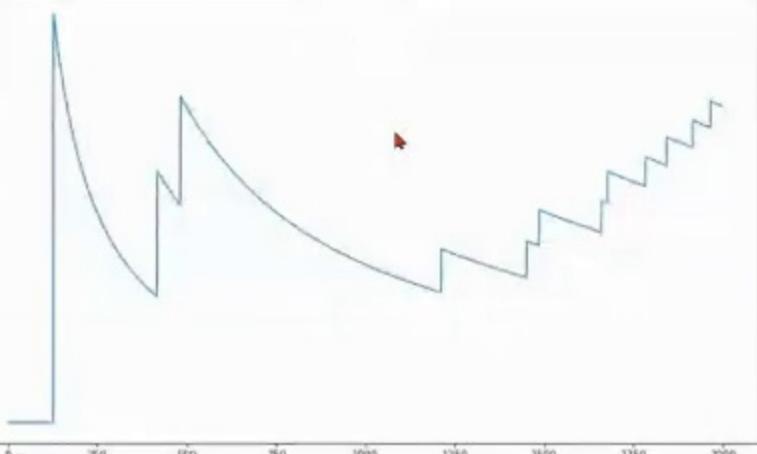
## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help All changes saved

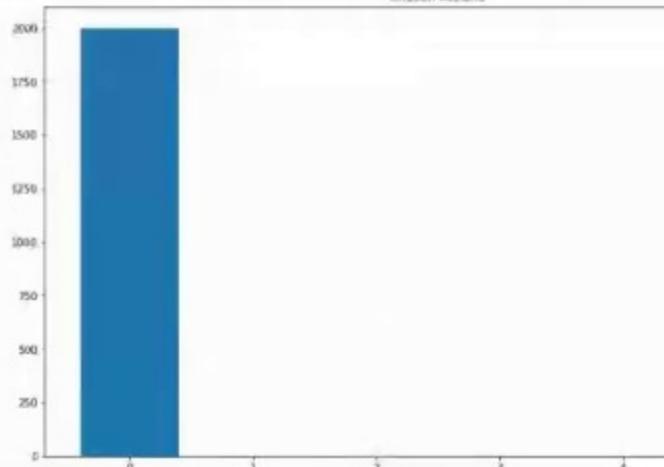
ext  
g plot for epsilon greedy algorithm

history(eg\_history)

Cumulative Rewards



Chosen Actions



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_ySHpzKrnNYiWivQSfjLDg#scrollTo=KSnyGvI55N90p

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help Saving...

ext

return {"arms": arm\_counts, "rewards": rewards, "cum\_rewards": cum\_rewards}

Presenting Epsilon greedy algorithm

```
y_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=1) # instantiate the class
history = egreedy_agent.act() # make the agent to act
f"TOTAL REWARD : {sum(eg_history['rewards'])}"
```

REWARD : 1140.0

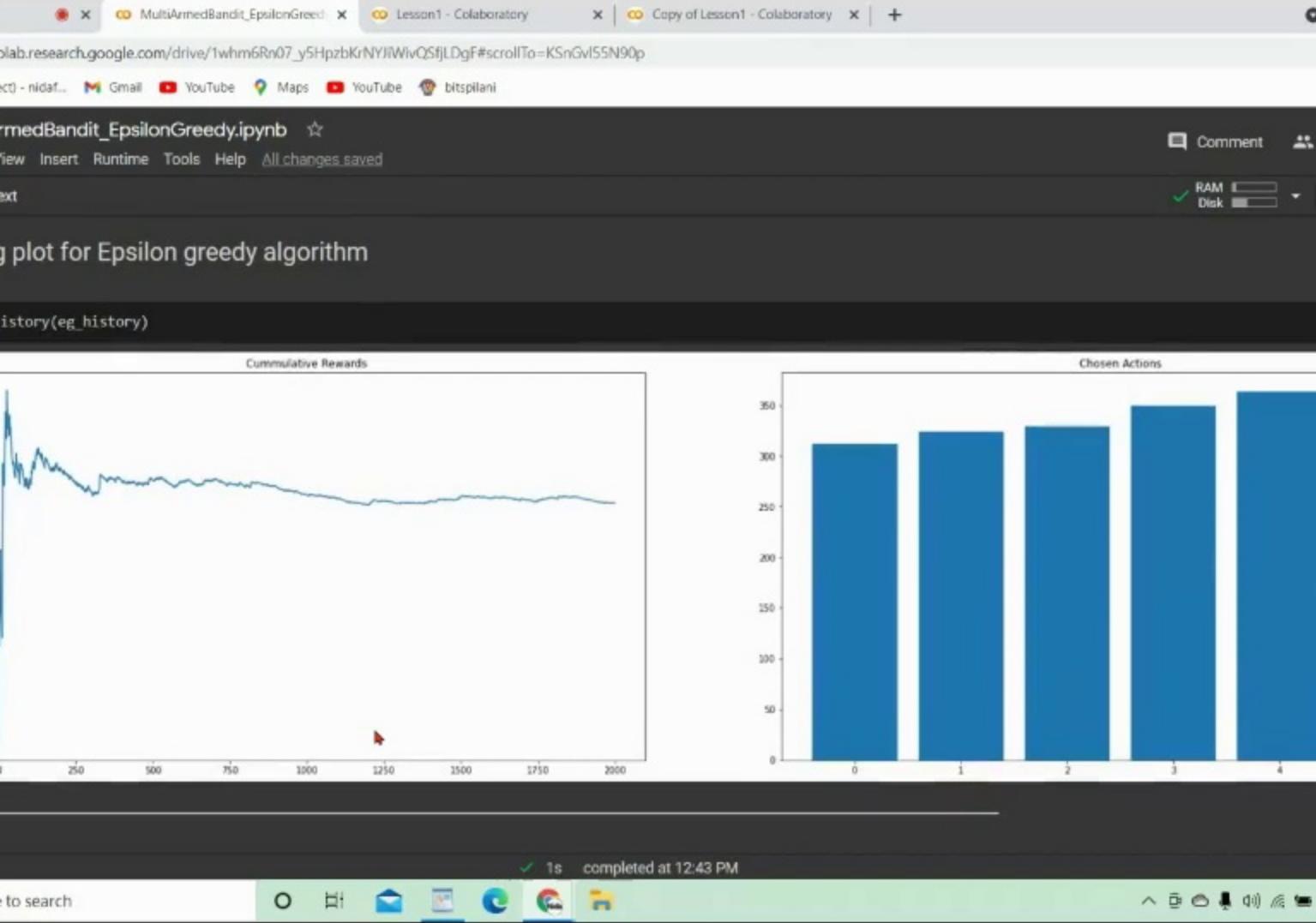
Plot for Epsilon greedy algorithm

```
history(eg_history)
```

Cumulative Rewards

Chosen Actions

0s completed at 12:41 PM



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_ySHpbzKrNYIiWivQSfjLDg#scrollTo=pJ5677k3PDoF

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb ☆

View Insert Runtime Tools Help All changes saved

ext

Comment RAM Disk

## Implementing Epsilon greedy algorithm

```
eg_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=1.0)
history = eg_agent.act()
f"TOTAL REWARD : {sum(eg_history['rewards'])})"
```

REWARD : 12473.799999999987

## Plot for Epsilon greedy algorithm

history(eg\_history)



## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



```
ext
    # update the values
    arm_counts[arm] += 1
    q_values[arm] = arm_rewards[arm]/arm_counts[arm]

    rewards.append(reward)           # append the values in list
    cum_rewards.append(sum(rewards)/ len(rewards))

    return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}
```

✓ RAM
✓ Disk

## Implementing Epsilon greedy algorithm

```
egy_agent = EpsilonGreedyAgent(environment, max_iterations=2000, epsilon=1.0)
egy_agent = egreedy_agent.act()                                # instantiate the class
                                                               # make the agent to act
print(f"TOTAL REWARD : {sum(eg_history['rewards'])}")
```

REWARD : 12473.79999999987

## Plot for Epsilon greedy algorithm

history(eg\_history)

Cumulative Rewards



## MultiArmedBandit\_EpsilonGreedy.ipynb

View Insert Runtime Tools Help All changes saved

Comment



ext

RAM Disk

, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05]

## Exploration and Exploitation with Epsilon greedy algorithm

```
EpsilonGreedyAgent(object):
    Setting all the parameters as member variables of class
    __init__(self, env, max_iterations=200, epsilon=0.1):
        If.env = env
        If.iterations = max_iterations
        If.epsilon = epsilon

    Method that let the agent act within the environment
    act(self):
        values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
        sum_rewards = np.zeros(self.env.k_arms)  # Total rewards of each arm is set to 0
        sum_counts = np.zeros(self.env.k_arms)   # Number of times each arm is pulled

        rewards = []                          # list to store the actual rewards that agent makes
        sum_rewards = []                      # Average of all the rewards

    for i in range(i, self.iterations + 1): # choose action using epsilon greedy algorithm

        if np.random.random() < self.epsilon: # random action/exploration
            arm = np.random.choice(self.env.k_arms)

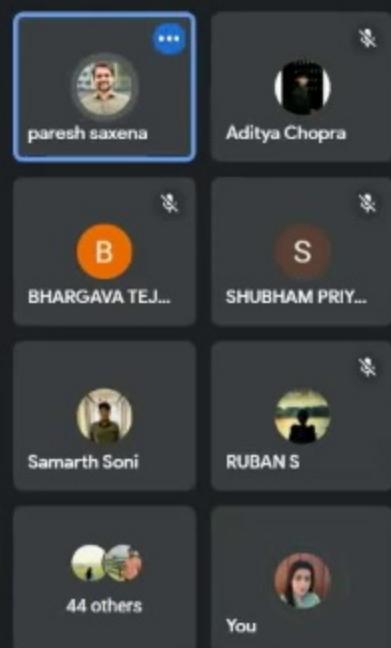
        else:                                # greedy action/exploitation
```

✓ 1s completed at 12:43 PM

You have extensions installed that may affect the quality of your call

You're presenting to everyone

A screenshot of a Google Meet presentation slide. At the top, it says "You're presenting to everyone". Below that, there's a message about extensions affecting call quality. A modal window titled "Presenting" shows a list of participants: Aditya Chopra, paresh saxena, Pratyush Banerjee, BHARGAVA TEJ..., SHUBHAM PRIY..., Samarth Soni, RUBAN S, and 44 others. The participant "paresh saxena" is highlighted with a blue border. The main slide area shows a grid of participant icons. At the bottom, there's a message: "Ring, don't share your entire screen or browser window. Share a different window instead." A "Stop presenting" button is visible.



### In-call messages

Aditya Chopra 12:38 PM  
in cum\_rewards, why are we taking

Pratyush Banerjee 12:40 PM  
it's a running average right

Yashaswi Yenugu 12:41 PM  
it's supposed to be the sum right?

Saloni Singh 12:41 PM  
always exploits

Aditya Chopra 12:42 PM  
Exp Value = reward Prob[] \* reward

Samarth Soni 12:43 PM  
@aditya you mean expected reward

Aditya Chopra 12:44 PM  
yeah

Send a message to everyone



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_ySHpbKrnNYIiWivQSFjLDg#scrollTo=pJ5677k3P0DF

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

RAM Disk

, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05]

## Using Exploration and Exploitation with Epsilon greedy algorithm

```
EpsilonGreedyAgent(object):
    Setting all the parameters as member variables of class
    __init__(self, env, max_iterations=200, epsilon=0.1):
        If.env = env
        If.iterations = max_iterations
        If.epsilon = epsilon

    Method that let the agent act within the environment
    act(self):
        values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
        m_rewards = np.zeros(self.env.k_arms)    # Total rewards of each arm is set to 0
        m_counts = np.zeros(self.env.k_arms)     # Number of times each arm is pulled

        rewards = []                          # list to store the actual rewards that agent makes
        m_rewards = []                        # Average of all the rewards

        for i in range(i, self.iterations + 1): # choose action using epsilon greedy algorithm
            if np.random.random() < self.epsilon: # random action/exploration
                arm = np.random.choice(self.env.k_arms)
            else:                                # greedy action/exploitation
                arm = np.argmax(values)

            reward = env.step(arm)               # get reward for taking action
            values[arm] += reward / m_counts[arm] # update payout for arm
            m_rewards[arm] += reward             # update total reward for arm
            m_counts[arm] += 1                  # increment count of times arm was pulled

    def get_average_reward(self):
        return np.mean(m_rewards)
```

✓ 1s completed at 12:43 PM

Google Chrome You're presenting to Click here to return to when you're ready to meet.google.com

## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



```

ext
lf.epsilon = epsilon

method that let the agent act within the environment
act(self):
    values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
    arm_rewards = np.zeros(self.env.k_arms)   # Total rewards of each arm is set to 0
    arm_counts = np.zeros(self.env.k_arms)     # Number of times each arm is pulled

    rewards = []                            # list to store the actual rewards that agent makes
    cum_rewards = []                        # Average of all the rewards

for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

    if np.random.random() < self.epsilon: # random action/exploration
        arm = np.random.choice(self.env.k_arms)

    else:                                # greedy action/exploitation
        arm = np.argmax(q_values)          # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

    reward = self.env.choose_arm(arm)

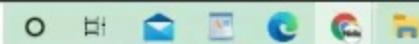
    arm_rewards[arm] += reward           # Update the values
    arm_counts[arm] += 1
    q_values[arm] = arm_rewards[arm]/arm_counts[arm]

    rewards.append(reward)              # append the values in list
    cum_rewards.append(sum(rewards)/ len(rewards))

return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}

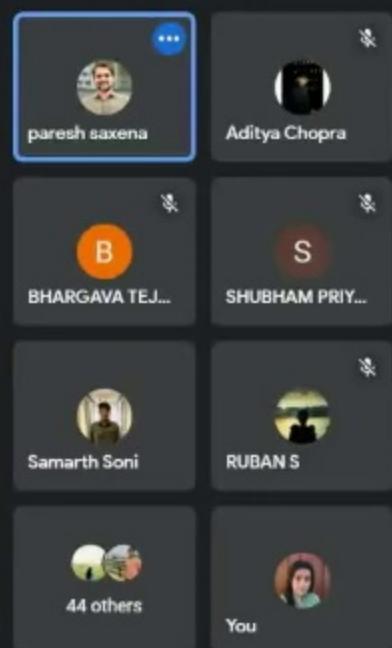
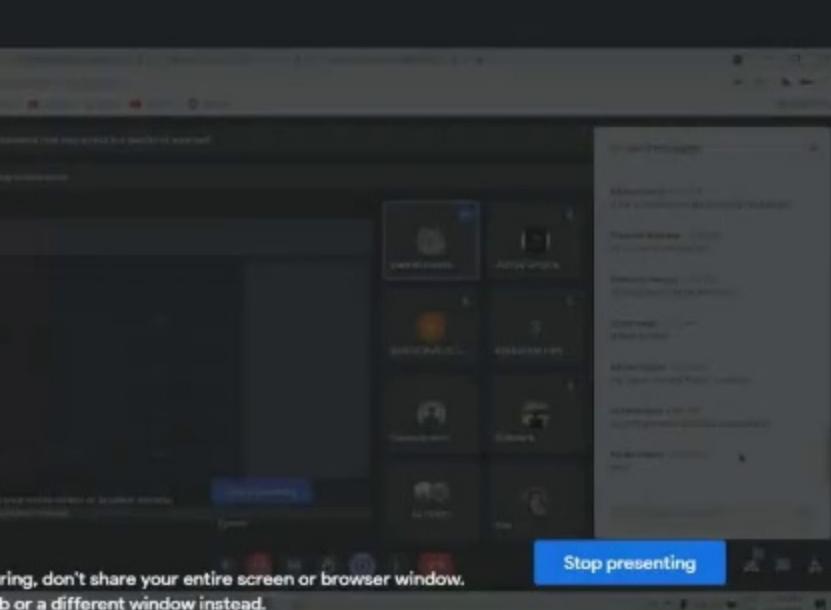
```

✓ 1s completed at 12:43 PM



You have extensions installed that may affect the quality of your call

You're presenting to everyone



### In-call messages

Aditya Chopra 12:38 PM  
in cum\_rewards, why are we taking

Pratyush Banerjee 12:40 PM  
it's a running average right

Yashaswi Yenugu 12:41 PM  
it's supposed to be the sum right?

Saloni Singh 12:41 PM  
always exploits

Aditya Chopra 12:42 PM  
Exp Value = reward Prob[] \* reward

Samarth Soni 12:43 PM  
@aditya you mean expected reward

Aditya Chopra 12:44 PM  
yeah

Send a message to everyone



MultiArmedBandit\_EpsilonGreedy | Lesson1 - Colaboratory | Copy of Lesson1 - Colaboratory | +

lab.research.google.com/drive/1GICF0zNQF\_ayFjB\_6-2Xu4Axz-BXXR#scrollTo=R1qc-8-xrf5I

Gmail YouTube Maps YouTube bitspilani

New Insert Runtime Tools Help All changes saved

Comment Reconnect

```
numpy as np
matplotlib.pyplot as plt
%matplotlib inline
```

↑ ↓ ↻

## the plots

```
def plot_history(history):
    rds = history["rewards"]
    rewards = history["cum_rewards"]
    en_arms = history["arms"]

    plt.figure(figsize=[30,8])

    fig.add_subplot(121)
    plot(cum_rewards, label="avg rewards")
    set_title("Cumulative Rewards")

    fig.add_subplot(122)
    bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
    set_title("Chosen Actions")
```

**Runtime disconnected**

Your runtime has been disconnected due to inactivity or reaching its maximum duration. [Learn more](#)  
If you are interested in longer runtimes with more lenient timeouts, you may want to check out [Colab Pro](#).

[Close](#) [Reconnect](#)

Google Chrome

You're presenting to  
Click here to return to  
when you're ready to  
meet.google.com

Environment

completed at 10:40 AM

to search

O Hi ! Gmail Drive Google Sheets Google Slides Google Sheets

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1IGICF0zNQF\_ayFj8\_6-2Xu4Axz-BXXR#scrollTo=R1qc-8-xrf5I

Gmail YouTube Maps YouTube bitspilani

All changes saved

```
ext
= fig.add_subplot(121)
plot(cum_rewards, label="avg rewards")
set_title("Cumulative Rewards")

= fig.add_subplot(122)
bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
set_title("Chosen Actions")
```

## Environment

I

ating the environment

ing a selective action

ng Exploration and Exploitation with epsilon greedy algorithm

✓ 0s completed at 10:40 AM

to search



## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



ext

RAM

Disk

```
EpsilonGreedyAgent(object):
    # Setting all the parameters as member variables of class
    def __init__(self, env, max_iterations=200, epsilon=0.1):
        self.env = env
        self.iterations = max_iterations
        self.epsilon = epsilon

    # Method that let the agent act within the environment
    def act(self):
        values = np.zeros(self.env.k_arms)          # Payout of each arm is set to 0
        arm_rewards = np.zeros(self.env.k_arms)      # Total rewards of each arm is set to 0
        arm_counts = np.zeros(self.env.k_arms)        # Number of times each arm is pulled

        rewards = []                                # List to store the actual rewards that agent makes
        avg_rewards = []                            # Average of all the rewards

        for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

            if np.random.random() < self.epsilon: # random action/exploration
                arm = np.random.choice(self.env.k_arms)

            else:                               # greedy action/exploitation
                arm = np.argmax(q_values)         # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

            reward = self.env.choose_arm(arm)

            arm_rewards[arm] += reward          # Update the values
            arm_counts[arm] += 1
            q_values[arm] = arm_rewards[arm]/arm_counts[arm]
```

✓ 1s completed at 12:43 PM

## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help All changes saved

Comment



```
ext

if.iterations = max_iterations
if.epsilon = epsilon

method that let the agent act within the environment
act(self):
    values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
    arm_rewards = np.zeros(self.env.k_arms)   # Total rewards of each arm is set to 0
    arm_counts = np.zeros(self.env.k_arms)     # Number of times each arm is pulled

    rewards = []                            # list to store the actual rewards that agent makes
    cum_rewards = []                        # Average of all the rewards

for i in range(i, self.iterations + 1): # choose action using epsilon greedy algorithm

    if np.random.random() < self.epsilon: # random action/exploration
        arm = np.random.choice(self.env.k_arms)

    else:                                # greedy action/exploitation
        arm = np.argmax(q_values)          # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

    reward = self.env.choose_arm(arm)

    arm_rewards[arm] += reward           # Update the values
    arm_counts[arm] += 1
    q_values[arm] = arm_rewards[arm]/arm_counts[arm]

    rewards.append(reward)              # append the values in list
    cum_rewards.append(sum(rewards)/ len(rewards))

return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}
```

✓ 1s completed at 12:43 PM



## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



ext

RAM

Disk

## Creating an environment

Env(object):

```
    setting all the parameters as member variables of class
    def __init__(self, reward_probabilities, rewards):
        self.reward_probabilities = reward_probabilities # probability of receiving reward from that arm
        self.rewards = rewards # acquired reward from that arm
        self.k_arms = len(rewards) # number of arms
```

```
    choose_arm(self, arm):
        np.random.random() < self.reward_probabilities[arm]:
            return self.rewards[arm]
        else:
            return 0.0
```

## Creating the variables and the environment

```
reward_probabilities=[0.01, 1.0, 0.75, 0.99, 0.65, 1.0];
rewards=[95.0, 0.0, 25.5, 10.05, 5.45, 2.50];
environment = Env(reward_probabilities, rewards);
```

```
f"K_arms \t\t\t: {environment.k_arms}"
```

✓ 1s completed at 12:43 PM

MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_y5HpbzKrNYIiWivQSfjLDg#scrollTo=AdMsn8VsNmJb

Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

ext

```
plot(cum_rewards, label="avg rewards")
set_title("Cumulative Rewards")

fig.add_subplot(122)
bar([i for i in range(len(chosen_arms))], chosen_arms, label="chosen arms")
set_title("Chosen Actions")
```

an environment

```
Env(object):
    setting all the parameters as member variables of class
    __init__(self, reward_probabilities, rewards):
        self.reward_probabilities = reward_probabilities # probability of receiving reward from that arm
        self.rewards = rewards # acquired reward from that arm
        self.k_arms = len(rewards) # Number of arms

    choose_arm(self, arm):
        np.random.random() < self.reward_probabilities[arm]:
            return self.rewards[arm]
        else:
            return 0.0
```

Creating the variables and the environment

✓ 1s completed at 12:43 PM

o H e G F D

## MultiArmedBandit\_EpsilonGreedy.ipynb

File Insert Runtime Tools Help All changes saved

Comment



ext

RAM Disk

```
Rewards \t\t: {environment.rewards}

probabilities : 6
: [0.01, 1.0, 0.75, 0.99, 0.65, 1.0]
: [95.0, 0.0, 25.5, 10.05, 5.45, 2.5]
```

ing selective action

```
onment.choose_arm(3) for _ in range(10)]
, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05, 10.05]
```

## Exploration and Exploitation with Epsilon greedy algorithm

```
EpsilonGreedyAgent(object):
    setting all the parameters as member variables of class
    __init__(self, env, max_iterations=200, epsilon=0.1):
        If.env = env
        If.iterations = max_iterations
        If.epsilon = epsilon

    method that let the agent act within the environment
    act(self):
        values = np.zeros(self.env.k_arms) # Payout of each arm is set to 0
        m_rewards = np.zeros(self.env.k_arms) # Total rewards of each arm is set to 0
        m_counts = np.zeros(self.env.k_arms) # Number of times each arm is pulled
```

✓ 1s completed at 12:43 PM

## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



```

ext
class MultiArmedBandit:
    def __init__(self, k_arms, iterations):
        self.k_arms = k_arms
        self.iterations = iterations
        self.q_values = np.zeros(k_arms)
        self.arm_counts = np.zeros(k_arms)
        self.rewards = []
        self.cum_rewards = []

    def act(self):
        values = np.zeros(self.k_arms) # Payout of each arm is set to 0
        m_rewards = np.zeros(self.k_arms) # Total rewards of each arm is set to 0
        m_counts = np.zeros(self.k_arms) # Number of times each arm is pulled

        rewards = [] # list to store the actual rewards that agent makes
        cum_rewards = [] # Average of all the rewards

        for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm
            if np.random.random() < self.epsilon: # random action/exploration
                arm = np.random.choice(self.k_arms)

            else: # greedy action/exploitation
                arm = np.argmax(q_values) # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

            reward = self.env.choose_arm(arm)

            arm_rewards[arm] += reward # Update the values
            arm_counts[arm] += 1
            q_values[arm] = arm_rewards[arm]/arm_counts[arm]

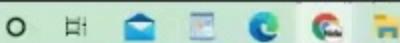
            rewards.append(reward) # append the values in list
            cum_rewards.append(sum(rewards)/ len(rewards))

        return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}

```

## Implementing Epsilon greedy algorithm

✓ 1s completed at 12:43 PM



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1IGICF0zNQF\_ayFj8\_6-2Xu4Axz-BXXR#scrollTo=R1qc-8-xrf5I

Gmail YouTube Maps YouTube bitspilani

11

View Insert Runtime Tools Help All changes saved

Comment

RAM Disk

ating the environment

ing a selective action

I

ng Exploration and Exploitation with epsilon greedy algorithm

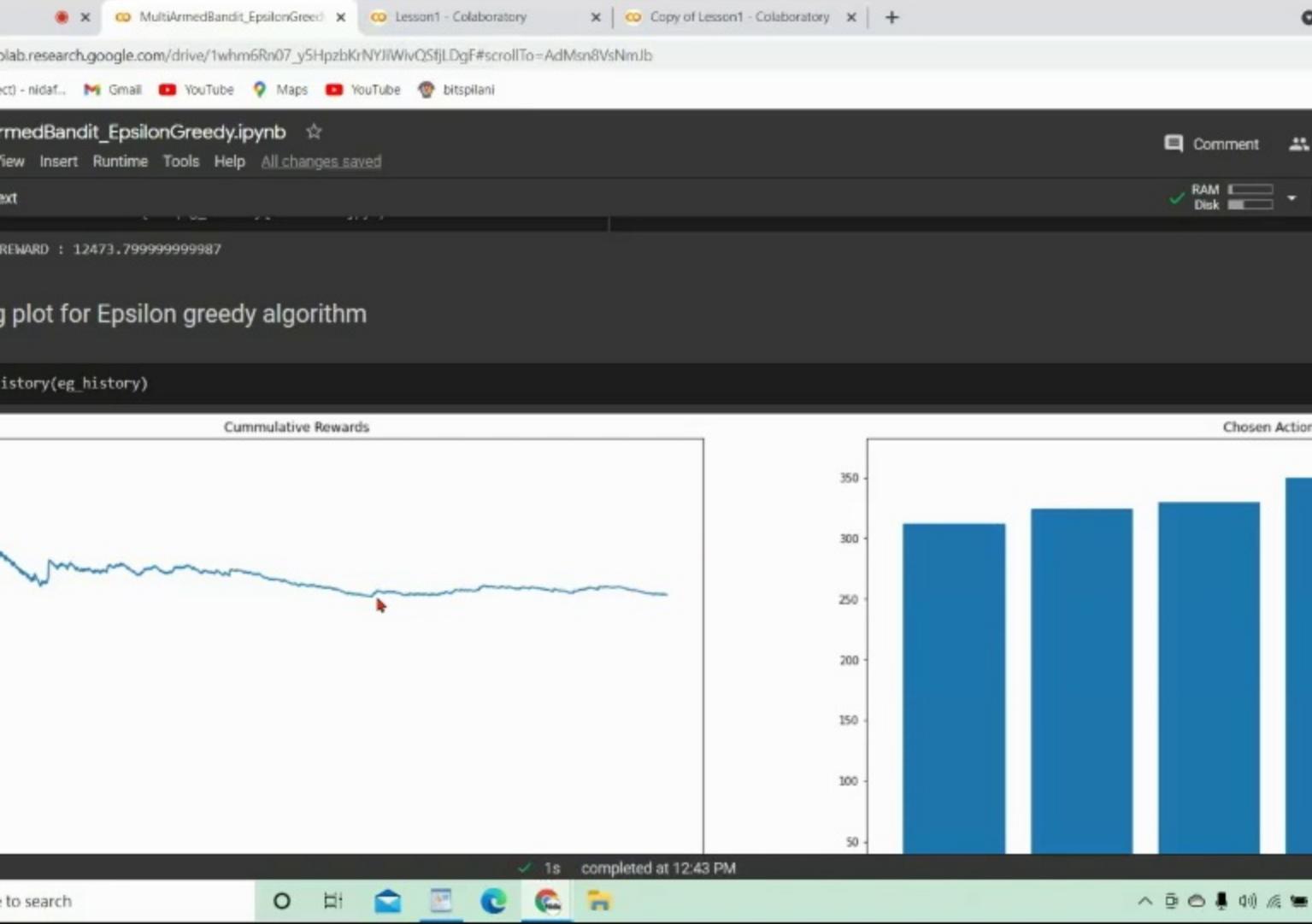
enting Epsilon greedy algorithm

g plot for Epsilon greedy algorithm

✓ 0s completed at 10:40 AM

to search





## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

View Insert Runtime Tools Help All changes saved

Comment



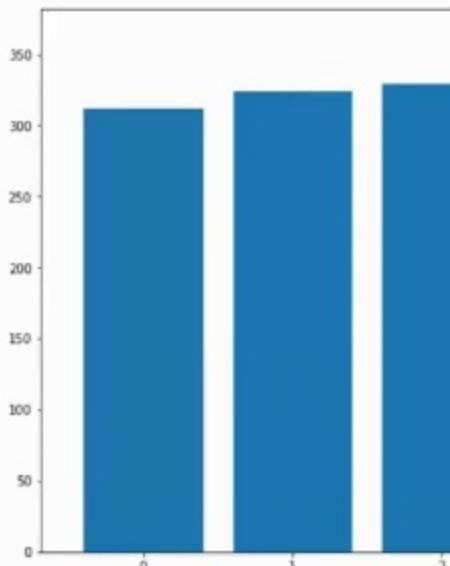
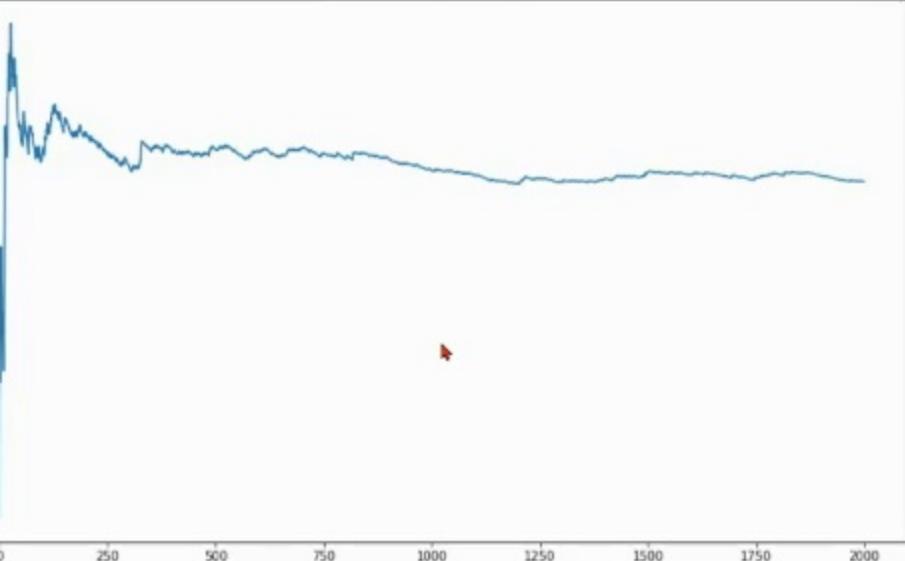
ext

RAM

Disk

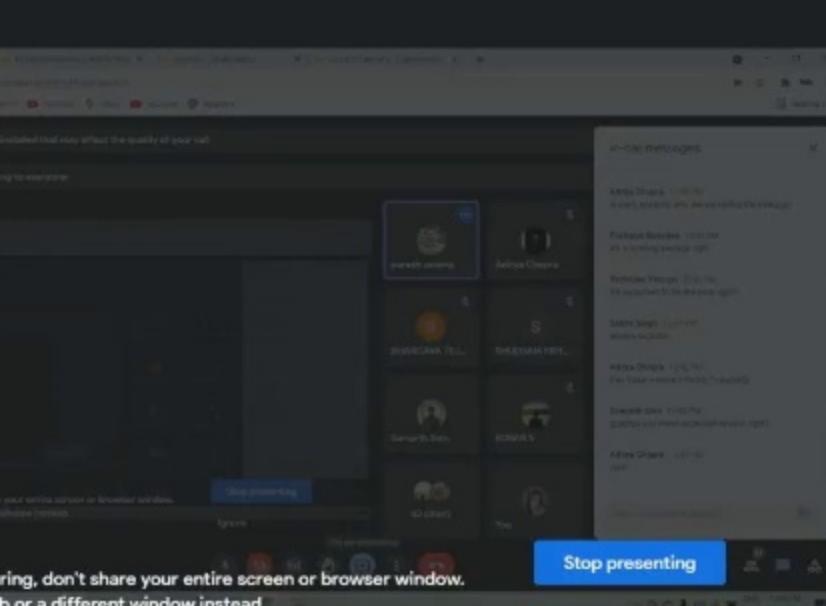
istory(eg\_history)

Cummulative Rewards



You have extensions installed that may affect the quality of your call

You're presenting to everyone



### In-call messages

Aditya Chopra 12:38 PM  
in cum\_rewards, why are we taking

Pratyush Banerjee 12:40 PM  
it's a running average right

Yashaswi Yenugu 12:41 PM  
it's supposed to be the sum right?

Saloni Singh 12:41 PM  
always exploits

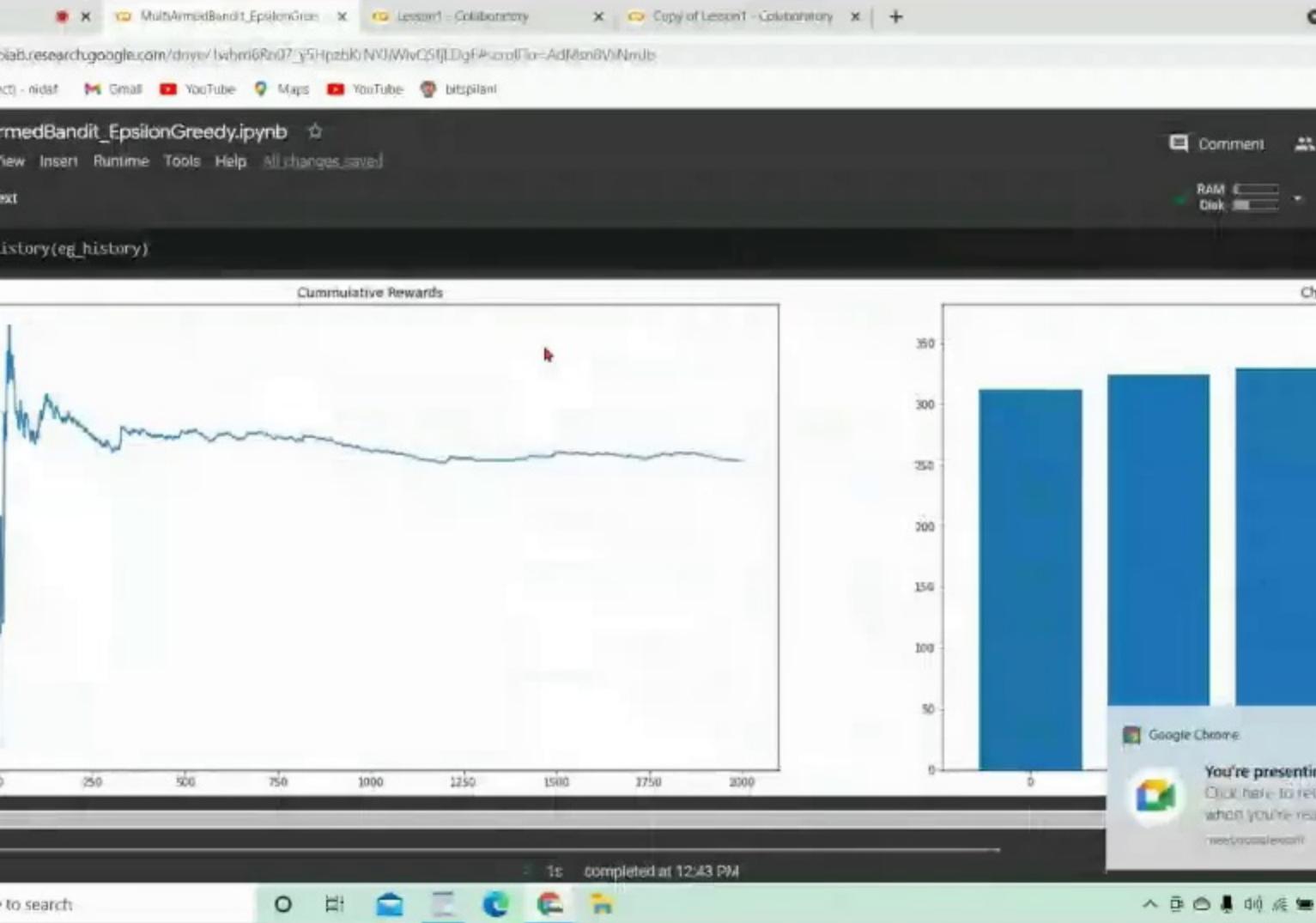
Aditya Chopra 12:42 PM  
Exp Value = reward Prob[] \* reward

Samarth Soni 12:43 PM  
@aditya you mean expected reward

Aditya Chopra 12:44 PM  
yeah

Send a message to everyone





## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment

...

```
method that let the agent act within the environment

def act(self):
    values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
    arm_rewards = np.zeros(self.env.k_arms)  # Total rewards of each arm is set to 0
    arm_counts = np.zeros(self.env.k_arms)   # Number of times each arm is pulled

    rewards = []                          # list to store the actual rewards that agent makes
    cum_rewards = []                      # Average of all the rewards

    for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

        if np.random.random() < self.epsilon: # random action/exploration
            arm = np.random.choice(self.env.k_arms)

        else:                                # greedy action/exploitation
            arm = np.argmax(q_values)          # argmax has a property that if 2 index has same qvalue then it chooses the lower index always

        reward = self.env.choose_arm(arm)

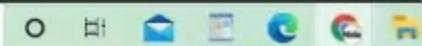
        arm_rewards[arm] += reward           # Update the values
        arm_counts[arm] += 1
        q_values[arm] = arm_rewards[arm]/arm_counts[arm]

        rewards.append(reward)              # append the values in list
        cum_rewards.append(sum(rewards)/ len(rewards))

    return {"arms": arm_counts, "rewards": rewards, "cum_rewards": cum_rewards}
```

## Implementing Epsilon greedy algorithm

✓ 1s completed at 12:43 PM



MultiArmedBandit\_EpsilonGreedy x Lesson1 - Colaboratory x | Copy of Lesson1 - Colaboratory x +

lab.research.google.com/drive/1whm6Rn07\_y5HpzbKrNYiWivQSfjLDgF#scrollTo=AdMsnaVsNmJb

ect) - nida... Gmail YouTube Maps YouTube bitspilani

MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

ext

## Exploration and Exploitation with Epsilon greedy algorithm

EpsilonGreedyAgent(object):

etting all the parameters as member variables of class

```
    __init__(self, env, max_iterations=200, epsilon=0.1):
        self.env = env
        self.iterations = max_iterations
        self.epsilon = epsilon
```

ethod that let the agent act within the environment

```
    act(self):
        values = np.zeros(self.env.k_arms) # Pay-off of each arm is set to 0
        total_rewards = np.zeros(self.env.k_arms) # Total rewards of each arm is set to 0
        n_counts = np.zeros(self.env.k_arms) # Number of times each arm is pulled

        rewards = [] # list to store the actual rewards that agent makes
        avg_rewards = [] # Average of all the rewards
```

r i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

```
    if np.random.random() < self.epsilon: # random action/exploration
        arm = np.random.choice(self.env.k_arms)
```

```
    else: # greedy action/exploitation
        arm = np.argmax(q_values) # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always
```

reward = self.env.choose\_arm(arm)

✓ 1s completed at 12:43 PM

o H e S C F

## MultiArmedBandit\_EpsilonGreedy.ipynb ☆

File Insert Runtime Tools Help All changes saved

Comment



ext

RAM Disk

## Using Exploration and Exploitation with Epsilon greedy algorithm



```
EpsilonGreedyAgent(object):

    Setting all the parameters as member variables of class
    __init__(self, env, max_iterations=200, epsilon=0.1):
        self.env = env
        self.iterations = max_iterations
        self.epsilon = epsilon

    Method that let the agent act within the environment
    act(self):
        values = np.zeros(self.env.k_arms)      # Payout of each arm is set to 0
        n_rewards = np.zeros(self.env.k_arms)    # Total rewards of each arm is set to 0
        n_counts = np.zeros(self.env.k_arms)     # Number of times each arm is pulled

        rewards = []                          # list to store the actual rewards that agent makes
        avg_rewards = []                     # Average of all the rewards

        for i in range(1, self.iterations + 1): # choose action using epsilon greedy algorithm

            if np.random.random() < self.epsilon: # random action/exploration
                arm = np.random.choice(self.env.k_arms)

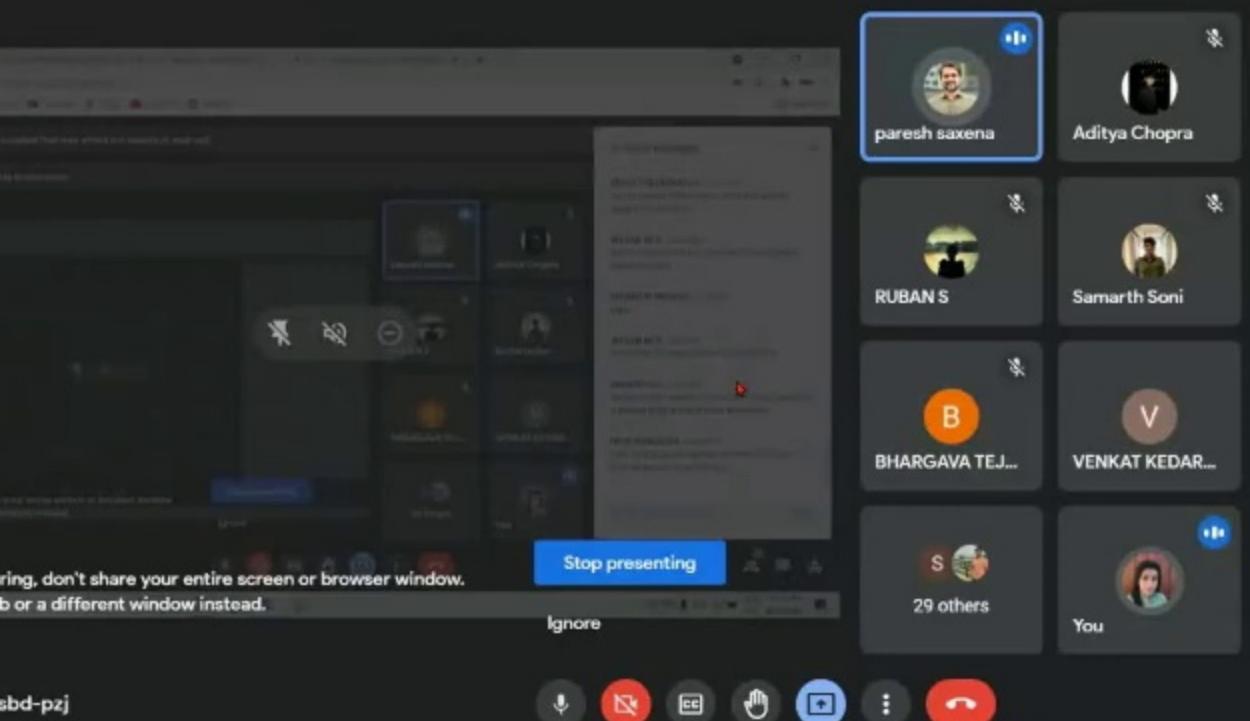
            else:                           # greedy action/exploitation
                arm = np.argmax(values)       # argmax has a property that if 2 index has same Qvalue then it chooses the lower index always

            reward = self.env.choose_arm(arm)
```

✓ 1s completed at 12:43 PM

have extensions installed that may affect the quality of your call.

You're presenting to everyone



## In-call messages

VENKAT KEDARNATH K. 12:50 PM  
can you please explain again when  
going for exploration

JEEVAN REJI 12:50 PM  
can you upload the class video which was recorded on thursday as well

SHUBHAM PRIYANK 12:50 PM  
yes sir

JEEVAN REJI 12:50 PM

Samarth Soni 12:51 PM  
can you please explain how we can  
in epsilon (high first and lower after)

FENIL BARDOLIYA 12:52 PM  
if we choose random epsilon between 0.01 will we get more efficiency?

Send a message to everyone



# Problem: Bandit Problems

innovate

achieve

- 3 armed bandit
- Epsilon Greedy Policy
- Identical Initial Q Values (Equals to 0)

Time Step	Action Chosen	Reward Obtained
1	1	1
2	1	0
3	3	2
4	2	4
5	2	-1
6	3	3

- Q1. In which time steps is the action definitely random ?  
Q2. In which time steps is the action definitely not random ?  
Q4. What is the most optimal action at time step 7 ?

# Problem: Bandit Problems

innovate

achieve

- 3 armed bandit
- Epsilon Greedy Policy
- Identical Initial Q Values (Equals to 0)

Time Step	Action Chosen	Reward Obtained
1	1	1
2	1	0
3	3	2
4	2	4
5	2	-1
6	3	3

- Q1. In which time steps is the action definitely random ?  
Q2. In which time steps is the action definitely not random ?  
Q4. What is the most optimal action at time step 7 ?

4	2	4
5	2	-1
6	3	3

- Q1. In which time steps is the action definitely random ?  
 Q2. In which time steps is the action definitely not random ?  
 Q4. What is the most optimal action at time step 7 ?

## Solution: Bandit Problems

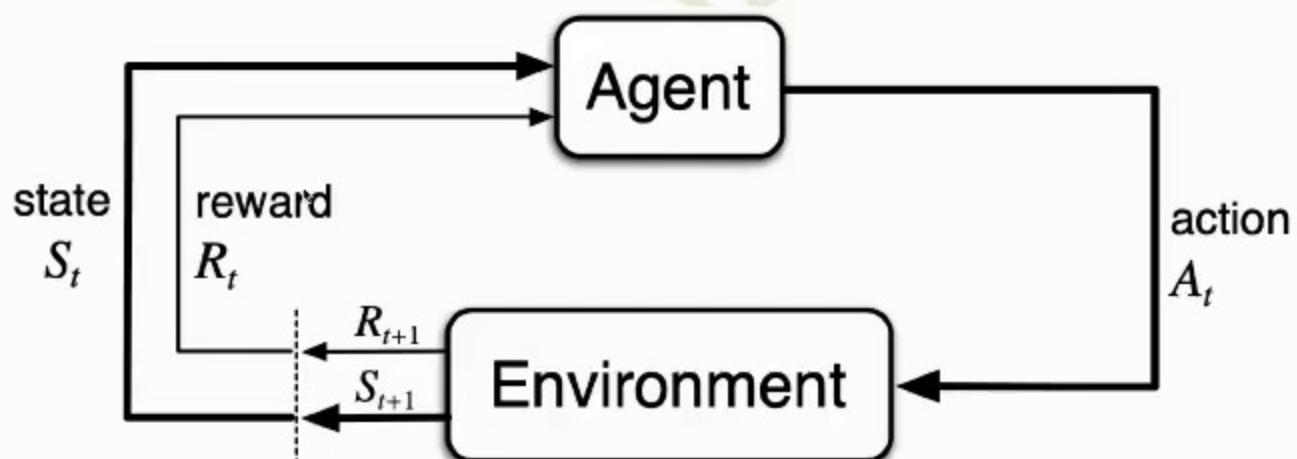
Time Step	Action Chosen	Reward Obtained
1	1	1
2	1	0
3	3	2
4	2	4
5	2	-1
6	3	3

Time Step	Action Chosen	Reward Obtained
1	1	1
2	1	0
3	3	2
4	2	4
5	2	-1
6	3	3

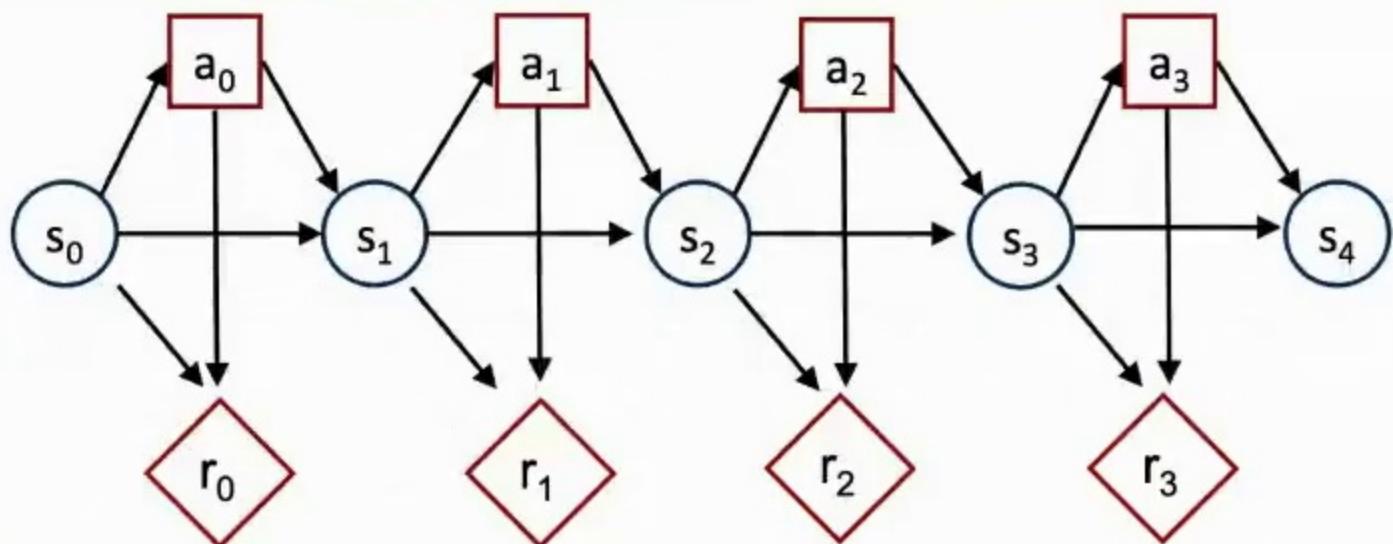
- Q1. In which time steps is the action definitely random ?  
 Q2. In which time steps is the action definitely not random ?  
 Q4. What is the most optimal action at time step 7 ?

Time	Action	Reward	Q(1)	Q(2)	Q(3)
1	1	1	0	0	0
2	1	0	1	0	0
3	3	2	0.5	0	0
4	2	4	0.5	0	2
5	2	-1	0.5	4	2
6	3	3	0.5	1.5	2
7	-	-	0.5	1.5	2.5

# Moving Towards Multiple States



# Markov Decision Process

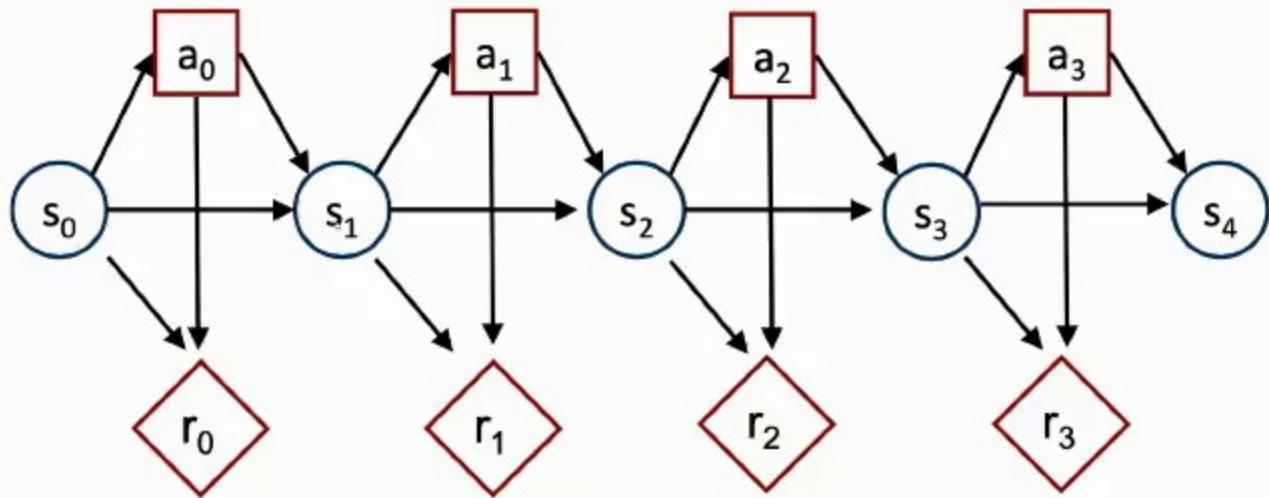


- States:  $s_t$
- Actions:  $a_t$
- Rewards:  $r_t$

- Uncertainty: stochastic process
- Time: sequential process
- Observability: fully observable states
- No Learning: complete model

$s_0 a_0 r_0 s_1 a_1 r_1 s_2 a_2 r_2 s_3 a_3 r_3 s_4$

# Markov Decision Process



- States:  $s_t$
- Actions:  $a_t$
- Rewards:  $r_t$

- Uncertainty: stochastic process
- Time: sequential process
- Observability: fully observable states
- No Learning: complete model
- Variable type: discrete (e.g., discrete states and actions)

$s_0 a_0 r_0 \quad s_1 a_1 r_1 \quad s_2 a_2 r_2$

$r_0$  $r_1$  $r_2$  $r_3$ 

- States:  $s_t$
- Actions:  $a_t$
- Rewards:  $r_t$

- Uncertainty: stochastic process
- Time: sequential process
- Observability: fully observable states
- No Learning: complete model
- Variable type: discrete (e.g., discrete states and actions)

 $s_0 a_0 r_0 \ s_1 a_1 r_1 \ s_2 a_2 r_2$ 

BITS Pilani, Hyderabad

## MDP - Dynamics

innovate

achieve

lead

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

$$s', s \in \mathcal{S}, r \in \mathcal{R}, \text{ and } a \in \mathcal{A}(s)$$

# MDP - Dynamics

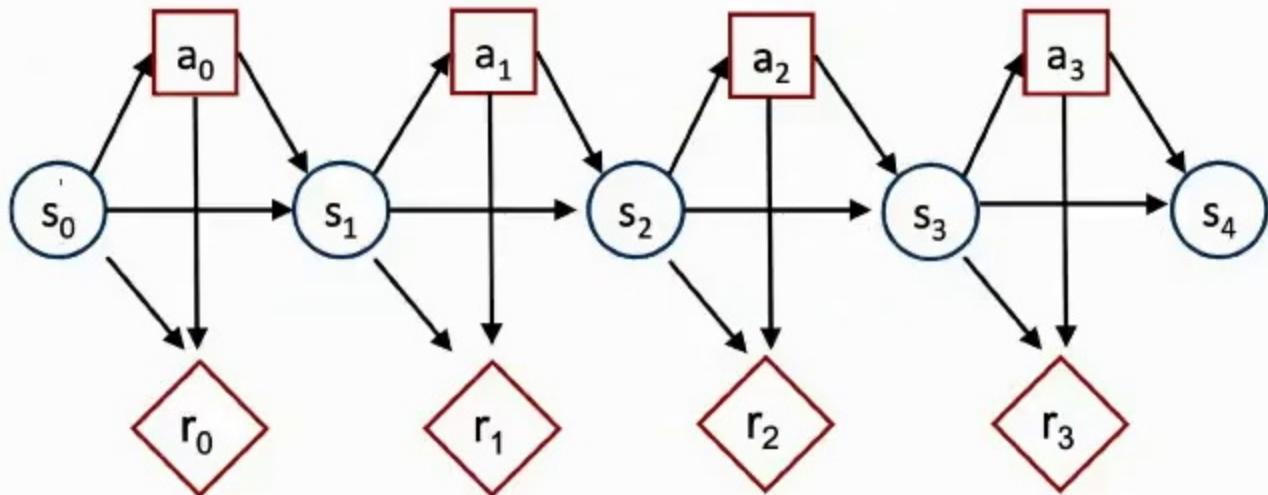
$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

$s', s \in \mathcal{S}, r \in \mathfrak{R}$ , and  $a \in \mathcal{A}(s)$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathfrak{R}} p(s', r | s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

$p()$  specifies a probability distribution for each choice of  $s$  and  $a$

# Markov Decision Process

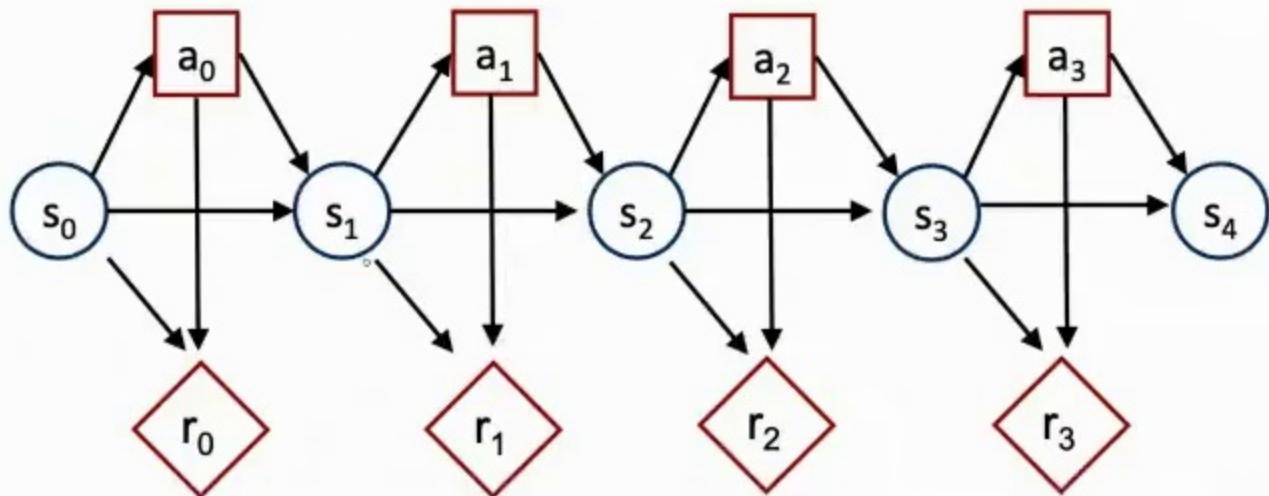


- States:  $s_t$
- Actions:  $a_t$
- Rewards:  $r_t$

- Uncertainty: stochastic process
- Time: sequential process
- Observability: fully observable states
- No Learning: complete model
- Variable type: discrete (e.g., discrete states and actions)

$s_0 a_0 r_0 \quad s_1 a_1 r_1 \quad s_2 a_2 r_2$

# Markov Decision Process



- States:  $s_t$
- Actions:  $a_t$
- Rewards:  $r_t$

- Uncertainty: stochastic process
- Time: sequential process
- Observability: fully observable states
- No Learning: complete model
- Variable type: discrete (e.g., discrete states and actions)

 $s_0a_0r_0 \quad s_1a_1r_1 \quad s_2a_2r_2$

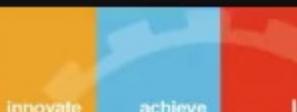
# Formulating and Understanding MDP Dynamics - I

- Two States: Battery High or Low
- Possible Actions
  - Go and Recharge
  - Wait for someone to bring can
  - Go and Pick empty cans
- $\mathcal{S} = \{\text{high, low}\}$
- $\mathcal{A} = \{\text{search, wait and recharge}\}$
- While low battery and searching, if battery finishes, robot is rescued and Recharged.



What is/are still missing for problem to be MDP ?

# Formulating and Understanding MDP Dynamics - I



- Two States: Battery High or Low
- Possible Actions
  - Go and Recharge
  - Wait for someone to bring can
  - Go and Pick empty cans
- $S=\{\text{high, low}\}$
- $A=\{\text{search, wait and recharge}\}$
- While low battery and searching, if battery finishes, robot is rescued and Recharged.



What is/are still missing for problem to be MDP ?

# MDP - Dynamics

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

$s', s \in \mathcal{S}, r \in \mathcal{R}$ , and  $a \in \mathcal{A}(s)$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

$p()$  specifies a probability distribution for each choice of  $s$  and  $a$

1	1	1
2	1	0
3	3	2
4	2	4
5	2	-1
6	3	3

- Q1. In which time steps is the action definitely random ?  
 Q2. In which time steps is the action definitely not random ?  
 Q4. What is the most optimal action at time step 7 ?

Time	Action	Reward	Q(1)	Q(2)	Q(3)
1	1	1	0	0	0
2	1	0	1	0	0
3	3	2	0.5	0	0
4	2	4	0.5	0	2
5	2	-1	0.5	4	2
6	3	3	0.5	1.5	2
7	-	-	0.5	1.5	2.5

# Moving Towards Multiple States

4	2	4
5	2	-1
6	3	3

- Q1. In which time steps is the action definitely random ?  
 Q2. In which time steps is the action definitely not random ?  
 Q4. What is the most optimal action at time step 7 ?

Time	Action	Reward	Q(1)	Q(2)	Q(3)
1	1	1	0	0	0
2	1	0	1	0	0
3	3	2	0.5	0	0
4	2	4	0.5	0	2
5	2	-1	0.5	4	2
6	3	3	0.5	1.5	2
7	-	-	0.5	1.5	2.5

•

## Moving Towards Multiple States

Time Step	Action Chosen	Reward Obtained
1	1	1
2	1	0
3	3	2
4	2	4
5	2	-1
6	3	3

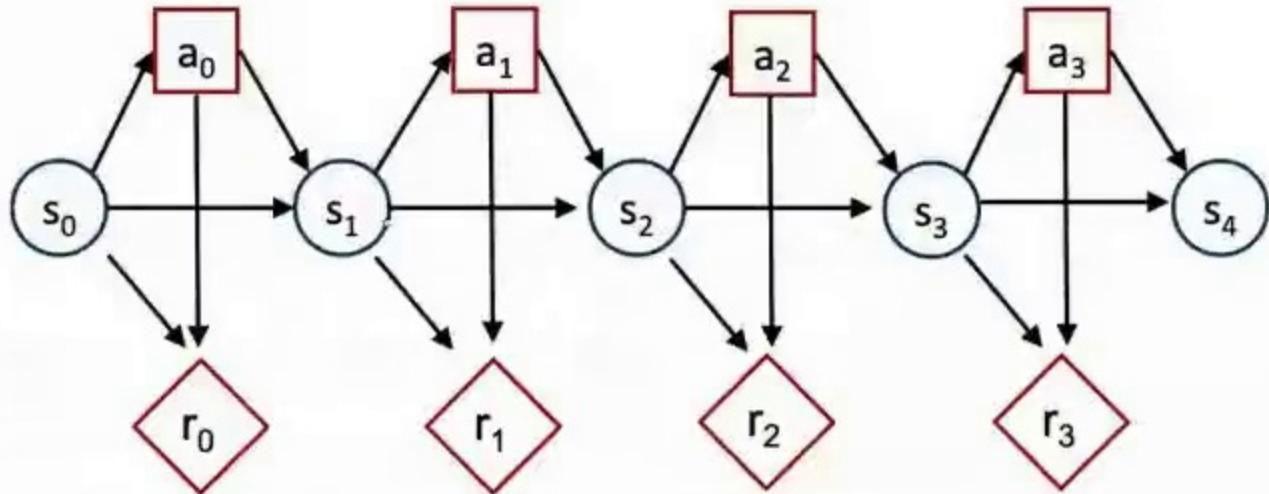
Q1. In which time steps is the action definitely random ?

Q2. In which time steps is the action definitely not random ?

Q4. What is the most optimal action at time step 7 ?

Time	Action	Reward	Q(1)	Q(2)	Q(3)
1	1	1	0	0	0
2	1	0	1	0	0
3	3	2	0.5	0	0
4	2	4	0.5	0	2
5	2	-1	0.5	4	2
6	3	3	0.5	1.5	2
7	-	-	0.5	1.5	2.5

# Markov Decision Process



- States:  $s_t$
- Actions:  $a_t$
- Rewards:  $r_t$

- Uncertainty: stochastic process
- Time: sequential process
- Observability: fully observable states
- No Learning: complete model
- Variable type: discrete (e.g., discrete states and actions)

$s_0 a_0 r_0 \quad s_1 a_1 r_1 \quad s_2 a_2 r_2$

# MDP - Dynamics

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

$s', s \in \mathcal{S}, r \in \mathcal{R}$ , and  $a \in \mathcal{A}(s)$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

$p()$  specifies a probability distribution for each choice of  $s$  and  $a$

$p()$  specifies a probability distribution for each choice of  $s$  and  $a$

# Formulating and Understanding MDP Dynamics - I

- Two States: Battery High or Low
- Possible Actions
  - Go and Recharge
  - Wait for someone to bring can
  - Go and Pick empty cans
- $S=\{\text{high, low}\}$
- $A=\{\text{search, wait and recharge}\}$
- While low battery and searching,  
if battery finishes, robot is rescued and

Wait

Go and  
Recharge



# Formulating and Understanding MDP Dynamics - I

innovate

achieve

lead

- Two States: Battery High or Low
- Possible Actions
  - Go and Recharge
  - Wait for someone to bring can
  - Go and Pick empty cans
- $S=\{\text{high, low}\}$
- $A=\{\text{search, wait and recharge}\}$
- While low battery and searching, if battery finishes, robot is rescued and Recharged.



What is/are still missing for problem to be MDP ?

# Formulating and Understanding MDP Dynamics - II

- Two States: Battery High or Low
- Possible Actions
  - Go and Recharge
  - Wait for someone to bring/put cans
  - Go and Pick empty cans
- $S=\{\text{high, low}\}$
- $A=\{\text{search, wait and recharge}\}$
- While low battery and searching, if battery finishes, robot is rescued and Recharged.



What is/are still missing for problem to be MDP ?

- Transition probabilities
- Rewards

# Formulating and Understanding MDP Dynamics - Transition Probabilities

innovate

achieve

state (s)	action (a)	new state (s')	p(s' s,a)
high	search	high	$\alpha$
high	search	low	$1-\alpha$
high	wait	high	$\gamma$
high	wait	low	$1-\gamma$
high	recharge	high	1
high	recharge	low	0
low	search	high	$1-\beta$
low	search	low	$\beta$
low	wait	high	0
low	wait	low	1
low	recharge	high	1
low	recharge	low	0

can be removed  
as well

$$\sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1$$

# Formulating and Understanding MDP Dynamics - II

- Two States: Battery High or Low
- Possible Actions
  - Go and Recharge
  - Wait for someone to bring/put cans
  - Go and Pick empty cans
- $S=\{\text{high, low}\}$
- $\mathcal{A}=\{\text{search, wait and recharge}\}$
- While low battery and searching, if battery finishes, robot is rescued and Recharged.



What is/are still missing for problem to

# Formulating and Understanding MDP Dynamics - Transition Probabilities

Innovate

achieve

state (s)	action (a)	new state (s')	$p(s' s,a)$
high	search	high	$\alpha$
high	search	low	$1-\alpha$
high	wait	high	$\gamma$
high	wait	low	$1-\gamma$
high	recharge	high	1
high	recharge	low	0
low	search	high	$1-\beta$
low	search	low	$\beta$
low	wait	high	0
low	wait	low	1
low	recharge	high	1
low	recharge	low	0

$$\sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1$$

for all  $s \in \mathcal{S}, a \in \mathcal{A}$

can be removed  
as well

- $\mathcal{S} = \{\text{high, low}\}$
- $\mathcal{A} = \{\text{search, wait and recharge}\}$
- While low battery and searching, if battery finishes, robot is rescued and Recharged.



What is/are still missing for problem to be MDP?

- Transition probabilities
- Rewards

## Formulating and Understanding MDP Dynamics - Transition Probabilities



state (s)	action (a)	new state (s')	$p(s' s,a)$
high	search	high	$\alpha$
high	search	low	$1-\alpha$
high	wait	high	$\gamma$
high	wait	low	$1-\gamma$
high	recharge	high	1
high	recharge	low	0
low	search	high	$1-\beta$
low	search	low	$\beta$
low	wait	high	0
low	wait	low	1
low	recharge	high	1
low	recharge	low	0

$$\sum_{s \in \mathcal{S}} \sum_{r \in \mathbb{R}} p(s', r | s, a) = 1, \quad \text{for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

can be removed

as well

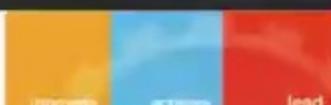
state (s)	action (a)	new state (s')	p(s' s,a)
high	search	high	$\alpha$
high	search	low	$1-\alpha$
high	wait	high	$\gamma$
high	wait	low	$1-\gamma$
high	recharge	high	1
high	recharge	low	0
low	search	high	$1-\beta$
low	search	low	$\beta$
low	wait	high	0
low	wait	low	1
low	recharge	high	1
low	recharge	low	0

$$\sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

can be removed  
as well

BITS Pilani, Hyderabad Campus

## Formulating and Understanding MDP Dynamics - Rewards



state (s)	action (a)	new state (s')	p(s' s,a)	r(s',s,a)
-----------	------------	----------------	-----------	-----------

# Formulating and Understanding MDP Dynamics - Transition Probabilities

innovate

achieve

lead

state (s)	action (a)	new state (s')	p(s' s,a)
high	search	high	$\alpha$
high	search	low	$1-\alpha$
high	wait	high	$\gamma$
high	wait	low	$1-\gamma$
high	recharge	high	1
high	recharge	low	0
low	search	high	$1-\beta$
low	search	low	$\beta$
low	wait	high	0
low	wait	low	1
low	recharge	high	1
low	recharge	low	0

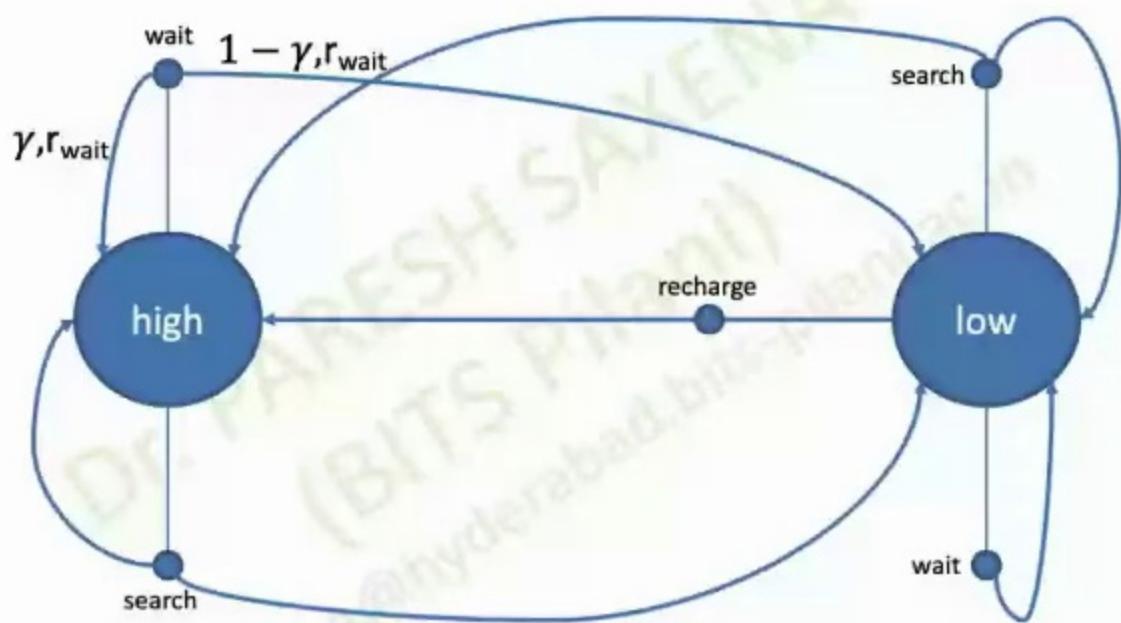
can be removed  
as well

$$\sum_{s \in S} \sum_{r \in R} p(s', r | s, a) = 1, \text{ for all } s \in S, a \in A(s)$$

# Formulating and Understanding MDP Dynamics – Transition Graph

Innovate

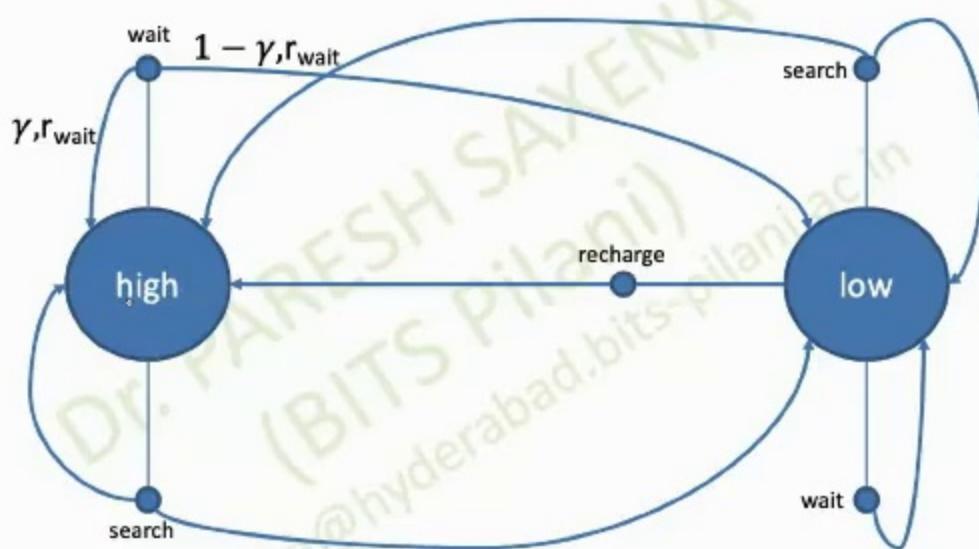
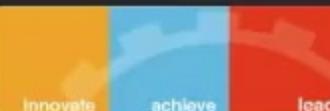
achieve



# MDP Dynamics - Rewards

state (s)	action (a)	new state (s')	p(s' s,a)	r(s',s,a)
high	search	high	$\alpha$	$r_{\text{search}}$
high	search	low	$1-\alpha$	$r_{\text{search}}$
high	wait	high	$\delta$	$r_{\text{wait}}$
high	wait	low	$1-\delta$	$r_{\text{wait}}$
high	recharge	high	1	0
high	recharge	low	0	0
low	search	high	$1-\beta$	-3 (rescue)
low	search	low	$\beta$	$r_{\text{search}}$
low	wait	high	0	$r_{\text{wait}}$
low	wait	low	1	$r_{\text{wait}}$
low	recharge	high	1	0
low	recharge	low	0	0

# Formulating and Understanding MDP Dynamics – Transition Graph



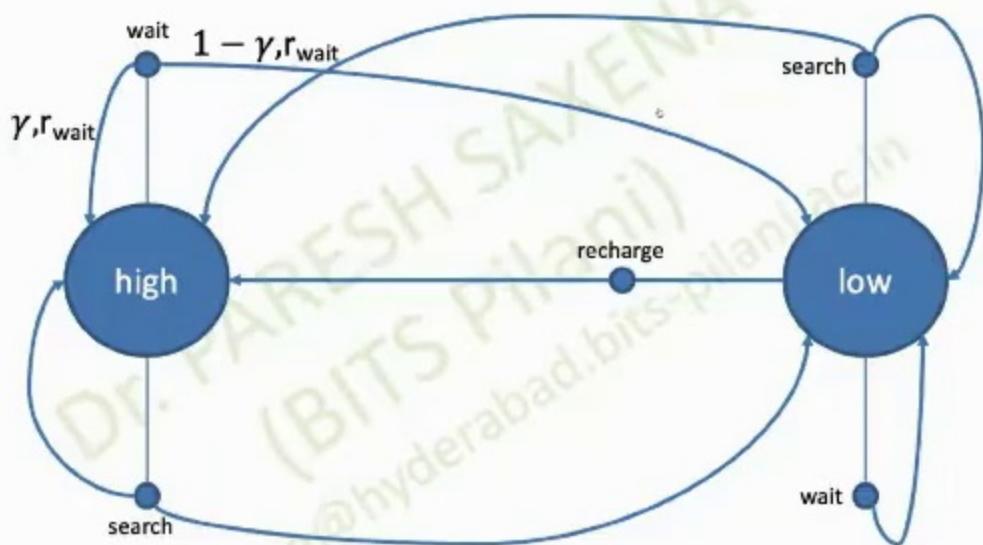
Complete it !!

# Formulating and Understanding MDP Dynamics – Transition Graph

innovate

achieve

lead



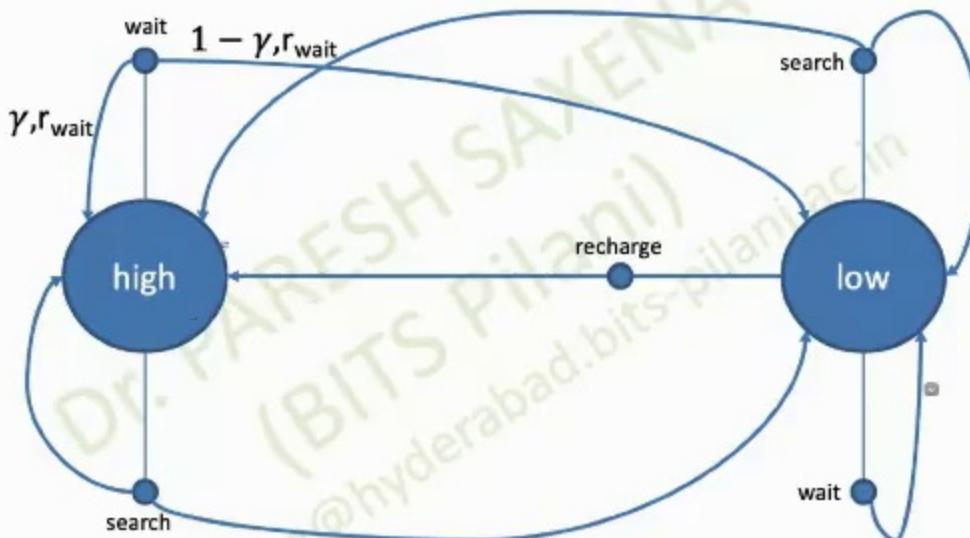
Complete it !!

# Formulating and Understanding MDP Dynamics – Transition Graph

innovate

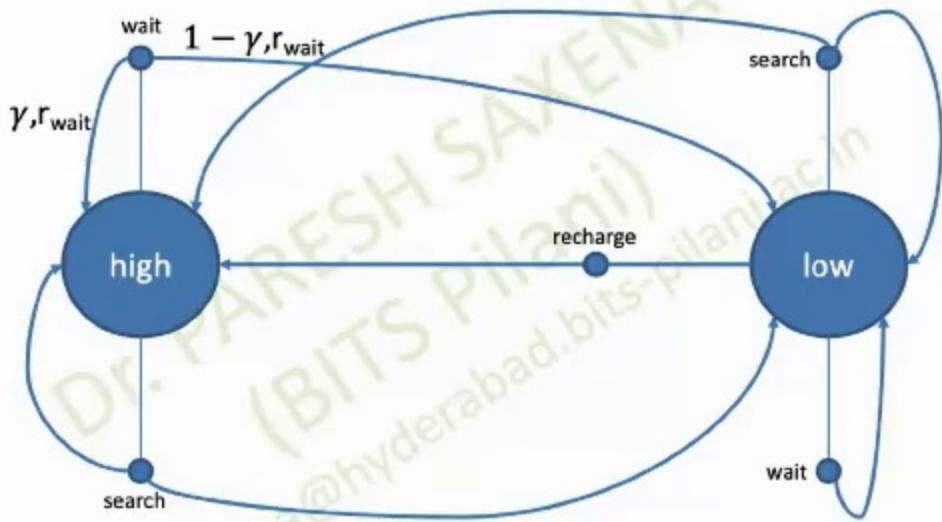
achieve

lead



Complete it !!

# Formulating and Understanding MDP Dynamics – Transition Graph



Complete it !!



# Episodic Tasks vs Continuing Tasks

## Continuing Tasks:

- Recycling Robot (one in previous slides)
- Pole Balancing



## Episodic Tasks:

- Game of Chess
- Maze Problems

# Discounted Rewards

Goal: Maximize sum of rewards:  $\sum_t R(s_t, a_t)$

Two issues:

- What if  $t \rightarrow \infty$  ?
- What if we want to give less weight to future rewards ?

Goal: Maximize sum of discounted rewards:

$$\text{Return } G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where  $0 < \gamma < 1$  is a discount factor/rate. Also,

$$G_t = R_{t+1} + \gamma G_{t+1}$$

where  $0 < \gamma < 1$  is a discount factor/rate. Also,

$$G_t = R_{t+1} + \gamma G_{t+1}$$

## Discounted Rewards: Exercise



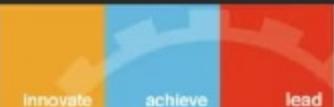
Goal: Maximize sum of discounted rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where  $\gamma < 1$  is a discount factor/rate. Also,

$$G_t = R_{t+1} + \gamma G_{t+1}$$

- Given  $\gamma = 0.8$ , first reward is 4 and all the subsequent rewards are 9. Calculate  $G_0, G_1$  and  $G_{100}$ .



# Discounted Rewards: Exercise

Goal: Maximize sum of discounted rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where  $\gamma < 1$  is a discount factor/rate. Also,

$$G_t = R_{t+1} + \gamma G_{t+1}$$

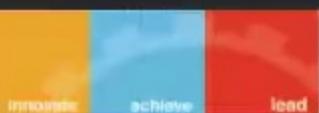
- Given  $\gamma = 0.8$ , first reward is 4 and all the subsequent rewards are 9. Calculate  $G_0, G_1$  and  $G_{100}$ .

where  $\gamma < 1$  is a discount factor/rate. Also,

$$G_t = R_{t+1} + \gamma G_{t+1}$$

- Given  $\gamma = 0.8$ , first reward is 4 and all the subsequent rewards are 9. Calculate  $G_0, G_1$  and  $G_{100}$ .

Answers:  $G_0=40$  and  $G_1=G_{100}=45$ .



## Solution

- $G_0 = R_1 + \gamma G_1$
- $G_1 = R_2 + \gamma G_2$
- $G_2 = R_3 + \gamma G_3$

# Solution

- $G_0 = R_1 + \gamma G_1$
- $G_1 = R_2 + \gamma G_2$
- $G_2 = R_3 + \gamma G_3$

.

- $G_n = R_{n+1} + \gamma G_{n+1}$

Rewriting  $G_0$ :

- $G_0 = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 \dots \dots$

Now,  $R_2 = R_3 \dots = R_n = 9$

- $G_0 = R_1 + \gamma R \frac{1}{1-\gamma} = 4 + 0.8 * 9 * (1/1-0.8) = 40$
- $G_2 = G_3 = \dots G_{100} = 45$

# Unified Notation for Episodic and Continuing Tasks



Goal: Maximize sum of discounted rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where  $\gamma < 0$  is a discount factor/rate. Also,

$$G_t = R_{t+1} + \gamma G_{t+1}$$

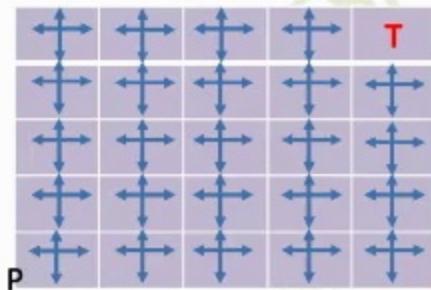
## Unified Notation:

$$G_t = \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

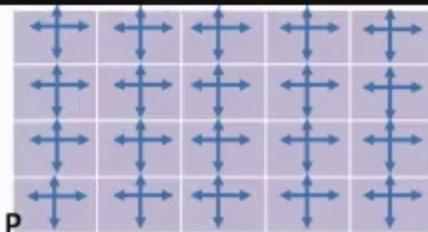
# Policy and Value Functions

**Policy: Mapping from States to Probabilities of selecting each possible action.**

**Example:**

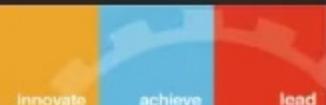


Policy: At each state P can go left, right, up and down with equal probability !!



Policy: At each state P can go left, right, up and down with equal probability !!

## State-Value Function



- The value function of a state  $s$  under a policy  $\pi$  is  $v_\pi(s)$ . It is the expected return when starting with  $s$  and following  $\pi$  after.
- Also refer to as state-value function for policy  $\pi$

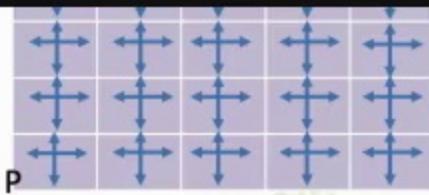
$$v_\pi(s) \doteq E_\pi[G_t | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s],$$

# State-Value Function

- The value function of a state  $s$  under a policy  $\pi$  is  $v_\pi(s)$ . It is the expected return when starting with  $s$  and following  $\pi$  after.
- Also refer to as state-value function for policy  $\pi$

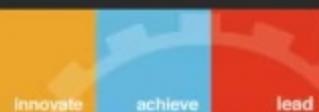
$$v_\pi(s) \doteq E_\pi[G_t | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s],$$

for all  $s \in \mathcal{S}$



Policy: At each state P can go left, right, up and down with equal probability !!

## State-Value Function



- The value function of a state  $s$  under a policy  $\pi$  is  $v_\pi(s)$ . It is the expected return when starting with  $s$  and following  $\pi$  after.
- Also refer to as state-value function for policy  $\pi$

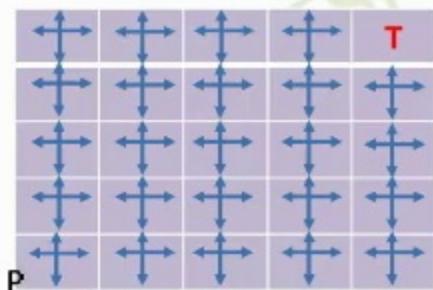
$$v_\pi(s) \doteq E_\pi[G_t | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s],$$

for all  $s \in \mathcal{S}$

# Policy and Value Functions

**Policy:** Mapping from States to Probabilities of selecting each possible action.

**Example:**



Policy: At each state P can go left, right, up and down with equal probability !!

# State-Value Function

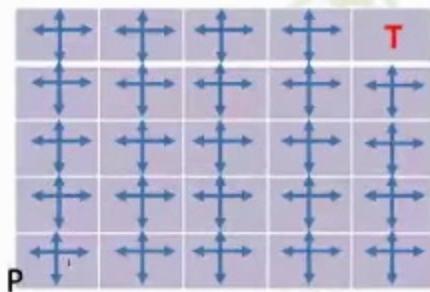
- The value function of a state  $s$  under a policy  $\pi$  is  $v_\pi(s)$ . It is the expected return when starting with  $s$  and following  $\pi$  after.
- Also refer to as state-value function for policy  $\pi$

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s],$$

for all  $s \in \mathcal{S}$

**Policy: Mapping from States to Probabilities of selecting each possible action.**

**Example:**



Policy: At each state P can go left, right, up and down with equal probability !!

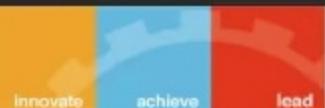


# State-Value Function

- The value function of a state  $s$  under a policy  $\pi$  is  $v_\pi(s)$ . It is the expected return when starting with  $s$  and following  $\pi$  after.
- Also refer to as state-value function for policy  $\pi$

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s],$$

for all  $s \in \mathcal{S}$



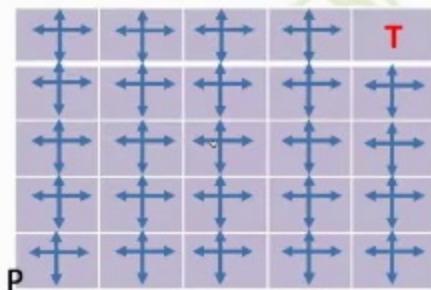
# Action-Value Function

- The action-value function is  $q_\pi(s, a)$
- It is the expected return starting from state  $s$ , taking action  $a$  and then following the policy  $\pi$ .

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]$$

**Policy: Mapping from States to Probabilities of selecting each possible action.**

**Example:**



Policy: At each state P can go left, right, up and down with equal probability !!



# State-Value Function

- The value function of a state  $s$  under a policy  $\pi$  is  $v_\pi(s)$ . It is the expected return when starting with  $s$  and following  $\pi$  after.
- Also refer to as state-value function for policy  $\pi$

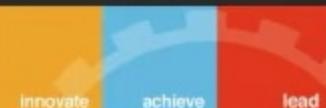
$$v_\pi(s) \doteq E_\pi[G_t | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s],$$

for all  $s \in \mathcal{S}$

# Action-Value Function

- The action-value function is  $q_\pi(s, a)$
- It is the expected return starting from state  $s$ , taking action  $a$  and then following the policy  $\pi$ .

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]$$



## Bellman Equation for $v_\pi$

- Relationship between the value of state and its successor state

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')], \end{aligned}$$

Over all actions      Transition Probabilities      Discounted successor states



## Action-Value Function

- The action-value function is  $q_\pi(s, a)$
- It is the expected return starting from state  $s$ , taking action  $a$  and then following the policy  $\pi$ .

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]$$



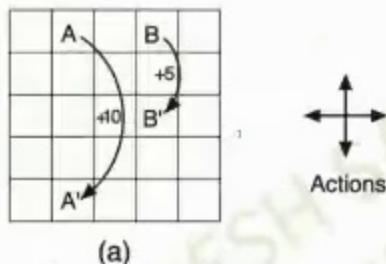
# Bellman Equation for $v_\pi$

- Relationship between the value of state and its successor state

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')], \end{aligned}$$

Over all actions      Transition Probabilities      Discounted successor states

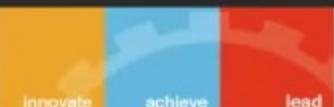
# Value Function Example



(b)

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

- Each cell one state,  $\gamma=0.9$
- Actions – Four: North, South, East, West
- Policy – Equal probability of taking any action
- Rewards:
  - -1 going out of grid
  - 0 elsewhere in the grid except
  - At A and B, where at A, reward is 10 and at B reward is 5.

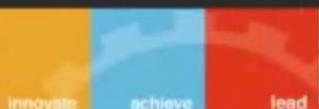


xx	xx	xx	xx	xx
xx	xx	3.2	xx	xx
xx	1.7	???	1.1	xx
xx	xx	-0.9	xx	xx
xx	xx	xx	xx	xx

**Remember:**

$$\begin{aligned}
 v_{\pi}(s) &\doteq E_{\pi}[G_t | S_t = s] \\
 &= \\
 &\sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_{\pi}(s')], 
 \end{aligned}$$

- Calculate ???
- Each cell one state,  $\gamma=0.9$
- Actions – Four: North, South, East, West
- Policy –  $p(\text{north})=1/2$ ,  $p(\text{South})=p(\text{East})=p(\text{West})$
- Rewards:
  - -1 going out of grid
  - 0 elsewhere in the grid



$$??? = 1.725$$

$$??? =$$

$$\frac{1}{2}(0+0.9 \times 3.2) + \frac{1}{6}(0+0.9 \times 1.1) + \frac{1}{6}(0+0.9 \times 1.7) + \frac{1}{6}(0+0.9 \times 0.9)$$

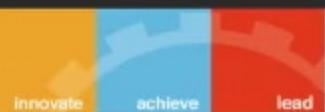
xx	xx	xx	xx	xx
xx	xx	3.2	xx	xx
xx	1.7	???	1.1	xx
xx	xx	-0.9	xx	xx
xx	xx	xx	xx	xx

**Remember:**

$$\begin{aligned}
 v_{\pi}(s) &\doteq E_{\pi}[G_t | S_t = s] \\
 &= \\
 &\sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_{\pi}(s')], 
 \end{aligned}$$

- Calculate ???
- Each cell one state,  $\gamma=0.9$
- Actions – Four: North, South, East, West
- Policy –  $p(\text{north})=1/2$ ,  $p(\text{South})=p(\text{East})=p(\text{West})$
- Rewards:
  - -1 going out of grid
  - 0 elsewhere in the grid

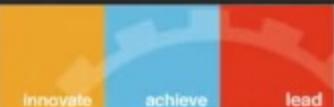




$$??? = 1.725$$

$$??? =$$

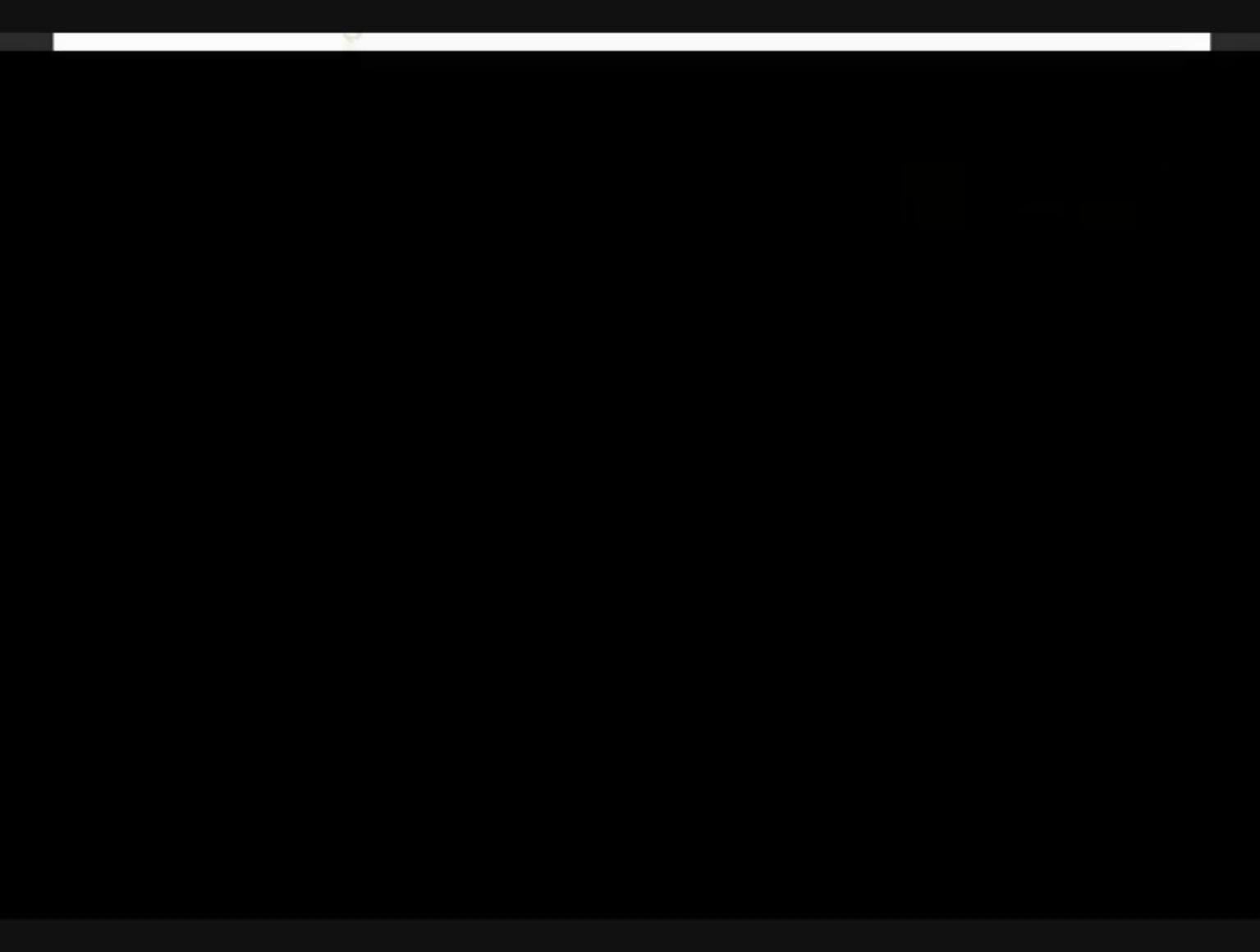
$$\frac{1}{2}(0+0.9 \times 3.2) + \frac{1}{6}(0+0.9 \times 1.1) + \frac{1}{6}(0+0.9 \times 1.7) + \frac{1}{6}(0+0.9 \times 0.9)$$



$$??? = 1.725$$

$$??? =$$

$$\frac{1}{2}(0+0.9 \times 3.2) + \frac{1}{6}(0+0.9 \times 1.1) + \frac{1}{6}(0+0.9 \times 1.7) + \frac{1}{6}(0+0.9 \times 0.9)$$



olab.research.google.com/drive/1Dbw0H0lquiZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=LIAo0y42NTK

 Gmail  YouTube  Maps  YouTube  YouTube

2 11/09/2021 ☆

View Insert Runtime Tools Help All changes saved

Comment

3

#### **Orientation for the environment**

### *hidden*

↑ ↓ ⏪

### **Take environment**

### **as hidden**



2 11/09/2021 ☆

View Insert Runtime Tools Help All changes saved

Comment

2

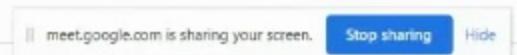
#### **Orientation for the environment**

### *hidden*

↑ ↓ ⏪

### **Take environment**

as hidden



Environments Documentation



Algorithms

Atari

Box2D

**Classic control**

MuJoCo

Robotics

Toy text EASY

Third party environments

## Classic control

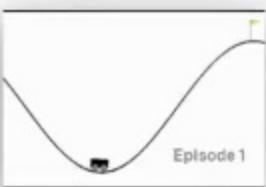
Control theory problems from the classic RL literature.



**Acrobot-v1**  
Swing up a two-link robot.



**CartPole-v1**  
Balance a pole on a cart.



**MountainCar-v0**  
Drive up a big hill.



**MountainCarContinuous-v0**



**Stop sharing**

- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=NwjL41ZjQr92

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help Last saved at 12:17 PM

Reconnect

## ntation for the environment

[openai.com/](https://openai.com/)

```
install torch
install torchvision
install numpy
install matplotlib
install gym
install box2d-py
install atari-py
```

```
gym
numpy as np
matplotlib.pyplot as plt
```

## ake environment

- Frozen Lake environment
- 0 reward per step: +1 for escaping

## environment

- Frozen Lake environment
- 0 reward per step; +1 for escaping
- Agent slides!
- Holes (H) terminate episode

ent:

The diagram illustrates a mapping from a 4x4 grid state space to a 16-state representation. On the left, a 4x4 grid shows states labeled F, H, and G. The top-left cell contains 'F'. The second column contains 'F' at row 1 and 'H' at row 2. The third column contains 'H' at row 1 and 'G' at row 4. The bottom-right cell contains 'G'. An arrow points from this grid to a 4x4 matrix on the right, which represents the 16 states. The matrix has columns labeled 0, 1, 2, 3 and rows labeled 0, 1, 2, 3. The values in the matrix are: Row 0: 0, 1, 2, 3. Row 1: 4, 5, 6, 7. Row 2: 8, 9, 10, 11. Row 3: 12, 13, 14, 15.

F	F		
F	H		
F	H		
F	G		

→

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

ce:

Left = 0

### Agent slides:

- Holes (H) terminate episode

ment:

A diagram illustrating state representation. On the left, there is a 4x3 grid of states. The first row contains three 'F's. The second row contains an 'H' at the second position. The third row contains two 'F's and one 'H'. The fourth row contains two 'F's and one 'G'. An arrow points from this grid to a larger 4x4 grid on the right.

F	F	F
H	F	H
F	F	H
F	F	G

→

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

pace:

- Left = 0
- Down = 1
- Right = 2
- Top = 3

- Colabora! x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HplqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=NwjL41ZjQr92

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help Last saved at 12:17 PM

Comment

Reconnect

ext

F	F	H	8	9	10	11
F	F	G	12	13	14	15

Space:

Left = 0

Down = 1

Right = 2

Top = 3

[https://github.com/openai/gym/blob/master/gym/envs/toy\\_text/frozen\\_lake.py](https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py)

ntiate our environment

the grid

[https://github.com/openai/gym/blob/master/gym/envs/toy\\_text/frozen\\_lake.py](https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py)

to search



- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=j\_5ja4YWrkj0

(ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

openai.com/

```
install torch
install torchvision
install numpy
install matplotlib
install gym
install box2d-py
install atari-py

gym
numpy as np
matplotlib.pyplot as plt
```

## Lake environment

- Frozen Lake environment
- 0 reward per step; +1 for escaping
- Agent slides!

lab.research.google.com/drive/1Dbw0HplqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=j\_5ja4YWrkj0

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help Saving...

ext

Comment

...

RAM Disk

[openai.com/](https://openai.com/)

```
install torch
install torchvision
install numpy
install matplotlib
install gym
install box2d-py
install atari-py
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (1.9.0+cu102)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch) (3.7.4.3)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (0.10.0+cu102)
Requirement already satisfied: pillow>=5.3.0 in /usr/local/lib/python3.7/dist-packages (from torchvision) (7.1.2)
Requirement already satisfied: torch==1.9.0 in /usr/local/lib/python3.7/dist-packages (from torchvision) (1.9.0+cu102)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvision) (1.19.5)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch==1.9.0->torchvision) (3.7.4.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.19.5)
```

```
gym
numpy as np
matplotlib.pyplot as plt
```

## make environment

Executing (8s) Cell > system() > \_system\_compat() > \_run\_command() > \_monitor\_process() > \_poll\_process()

- Colaboratory x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HplqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=j\_5jaFYWrkj0

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help

Comment

RAM Disk

```
ext
element already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (2.8.2)
element already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (2.4.7)
element already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (0.10.0)
element already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.3.1)
element already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.19.5)
element already satisfied: six in /usr/local/lib/python3.7/dist-packages (from cycler>=0.10->matplotlib) (1.15.0)
element already satisfied: gym in /usr/local/lib/python3.7/dist-packages (0.17.3)
element already satisfied:云cloudpickle<1.7.0,>=1.2.0 in /usr/local/lib/python3.7/dist-packages (from gym) (1.3.0)
element already satisfied: pyglet<=1.5.0,>=1.4.0 in /usr/local/lib/python3.7/dist-packages (from gym) (1.5.0)
element already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from gym) (1.4.1)
element already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.7/dist-packages (from gym) (1.19.5)
element already satisfied: future in /usr/local/lib/python3.7/dist-packages (from pyglet<=1.5.0,>=1.4.0->gym) (0.16.0)
ing box2d-py
loading box2d-py-2.3.8-cp37-cp37m-manylinux1_x86_64.whl (448 kB)
|██████████| 448 kB 5.1 MB/s
ing selected packages: box2d-py
successfully installed box2d-py-2.3.8

gym
numpy as np
matplotlib.pyplot as plt
```

## ake environment

- Frozen Lake environment

Executing (18s) Cell > system() > \_system\_compat() > \_run\_command() > \_monitor\_process() > \_poll\_process()

- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=168Ch1fJZSJ9

(- nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help

ext

RAM Disk

gym.make('FrozenLake-v0') #instantiate our environment

the grid

the grid

number of states

number of actions

rm action and see the result

---

agent

000 games by randomly sampling actions from the action space  
in percentage per ten games over time

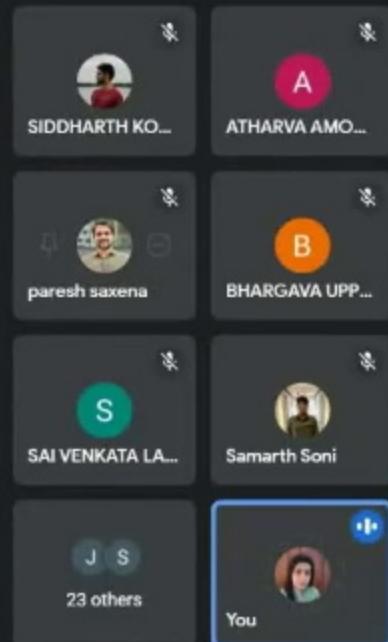
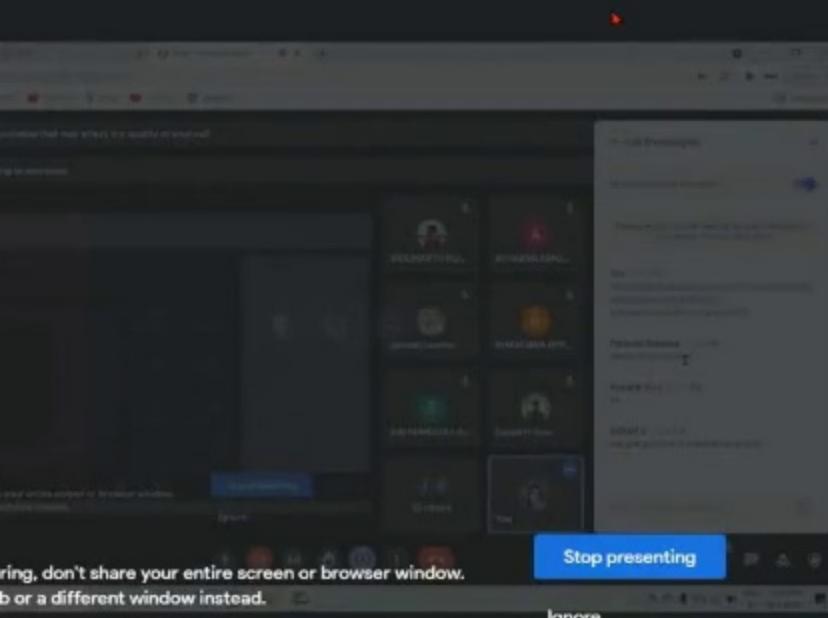
hidden

Deterministic Policy

000 games by following deterministic policy

Executing (0s) Cell > make() > make() > make() > load() > import\_module() > <module> > <module> > <module> > <module> > <module>

You're presenting to everyone



## In-call messages

Let everyone send messages

Messages can only be seen by people who are deleted when the account is.

You 12:15 PM  
<https://colab.research.google.com/u/2/p/8S6ctelSZEidyBZXonY?authuser=1#scrollTo=LIAc0v42NT>

Pratyush Banerjee 12:16 PM  
please disable editing

Samarth Soni 12:17 PM  
++

RUBAN S 12:29 PM

[Send a message to everyone](#)



- Colaboral x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lzWmN2TqZI

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

v.nA #total number of actions

rw action and see the result

```
set()
= 8
vaction, reward, done, prob) = env.step(
nder()
observation, reward, done, prob)
t)
False [{"prob": 0.3333333333333333}])
```

agent

000 games by randomly sampling actions from the action space  
in percentage per ten games over time

hidden

to search

Share with people and groups

No one has been added yet

Get link

<https://colab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyB...> Copy link

Anyone with the link ▾  
Anyone on the internet with this link can edit

Editor ▾

Done

0s completed at 12:28 PM

- Colabora! x Gym x Meet - rdx-udsw-bhn x

meet.google.com/rdx-udsw-bhn?pli=1&authuser=1

Gmail YouTube Maps YouTube bitspilani

You have extensions installed that may affect the quality of your call

You're presenting to everyone

Stop presenting

Ignore

J S  
23 others

You

In-call messages

Let everyone send messages

Messages can only be seen by people in the call. They are deleted when the call ends.

You 12:15 PM

<https://colab.research.google.com/u/2/jn8S6fctelSZEUdyBZXon?authuser=1#scrollTo=LIIApQy42N>

Pratyush Banerjee 12:16 PM

please disable editing

Samarth Soni 12:17 PM

++

RUBAN S 12:29 PM

can going from 6 to 2 I also be an ad

Send a message to everyone

31

o II e G

- Colabora: x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lzWmN2TqZI

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help All changes saved

ext

Comment

RAM Disk

nment:

F	F	F
H	F	H
F	F	H
F	F	G



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Space:

Left = 0

Down = 1

Right = 2

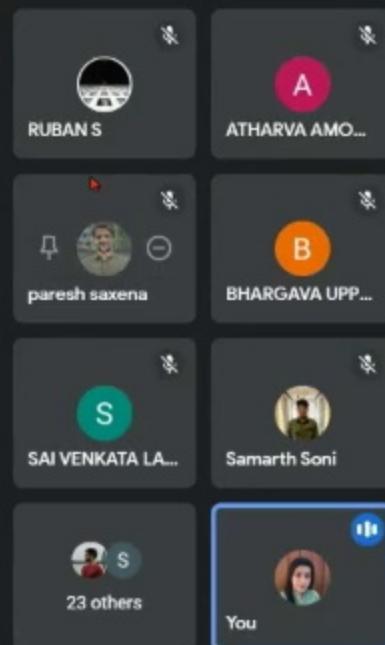
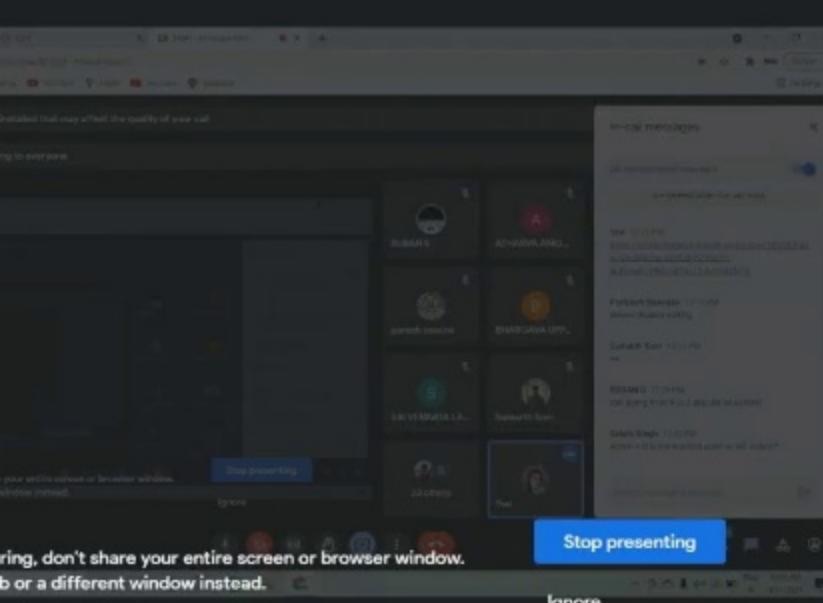
Top = 3

✓ 0s completed at 12:28 PM

to search



You're presenting to everyone.



## In-call messages

Let everyone send messages

are deleted when the call

You 12:15 PM  
<https://colab.research.google.com/gui?jo8S6ctelSZFUDyBZXonY?authuser=1#scrollTo=LILAc0v42N>

Pratyush Banerjee 12:16 PM  
please disable editing

Samarth Soni 12:17 PM  
++

RUBAN S 12:29 PM  
can going from 6 to 2 also be an ad

Saloni Singh 12:32 PM

Send a message to everyone

- Colabora: x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=U2HUVecfqYyY

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```
v.ns #total number of states

v.nA #total number of actions

v.P[6][1]

33333333333333, 5, 0.0, True),
33333333333333, 10, 0.0, False),
33333333333333, 7, 0.0, True)]

rm action and see the result

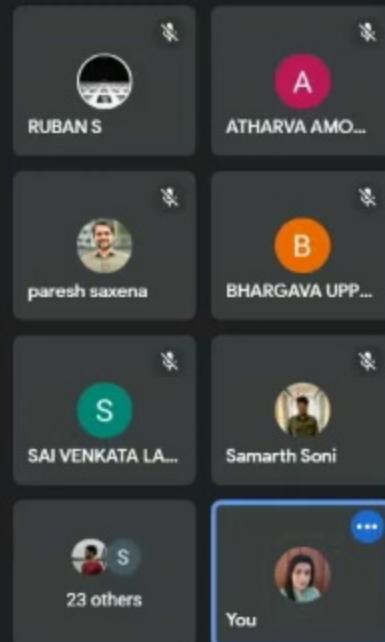
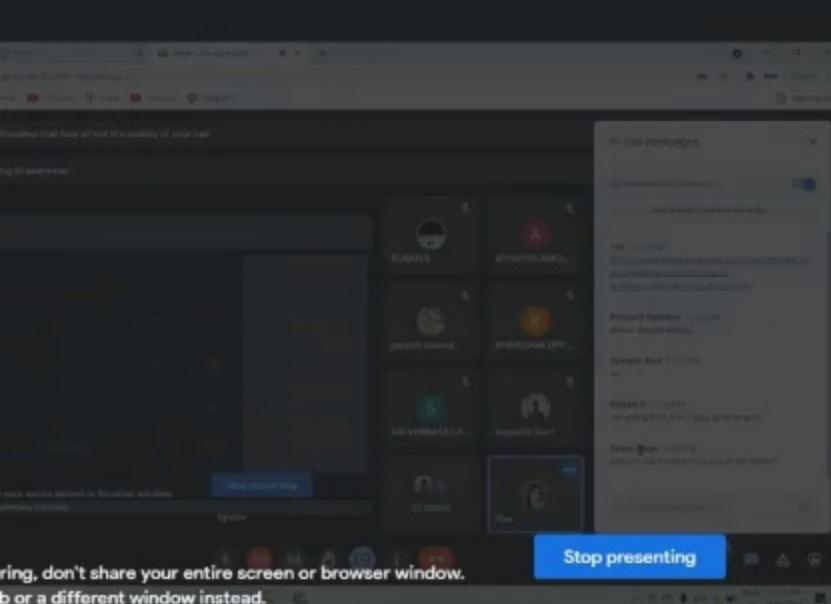
set()
- 0
vation, reward, done, prob) = env.step(action)
nder()
observation, reward, done, prob)

t)
```

✓ 0s completed at 12:32 PM

have extensions installed that may affect the quality of your call.

You're presenting to everyone.



## In-call messages

Let everyone send messages

are deleted when the call

You 12:15 PM  
<https://colab.research.google.com/gui?j=0S6ctelS7EUDyBZKx0Y?authuser=1#scrollTo=LILAc0v42N>

Pratyush Banerjee 12:16 PM  
please disable editing

Samarth Soni 12:17 PM  
++

RUBAN S 12:29 PM  
can going from 6 to 2 also be an ad

Saloni Singh 12:32 PM  
action = 0 is the starting point or le

Send a message to everyone



- Colabora! x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=U2HUVecfqYyY

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

F	F	F
H	F	H
F	F	H
F	F	G

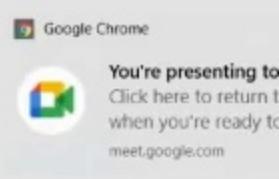
  

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Space:

Left = 0  
 Down = 1  
 Right = 2  
 Top = 3

[https://github.com/openai/gym/blob/master/gym/envs/toy\\_text/frozen\\_lake.py](https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py)



✓ 0s completed at 12:32 PM

to search



- Colabora! x Gym x Meet - rdx-udsw-bhn x

meet.google.com/rdx-udsw-bhn?pli=1&authuser=1

Gmail YouTube Maps YouTube bitspilani

You have extensions installed that may affect the quality of your call

You're presenting to everyone

In-call messages

Let everyone send messages

please disable editing

Samarth Soni 12:17 PM  
++

RUBAN S 12:29 PM  
can going from 6 to 2 also be an ac

Saloni Singh 12:32 PM  
action = 0 Is the starting point or le

what does action =0 denote?

Ankita Behera 12:34 PM  
Can anyone else access the notebook loading error for me

Saloni Singh 12:34 PM  
okay ma'am

Send a message to everyone

Stop presenting

Ignore

23 others

You

RUBAN S

ATHARVA AMO...

pares saxena

BHARGAVA UPP...

SAI VENKATA LA...

Samarth Soni

S

Ignore

31

- Colabora! x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=U2HUVecfqYyY

Gmail YouTube Maps YouTube bitsplani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

False {'prob': 0.3333333333333333}

agent

000 games by randomly sampling actions from the action space  
in percentage per ten games over time

hidden

Deterministic Policy

000 games by following deterministic policy  
in percentage per ten games over time

hidden

Google Chrome

You're presenting to

Click here to return to

meet.google.com

completed at 12:32 PM

- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HplqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnlKr5djk

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help

ext

000 games by randomly sampling actions from the action space  
in percentage per ten games over time

s=1000 #number of games  
t = [] #empty list to keep track of win percentages  
 = [] #empty list to keep track of scores

in range(n\_games) #loop over the number of games  
e = False #at the top of every episode reset the done flag  
 = env.reset() #reset the environment  
ore = 0 #set score for the episode to zero

ile not done #play the episode  
tion = env.actions\_space.sample() #TAKE RANDOM ACTION  
s, reward #set the new information  
ep track reward float

the end of every episode, append score for that episode to list of scores

every 10 games we keep track of win percentage

ut: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.

## Deterministic Policy

000 games by following deterministic policy  
in percentage per ten games over time

✓ 0s completed at 12:32 PM

to search

- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HplqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnlKr5djk

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help

ext

RAM Disk

rm action and see the result

```
set()
    = 0
env, reward, done, prob) = env.step(action)
render()
observation, reward, done, prob)
```

t)

False{'prob': 0.3333333333333333}

---

agent

000 games by randomly sampling actions from the action space  
in percentage per ten games over time

```
s=1000 #number of games
t = [] #empty list to keep track of win percentages
= [] #empty list to keep track of scores

i in range(n_games): #loop over the number of games
done = False #at the top of every episode reset the done flag
```

✓ 0s completed at 12:32 PM

- Colaboratory x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HplqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnlKr5djk

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext agent

000 games by randomly sampling actions from the action space  
in percentage per ten games over time

```
s=1000    #number of games
t = []    #empty list to keep track of win percentages
 = []    #empty list to keep track of scores

for i in range(n_games): #loop over the number of games
    done = False        #at the top of every episode reset the done flag
    s = env.reset()     #reset the environment
    score = 0            #set score for the episode to zero

    while not done:      #play the episode
        action = env.action_space.sample()          #TAKE RANDOM ACTION
        obs, reward, done, info = env.step(action)   #set the new information
        score += reward                            #keep track of the score for the episode

    scores.append(score)    #at the end of every episode, append score for that episode to list of scores

    if i%10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        print(win_pct)
        now()

print("Out: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.")

<ipython-input-11-cb52bbbe2afe>, line 5
for i in range(n_games): #loop over the number of games
```

0s completed at 12:44 PM

- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnlKr5djk

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help

ext

Comment

RAM Disk

000 games by randomly sampling actions from the action space  
in percentage per ten games over time

```
s=1000    #number of games
t = []    #empty list to keep track of win percentages
es = []    #empty list to keep track of scores

for i in range(n_games):    #loop over the number of games
    done = False            #at the top of every episode reset the done flag
    s = env.reset()          #reset the environment
    score = 0                #set score for the episode to zero

    while not done:          #play the episode
        action = env.action_space.sample()      #TAKE RANDOM ACTION
        obs, reward, done, info = env.step(action)    #set the new information
        score += reward                         #keep track of the score for the episode

    es.append(score)           #at the end of every episode, append score for that episode to list of scores

    if i%10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        ot(win_pct)
        ow()

out: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.
```

0s completed at 12:44 PM

- Colabora: x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJquZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnKr5djx

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

not done

```
s=1000    #number of games
t = []    #empty list to keep track of win percentages
 = []    #empty list to keep track of scores

for i in range(n_games): #loop over the number of games
    done = False          #at the top of every episode reset the done flag
    s = env.reset()        #reset the environment
    score = 0               #set score for the episode to zero

    while not done:         #play the episode
        action = env.action_space.sample()           #TAKE RANDOM ACTION
        obs, reward, done, info = env.step(action)   #set the new information
        score += reward                            #keep track of the score for the episode

    scores.append(score)      #at the end of every episode, append score for that episode to list of scores

    if score == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        print(win_pct)
        print(score)

print("out: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.")


Traceback (most recent call last)
in _input_14-198bc4207dc3> in <module>()
9
10 while not done:          #play the episode
11     action = env.action_space.sample()           #TAKE RANDOM ACTION
12     obs, reward, done, info = env.step(action)
13     score += reward
14
15 completed at 12:44 PM
```

- Colaboratory x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnlKr5djk

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help Saving...

ext

RAM Disk

```
s=1000    #number of games
t = []      #empty list to keep track of win percentages
- []       #empty list to keep track of scores

in range(n_games): #loop over the number of games
done = False        #at the top of every episode reset the done flag
s = env.reset()     #reset the environment
ore = 0             #set score for the episode to zero

ile not done:
    action = env.action_space.sample()           #TAKE RANDOM ACTION
    obs, reward, done, info = env.step(action)   #set the new information
    score += reward                            #keep track of the score for the episode

ores.append(score)      #at the end of every episode, append score for that episode to list of scores

if i%10 == 0:
    average = np.mean(scores[-10:])
    win_pct.append(average) # every 10 games we keep track of win percentage
    ot(win_pct)
    ow()

ut: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.
```

uteError Traceback (most recent call last)

on->input-15-dae6e955c242> in <module>()

9
8 while not done: #play the episode
7 action = env.action\_space.sample() #TAKE RANDOM ACTION
6 obs, reward, done, info = env.step(action) #set the new information
5 score += reward #keep track of the score for the episode
4
3
2
1
0
9 Os completed at 12:45 PM

- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnlKr5djk  
ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help

ext

RAM Disk

```
s=1000    #number of games
t = []    #empty list to keep track of win percentages
r = []    #empty list to keep track of scores

for i in range(n_games):    #loop over the number of games
    done = False            #at the top of every episode reset the done flag
    s = env.reset()          #reset the environment
    score = 0                #set score for the episode to zero

    while not done:          #play the episode
        action = env.action_space.sample()      #TAKE RANDOM ACTION
        obs, reward, done, info = env.step(action)  #get the new information
        score += reward                      #keep track of the score for the episode

    scores.append(score)        #at the end of every episode, append score for that episode to list of scores

    if i%10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        print(win_pct)
        os.system('cls')

print("out: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.")
```



✓ 0s completed at 12:46 PM

to search

2\_11/09/2021 ☆

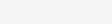
View Insert Runtime Tools Help All changes saved

Comment



ext

RAM Disk

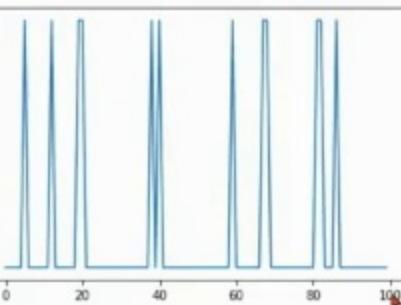


```
file not done:           #play the episode
action = env.action_space.sample()      #TAKE RANDOM ACTION
obs, reward, done, info = env.step(action)    #set the new information
score += reward                         #keep track of the score for the episode

scores.append(score)        #at the end of every episode, append score for that episode to list of scores

i%10 == 0:
average = np.mean(scores[-10:])
win_pct.append(average) # every 10 games we keep track of win percentage
ot(win_pct)
ow()
```

ut: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.



✓ Os completed at 12:46 PM

to search



- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnlKr5djk

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help

ext

in percentage per ten games over time

RAM Disk

```
s=1000    #number of games
t = []      #empty list to keep track of win percentages
            = []      #empty list to keep track of scores

for i in range(n_games): #loop over the number of games
    done = False          #at the top of every episode reset the done flag
    s = env.reset()        #reset the environment
    score = 0               #set score for the episode to zero

    while not done:         #play the episode
        action = env.action_space.sample()      #TAKE RANDOM ACTION
        obs, reward, done, info = env.step(action) #set the new information
        score += reward                      #keep track of the score for the episode

    scores.append(score)      #at the end of every episode, append score for that episode to list of scores

    if i%10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        ot(win_pct)
        cw()

out: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.
```

Executing (0s)

- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HplqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnlKr5djx

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

Comment

RAM Disk

000 games by randomly sampling actions from the action space  
in percentage per ten games over time

```
s=1000    #number of games
t = []    #empty list to keep track of win percentages
scores = []    #empty list to keep track of scores

for n_games in range(n_games):    #loop over the number of games
    done = False    #at the top of every episode reset the done flag
    s = env.reset()    #reset the environment
    score = 0    #set score for the episode to zero

    while not done:    #play the episode
        action = env.action_space.sample()    #TAKE RANDOM ACTION
        obs, reward, done, info = env.step(action)    #get the new information
        score += reward    #keep track of the score for the episode

    scores.append(score)    #at the end of every episode, append score for that episode to list of scores

    if n_games % 10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average)    #every 10 games we keep track of win percentage
        print(win_pct)
        print()

print("out: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.")
```

to search

✓ 0s completed at 12:46 PM



- Colabora: x Gym x Meet - rdx-udsw-bhn x +

meet.google.com/rdx-udsw-bhn?pli=1&authuser=1

(ct) - nida... Gmail YouTube Maps YouTube bitspilani

have extensions installed that may affect the quality of your call

You're presenting to everyone

In-call messages

Let everyone send messages

Pratyush Banerjee 12:34 PM we can access it

NEIL PARESH MEHTA 12:34 PM We want to take a particular action even a probability to go to other sta

ANJEL PATEL 12:34 PM use this link

<https://colab.research.google.com/gui/ZnR56ctalSZEUdyB7XonY>

BHARGAVA UPPULURI 12:34 PM @Ankita try opening it on Chrome. issue with Edge.

Ankita Behera 12:34 PM Works now, thanks

Send a message to everyone

Stop presenting

Ignore

RUBAN S

ATHARVA AMO...

pares saxena

BHARGAVA UPP...

SAI VENKATA LA...

Samarth Soni

A S

21 others

You

Microphone icon

Stop sharing icon

End call icon

Hand icon

Add participant icon

More options icon

Red end call icon

Search bar

Icons for Home, Task View, Mail, Google Sheets, Google Slides, Google Docs, and Google Chrome

Notification badge with number 29

2 11/09/2021 ☆

View Insert Runtime Tools Help All changes saved

 Comment

20

000 games by randomly sampling actions from the action space

in percentage per ten games over time

```

n_games = 1000    #number of games
t = []    #empty list to keep track of win percentages
s = []    #empty list to keep track of scores

for _ in range(n_games): #loop over the number of games
    done = False          #at the top of every episode reset the done flag
    s = env.reset()        #reset the environment
    score = 0              #set score for the episode to zero

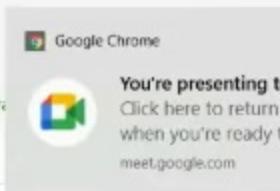
    while not done:        #play the episode
        action = env.action_space.sample()      #TAKE RANDOM ACTION | I
        obs, reward, done, info = env.step(action) #get the new information
        score += reward                      #keep track of the score for the episode

    scores.append(score)      #at the end of every episode, append score for that episode to list of scores

    if len(scores) % 10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        print(f'Episodes: {len(scores)}, Average Win Percentage: {average}, Last 10 Games Win Percentage: {win_pct[-1]}')
        print(f'Last 10 Games Win Percentage: {win_pct[-10:]}')

```

ut: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it v.



- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lygcnlKr5djk

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```
s=1000    #number of games
t = []    #empty list to keep track of win percentages
 = []    #empty list to keep track of scores

for i in range(n_games):    #loop over the number of games
done = False    #at the top of every episode reset the done flag
s = env.reset()    #reset the environment
score = 0    #set score for the episode to zero

while not done:    #play the episode
action = env.action_space.sample()    #TAKE RANDOM ACTION
obs, reward, done, info = env.step(action)    #get the new information
score += reward    #keep track of the score for the episode

scores.append(score)    #at the end of every episode, append score for that episode to list of scores

if i%10 == 0:
average = np.mean(scores[-10:])
win_pct.append(average)    # every 10 games we keep track of win percentage
print(win_pct)
print()

print("out: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.")



```

0s completed at 12:46 PM

- 0 reward per step; +1 for escaping
  - Agent slides!
  - Holes (H) terminate episode

onment:

F	F	F
H	F	H
F	F	H
F	F	G



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Space:

- Colabora: x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=TTt28blS66UD

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

Comment RAM Disk

↑ ↓ ↗ ↘

## Deterministic Policy

000 games by following deterministic policy  
in percentage per ten games over time

### Deterministic Policy:

```
= {0:1, 1:2, 2:1, 3:0, 4:1, 6:1, 8:2, 9:1, 10:1, 13:2, 14:2}
```

Down = 1 Right = 2 Top = 3

0 1 2 3

4 5 6 7

8 9 10 11

12 13 14 15

✓ 0s completed at 12:46 PM

to search

2\_11/09/2021 ★  
View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```
s=1000    #number of games
t = []    #empty list to keep track of win percentages
- []    #empty list to keep track of scores

in range(n_games): #loop over the number of games
done = False      #at the top of every episode reset the done flag
s = env.reset()   #reset the environment
score = 0          #set score for the episode to zero

ile not done:           #play the episode
action = env.action_space.sample()      #TAKE RANDOM ACTION
obs, reward, done, info = env.step(action)  #set the new information
score += reward           #keep track of the score for the episode

ores.append(score)        #at the end of every episode, append score for that episode to list of scores

i%10 == 0:
average = np.mean(scores[-10:])
win_pct.append(average) # every 10 games we keep track of win percentage
ot(win_pct)
ow()
```

ut: It simply falls in a hole and occasionally it does make it out. The maximum win percentage we see is around 10 percent and it varies from run to run.

- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=RncYXS7Q7HkD

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help Saving...

ext

RAM Disk

```
= {0:1, 1:2, 2:1, 3:0, 4:1, 6:1, 8:2, 9:1, 10:1, 13:2, 14:2}      # following deterministic policy
n_games = 1000      #number of games
t = []      #empty list to keep track of win percentages
s = []      #empty list to keep track of scores

for i in range(n_games):      #loop over the number of games
    done = False      #at the top of every episode reset the done flag
    s = env.reset()      #reset the environment
    score = 0      #set score for the episode to zero

    while not done:      #play the episode
        action = env.action_space.sample()      #TAKE RANDOM ACTION
        obs, reward, done, info = env.step(action)      #set the new information
        score += reward      #keep track of the score for the episode

    scores.append(score)      #at the end of every episode, append score for that episode to list of scores

    if i%10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average)      # every 10 games we keep track of win percentage
        print(win_pct)
        print()

print("gives pretty frequent win rate of 20 percent.Slightly better than the random agent.")


```

- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=RncYX57Q7HkD

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

RAM Disk

```
= {0:1, 1:2, 2:1, 3:0, 4:1, 6:1, 8:2, 9:1, 10:1, 13:2, 14:2}      # following deterministic policy
n_games = 1000      #number of games
t = []      #empty list to keep track of win percentages
s = []      #empty list to keep track of scores

for i in range(n_games):      #loop over the number of games
    done = False      #at the top of every episode reset the done flag
    s = env.reset()      #reset the environment
    score = 0      #set score for the episode to zero

    while not done:      #play the episode
        action = env.action_space.sample()      #TAKE RANDOM ACTION
        obs, reward, done, info = env.step(action)      #set the new information
        score += reward      #keep track of the score for the episode
    scores.append(score)      #append score for that episode to list of scores
    if i % 10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average)      # every 10 games we keep track of win percentage
    ot(win_pct)
    ow()

gives pretty frequent win rate of 20 percent.Slightly better than the random agent.
```

- Colaboratory x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=RncYX57Q7HkD

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help

Comment

RAM Disk

```
t = [] #empty list to keep track of win percentages
s = [] #empty list to keep track of scores

for i in range(n_games): #loop over the number of games
    done = False #at the top of every episode reset the done flag
    s = env.reset() #reset the environment
    score = 0 #set score for the episode to zero

    while not done: #play the episode
        action = policy(s)
        obs, reward, done, info = env.step(action) #set the new information
        score += reward #keep track of the score for the episode

    scores.append(score) #at the end of every episode, append score for that episode to list of scores

    if i%10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        print(win_pct)
        show()

gives pretty frequent win rate of 20 percent. Slightly better than the random agent.
```

✓ 0s completed at 12:46 PM

to search



- Colaboratory x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HplqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=RncYX57Q7HkD

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help

ext

RAM Disk

↑ ↓ ↻

```
= {0:1, 1:2, 2:1, 3:0, 4:1, 6:1, 8:2, 9:1, 10:1, 13:2, 14:2}      # following deterministic policy
n_games = 1000      #number of games
t = []      #empty list to keep track of win percentages
s = []      #empty list to keep track of scores

for i in range(n_games):      #loop over the number of games
    done = False      #at the top of every episode reset the done flag
    s = env.reset()      #reset the environment
    score = 0      #set score for the episode to zero

    while not done:      #play the episode
        action = policy[obs]
        obs, reward, done, info = env.step(action)
        score += reward

    scores.append(score)      #at the end of every episode, append score for that episode to list of scores

    if i % 10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average)      # every 10 games we keep track of win percentage
        print(win_pct)
        print()

gives pretty frequent win rate of 20 percent.slightly better than the random agent.
```

- Colaboratory x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=RncYX57Q7HkD

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

File Insert Runtime Tools Help All changes saved

RAM Disk

```
ext
- {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25} # FOLLOWING DETERMINISTIC POLICY
n_games = 1000 #number of games
t = [] #empty list to keep track of win percentages
scores = [] #empty list to keep track of scores

for i in range(n_games): #loop over the number of games
    done = False #at the top of every episode reset the done flag
    state = env.reset() #reset the environment
    score = 0 #set score for the episode to zero

    while not done: #play the episode
        action = policy[state]
        state, reward, done, info = env.step(action) #get the new information
        score += reward #keep track of the score for the episode

    scores.append(score) #at the end of every episode, append score for that episode to list of scores

    if i % 10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        print(f"Episode {i}: Win Percentage: {average:.2f}%")
        print(f"Score: {score}, Average Score: {average:.2f}, Last 10 Games: {scores[-10:]}")

print("gives pretty frequent win rate of 20 percent.Slightly better than the random agent.")
```

✓ 0s completed at 12:46 PM

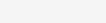
2\_11/09/2021

View Insert Runtime Tools Help All changes saved

Comment



RAM Disk



```
# [0:1, 1:2,2:1,3:0,4:1,6:1,8:2,9:1,10:1,13:2,14:2]    # following deterministic policy
n_games = 1000      #number of games
t = []             #empty list to keep track of win percentages
s = []             #empty list to keep track of scores

for i in range(n_games): #loop over the number of games
    done = False        #at the top of every episode reset the done flag
    s = env.reset()     #reset the environment
    score = 0            #set score for the episode to zero

    while not done:      #play the episode
        action = policy[s]
        obs, reward, done, info = env.step(action) #set the new information
        score += reward                         #keep track of the score for the episode

    scores.append(score)          #at the end of every episode, append score for that episode to list of scores

    if i % 10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        ot(win_pct)
        os()
```

gives pretty frequent win rate of 20 percent. Slightly better than the random agent.

- Colaboratory x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=RncYX57Q7HkD

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help

RAM Disk

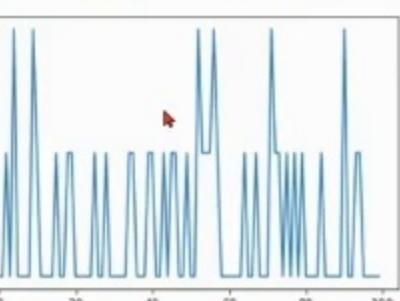
```
score = 0 #set score for the episode to zero

while not done: #play the episode
    action = policy[obs]
    obs, reward, done, info = env.step(action) #set the new information
    score += reward #keep track of the score for the episode

    scores.append(score) #at the end of every episode, append score for that episode to list of scores

    if i%10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
    i+=1
    print(i)

print("gives pretty frequent win rate of 20 percent.Slightly better than the random agent.")


```

✓ 0s completed at 12:53 PM

- Colaboratory x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=RncYX57Q7HkD

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```
= [0:1, 1:2, 2:1, 3:0, 4:1, 6:1, 8:2, 9:1, 10:1, 13:2, 14:2]      # following deterministic policy
s=1000    #number of games
t = []    #empty list to keep track of win percentages
s = []    #empty list to keep track of scores

for i in range(n_games):    #loop over the number of games
    done = False            #at the top of every episode reset the done flag
    s = env.reset()          #reset the environment
    score = 0                #set score for the episode to zero

    while not done:          #play the episode
        action = policy[s]
        obs, reward, done, info = env.step(action)    #set the new information
        score += reward                         #keep track of the score for the episode

    scores.append(score)        #at the end of every episode, append score for that episode to list of scores

    if i%10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average) # every 10 games we keep track of win percentage
        print(win_pct)
        show()

print("gives pretty frequent win rate of 20 percent.slightly better than the random agent.")
```

✓ 0s completed at 12:53 PM

- Colabora! x Gym x Meet - rdx-udsw-bhn x

meet.google.com/rdx-udsw-bhn?pli=1&authuser=1

(ct) - nida... Gmail YouTube Maps YouTube bitspilani

have extensions installed that may affect the quality of your call

You're presenting to everyone

RUBAN S ATHARVA AMO...  
A B  
paresh saxena BHARGAVA UPP...  
A Samarth Soni  
Ankita Behera  
J S 19 others You

In-call messages

Let everyone send messages

Pratyush Banerjee 12:34 PM we can access it

NEIL PARESH MEHTA 12:34 PM We want to take a particular action even a probability to go to other sta

ANJEL PATEL 12:34 PM use this link

<https://colab.research.google.com/gui/jpR5ictalSZEUdyB7XonY>

BHARGAVA UPPULURI 12:34 PM @Ankita try opening it on Chrome. issue with Edge.

Ankita Behera 12:34 PM Works now, thanks

Send a message to everyone

Stop presenting

Ignore

Microphone icon

Red video camera icon

Document icon

Hand icon

Blue square icon

Red phone receiver icon

More options icon

27 notifications

Search bar

Home icon

Back icon

Forward icon

Stop icon

Refresh icon

Network icon

Battery icon

Speaker icon

- Colaboral x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HIpJquZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=RncYX57Q7HkD

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```
= [0:1, 1:2, 2:1, 3:0, 4:1, 6:1, 8:2, 9:1, 10:1, 13:2, 14:2]      # following deterministic policy
n=1000      #number of games
t = []      #empty list to keep track of win percentages
s = []      #empty list to keep track of scores

for n_games in range(n_games):      #loop over the number of games
    done = False      #at the top of every episode reset the done flag
    s = env.reset()      #reset the environment
    score = 0      #set score for the episode to zero

    while not done:      #play the episode
        action = policy[s]
        obs, reward, done, info = env.step(action)      #set the new information
        score += reward      #keep track of the score for the episode

    scores.append(score)      #at the end of every episode, append score for that episode to list of scores

    if n % 10 == 0:
        average = np.mean(scores[-10:])
        win_pct.append(average)      # every 10 games we keep track of win percentage
        print(win_pct)
        show()

print("gives pretty frequent win rate of 20 percent.slightly better than the random agent.")
```

✓ 0s completed at 12:53 PM

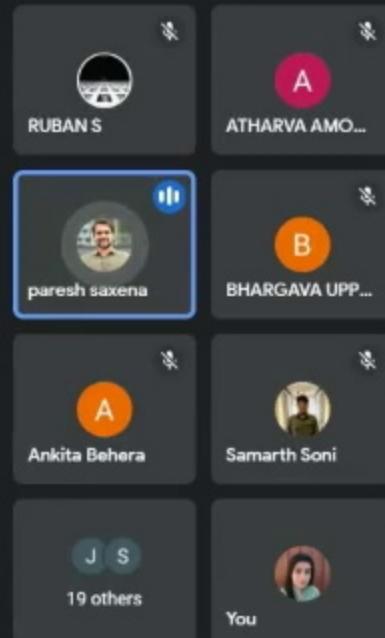
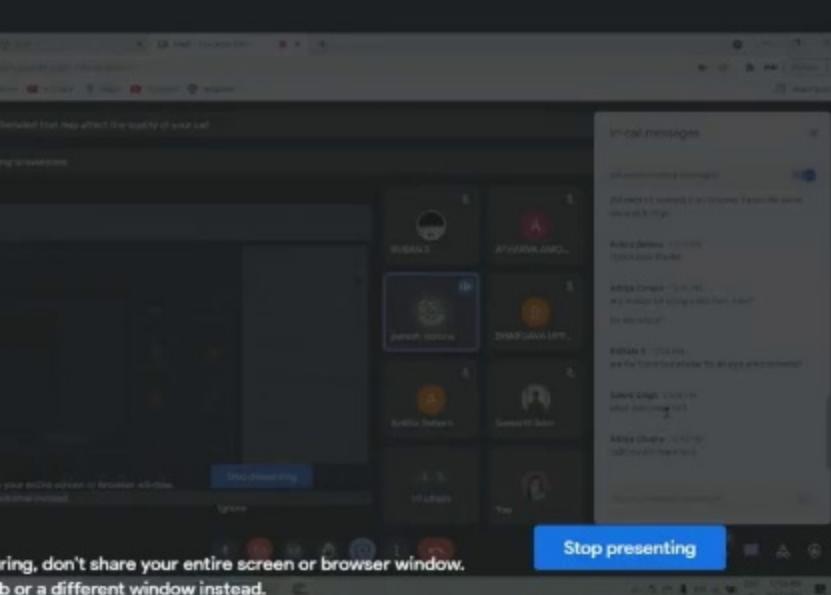
to search

Google Chrome

meet.google.com/rdx-udsw-bhn?pli=1&authuser=1

 Gmail  YouTube  Maps  YouTube  YouTube

You're presenting to everyone.



## In-call messages

Let everyone send messages

@Ankita try opening it on Chrome.  
issue with Edge.

Ankita Behera 12:34 PM  
Works now, thanks

Aditya Chopra 12:53 PM  
any reason for using a dict over a list  
for the policy?

RUBAN S 12:54 PM  
are the functions similar for all gym

Saloni Singh 12:54 PM  
what does reset do?

Aditya Chopra 12:54 PM  
right ma'am thank you!

Aditya Chopra 12:54 PM  
right ma'am thank you!

Aditya Chopra 12:54 PM  
right ma'am thank you!

Send a message to everyone



- Colabora: x Gym x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=RncYX57Q7HkD

ect) - nida... Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```
v.ns #total number of states
```

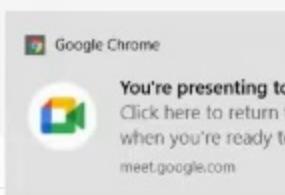
```
v.nA #total number of actions
```

```
v.P[6][1]
```

```
33333333333333, 5, 0.0, True),  
33333333333333, 10, 0.0, False),  
33333333333333, 7, 0.0, True)]
```

```
rm action and see the result
```

```
set()  
 = 0  
vation, reward, done, prob) = env.step(action)  
nder()  
observation, reward, done, prob)  
t)
```



✓ 0s completed at 12:53 PM

to search



- Colabora: x | Gym x | Meet rdx-udsw-bhn x | +

meet.google.com/rdx-udsw-bhn?pli=1&authuser=1

Gmail YouTube Maps YouTube bitspilani

You have extensions installed that may affect the quality of your call

You're presenting to everyone

RUBAN S

A

ATHARVA AMO...

B

pares saxena

BHARGAVA UPP...

A

Ankita Behera

Samarth Soni

J S

17 others

You

In-call messages

Let everyone send messages

in this case yes

Aditya Chopra 12:56 PM  
that have to be overridden always

RUBAN S 12:56 PM  
with a certain probability

VENKAT KEDARNATH K 12:56 PM  
here we have decided the action will  
particular state right

ok sir

Saloni Singh 12:57 PM  
What does reset do?

RUBAN S 12:57 PM  
resets the environment

Send a message to everyone

Stop presenting

Ignore

Microphone icon

Red square icon

Document icon

Hand icon

Blue square icon

Red phone icon

Info icon

User icon

25

- Colabora: x Gym x Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpJqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=2SXOMzTStC3n

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ★

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

4	5	6	7
8	9	10	11
12	13	14	15

```

s = [0:1, 1:2, 2:3, 3:0, 4:1, 5:1, 6:2, 9:1, 10:1, 13:2, 14:2]    # following deterministic policy
n_games = 10000
t = []      #empty list to keep track of win percentages
s = []      #empty list to keep track of scores

for i in range(n_games):    #loop over the number of games
    done = False            #set the flag of every episode reset the done flag
    s = env.reset()          #reset the environment
    score = 0                #set score for the episode to zero

    while not done:          #play the episode.
        action = policy[s]    #choose the action based on the state
        obs, reward, done, info = env.step(action)    #get the new information
        score += reward        #keep track of the score for the episode
    t.append(score/n_games)  #append the win percentage to the list
    s.append(score)           #append the score to the list

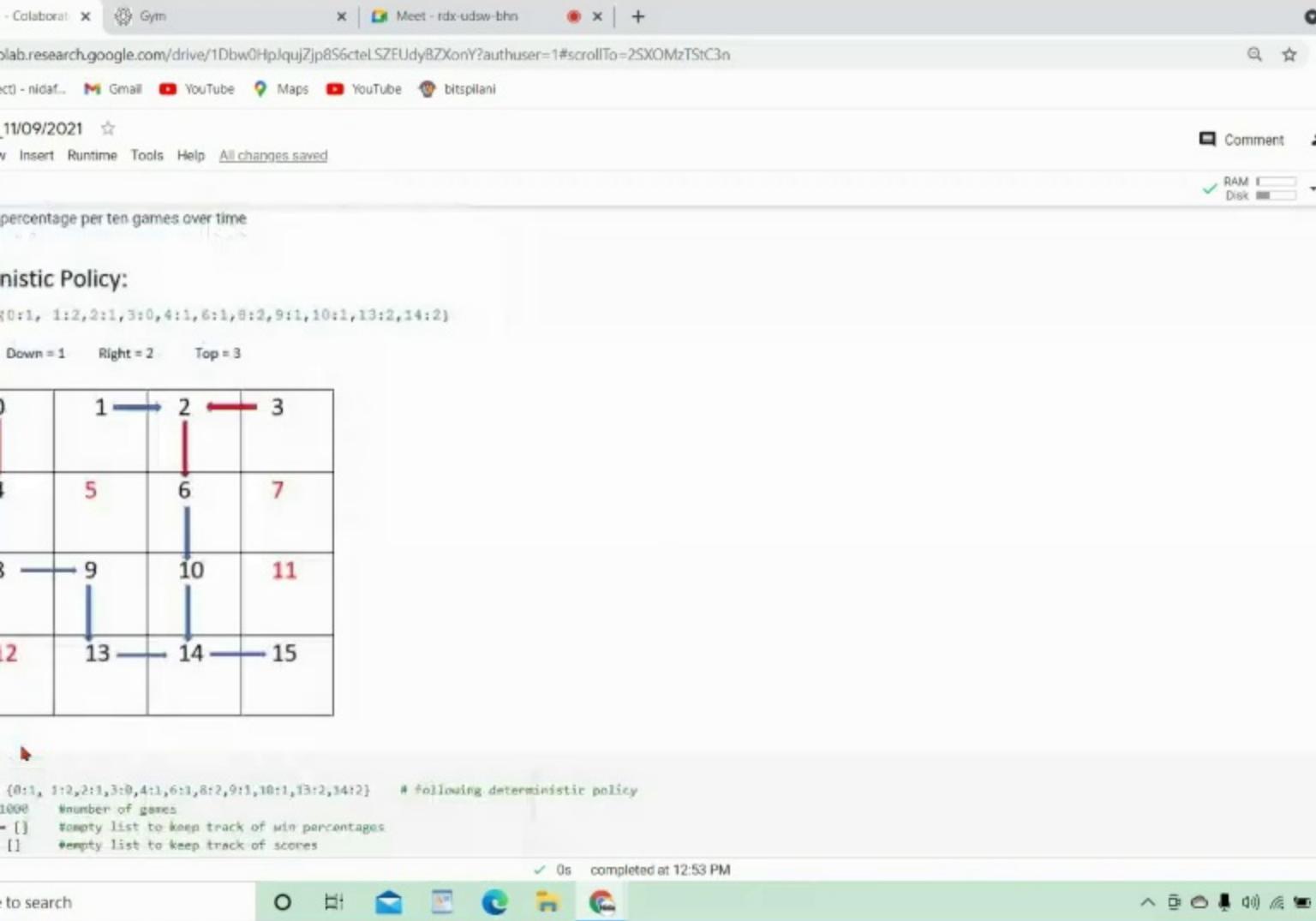
```

✓ Os completed at 12:53 PM

Google Chrome

You're presenting to [Click here to return to your presentation](#)  
meet.google.com

to search



- Colabora: x | Gym x | Meet - rdx-udsw-bhn x | +

meet.google.com/rdx-udsw-bhn?pli=1&authuser=1

(ct) - nida... Gmail YouTube Maps YouTube bitspilani

have extensions installed that may affect the quality of your call

You're presenting to everyone

RUBAN S ATHARVA AMO... A BHARGAVA UPP... B J Samarth Soni V A 9 others You

In-call messages

Let everyone send messages

ok sir

Saloni Singh 12:57 PM What does reset do?

RUBAN S 12:57 PM resets the environment

Saloni Singh 12:58 PM yes ma'am. Thank you

BHARGAVA UPPULURI 1:01 PM what is the difference between "ob-  
"observation" ?

Aditya Chopra 1:03 PM a greedy policy but not a greedy alg  
up with a policy

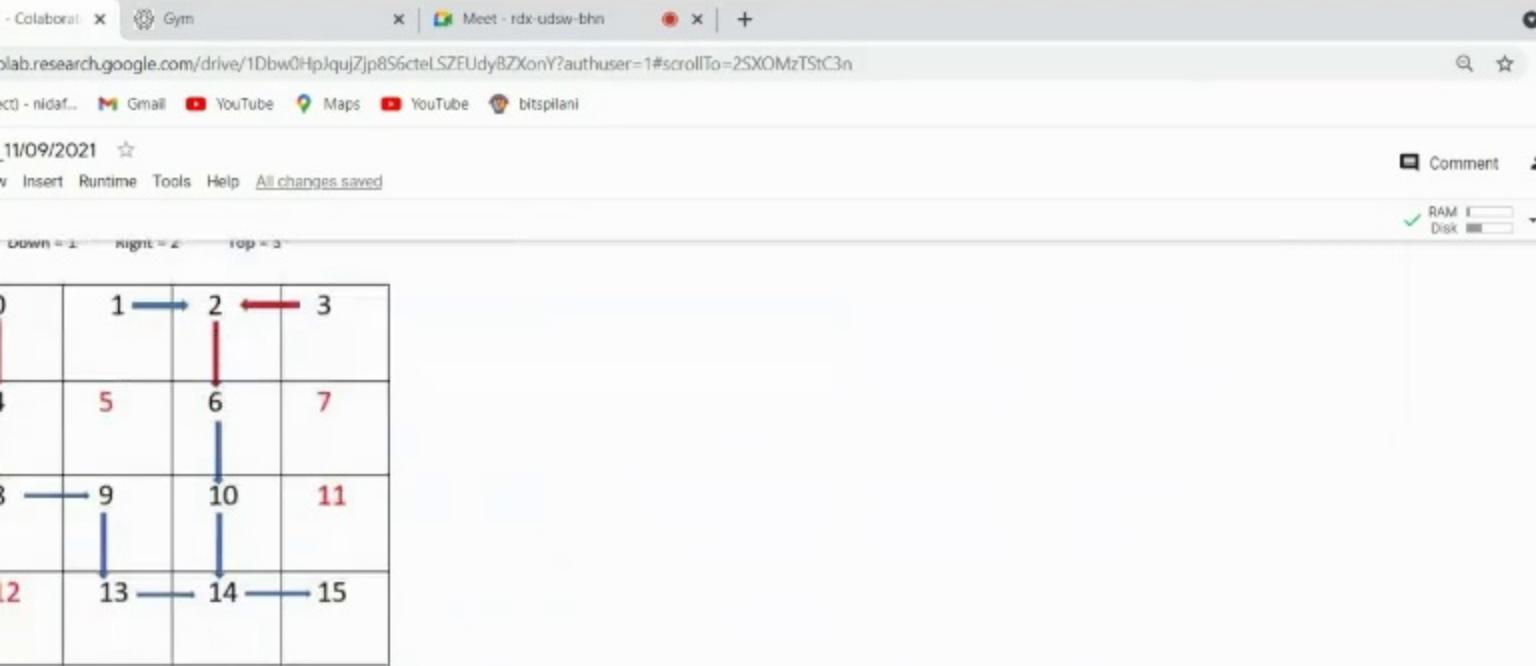
Send a message to everyone

Stop presenting

Ignore

Ignore

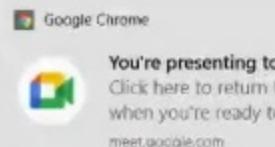
17

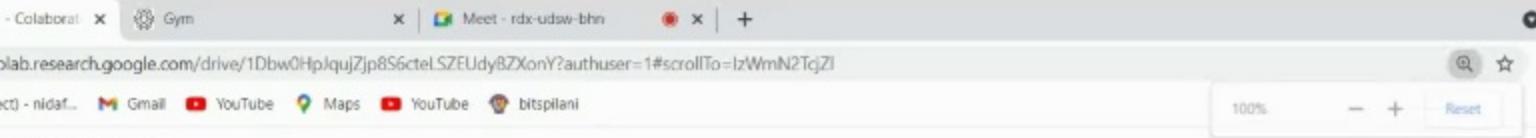


```
{0:1, 1:2, 2:1, 3:0, 4:1, 5:1, 6:1, 8:2, 9:1, 10:1, 11:2, 14:2} # following deterministic policy
1000 #number of games
[] #empty list to keep track of win percentages
[] #empty list to keep track of scores

range(n_games): #loop over the number of games
    done = False #at the top of every episode reset the done flag
    env.reset() #reset the environment
    score = 0 #set score for the episode to zero

    while not done: #play the episode
        action = policy[obs]
```





2\_11/09/2021 ★  
View Insert Runtime Tools Help All changes saved

ext

✓ RAM Disk

```
v.nA #total number of actions
```

```
v.P[6][1]
```

```
333333333333333, 5, 0.0, True),  
333333333333333, 10, 0.0, False),  
333333333333333, 7, 0.0, True)]
```

```
rm action and see the result
```

```
set()
```

```
= 0  
vation, reward, done, prob) = env.step(action)  
nder()  
observation, reward, done, prob)  
t)
```

```
False {'prob': 0.333333333333333}
```

✓ 0s completed at 12:53 PM

to search



jkterry1 redo black (#2272) ✓

Latest commit 78d2b51 on Jul 30 ⏺ History

A 15 contributors  +3

169 lines (143 sloc) 5.42 KB

[Raw](#) [Blame](#)   

```
1 import sys
2 from contextlib import closing
3
4 import numpy as np
5 from io import StringIO
6
7 from gym import utils
8 from gym.envs.toy_text import discrete
9
10 LEFT = 0
11 DOWN = 1
12 RIGHT = 2
13 UP = 3
14
15 MAPS = {
16     "4x4": ["FFFF", "FHFH", "FFFH", "HFFG"],
17     "8x8": [
18         "FFFFFFF",
19         "FFFFFFF",
20         "FFFFHFFF",
21         "FFFFFHFF",
22         "FFFHFFFF",
23         "FHFFFFHF",
24         "FHFFFHFH",
25         "FFFHFFFG",
26     ],
27 }
28
```

- Colabora: x Gym x | gym/frozen\_jake.py at master · o x | Meet - rdx-udsw-bhn x +

lab.research.google.com/drive/1Dbw0HpjqujZjp8S6cteLSZEUdyBZXonY?authuser=1#scrollTo=lzWmN2TqZI

Gmail YouTube Maps YouTube bitspilani

2\_11/09/2021 ☆

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

nment:

F	F	F
H	F	H
F	F	H
F	F	G



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Space:

Left = 0

Down = 1

Right = 2

Top = 3

✓ 0s completed at 12:53 PM

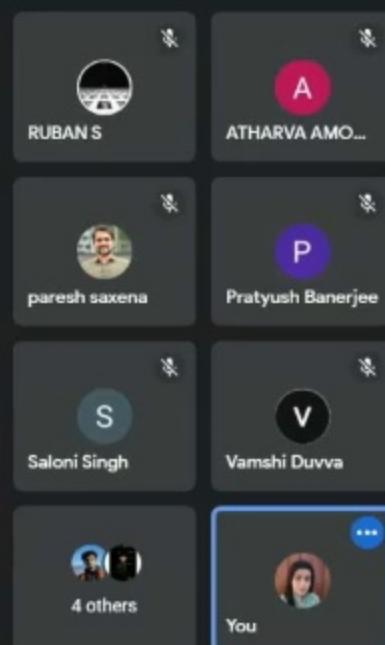
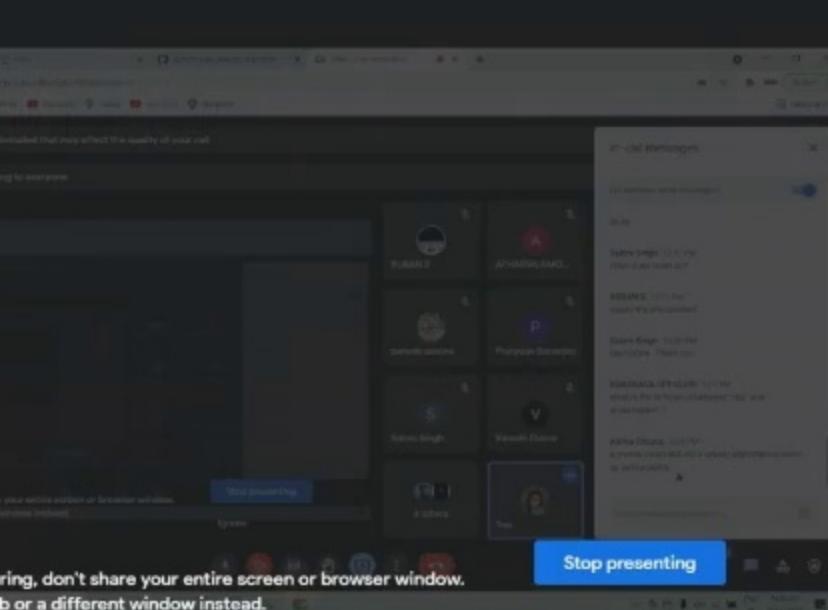
to search



meet.google.com/rdx-udsw-bhn?pli=1&authuser=1

 Gmail  YouTube  Maps  YouTube  bitcointalk

You're presenting to everyone



## In-call messages

Let everyone send messages

ok sir

Saloni Singh 12:57 PM  
What does reset do?

RUBAN S 12:57 PM  
resets the environment

Saloni Singh 12:58 PM  
yes ma'am. Thank you

BHARGAVA UPPULURI 1:01 PM  
what is the difference between "ob-  
"observation" ?

Aditya Chopra 1:03 PM  
a greedy policy but not a greedy alg  
up with a policy

Send a message to everyone





# Dynamic Programming

---

- To compute optimal policies
- Given a perfect model of the environment (Big Assumption !!)
- Computationally Expensive
- Policy Evaluation

Dr. PARESH SAXENA  
(BITS Pilani)  
psaxena@hyderabad.bits-pilani.ac.in

# Iterative Policy Evaluation

$$\begin{aligned}
 v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
 &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], 
 \end{aligned}$$

Start with some random policy  $v_0, v_1$ , and so on.

Input  $\pi$ , the policy to be evaluated

Initialize an array  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

For each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until  $\Delta < \theta$  (a small positive number)

Output  $V \approx v_\pi$

→ Don't update the values of terminal states.

Pseudocode

# Iterative Policy Evaluation: Example



	1	2	3
4	5	6	7
8		10	11
12	13	14	

$R = -1$   
on all transitions

Transition Probabilities:

$$\begin{aligned} p(7, -1|6, \text{right}) &= p(5, -1|6, \text{left}) \\ &= p(10, -1|6, \text{down}) = p(2, -1|6, \text{up}) = 1 \\ p(9, -1|6, \text{left}) &= p(10, -1|6, \text{left}) \\ &= p(13, -1|6, \text{left}) = p(14, -1|6, \text{left}) = 0 \end{aligned}$$

Lets solve this with the help of two arrays of size 14.

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')] \end{aligned}$$

Bellman Equation

# Iterative Policy Evaluation

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], \end{aligned}$$

Start with some random policy  $v_0, v_1$ , and so on.

Input  $\pi$ , the policy to be evaluated  
Initialize an array  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

For each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

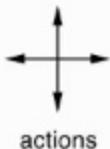
until  $\Delta < \theta$  (a small positive number)

Output  $V \approx v_\pi$

→ Don't update the values of terminal states.

Pseudocode

# Iterative Policy Evaluation: Example



	1	2	3
4	5	6	7
8		10	11
12	13	14	

$R = -1$   
on all transitions

Transition Probabilities:

$$\begin{aligned} p(7, -1|6, \text{right}) &= p(5, -1|6, \text{left}) \\ &= p(10, -1|6, \text{down}) = p(2, -1|6, \text{up}) = 1 \\ p(9, -1|6, \text{left}) &= p(10, -1|6, \text{left}) \\ &= p(13, -1|6, \text{left}) = p(14, -1|6, \text{left}) = 0 \end{aligned}$$

Lets solve this with the help of two arrays of size 14.



$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')] \end{aligned}$$

Bellman Equation

# Iterative Policy Evaluation

$$\begin{aligned}
 v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
 &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], 
 \end{aligned}$$

Start with some random policy  $v_0, v_1$ , and so on.

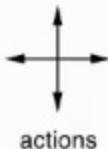
Input  $\pi$ , the policy to be evaluated  
 Initialize an array  $V(s) = 0$ , for all  $s \in \mathcal{S}$   
 Repeat

$\Delta \leftarrow 0$   
 For each  $s \in \mathcal{S}$ :  
 $v \leftarrow V(s)$   
 $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$   
 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$   
 until  $\Delta < \theta$  (a small positive number)  
 Output  $V \approx v_\pi$

Don't update the values of terminal states.

Pseudocode

# Iterative Policy Evaluation: Example



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$   
on all transitions

Transition Probabilities:

$$\begin{aligned} p(7, -1|6, \text{right}) &= p(5, -1|6, \text{left}) \\ &= p(10, -1|6, \text{down}) = p(2, -1|6, \text{up}) = 1 \\ p(9, -1|6, \text{left}) &= p(10, -1|6, \text{left}) \\ &= p(13, -1|6, \text{left}) = p(14, -1|6, \text{left}) = 0 \end{aligned}$$

Lets solve this with the help of two arrays of size 14.

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')] \end{aligned}$$

Bellman Equation

# Iterative Policy Evaluation

$$\begin{aligned}
 v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
 &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], 
 \end{aligned}$$

Start with some random policy  $v_0, v_1$ , and so on.

Input  $\pi$ , the policy to be evaluated

Initialize an array  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

For each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$  ↳

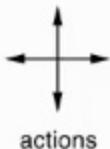
until  $\Delta < \theta$  (a small positive number)

Output  $V \approx v_\pi$

Don't update the values of terminal states.

Pseudocode

# Iterative Policy Evaluation: Example



	1	2	3
4	5	6	7
8		10	11
12	13	14	

$R = -1$   
on all transitions

Transition Probabilities:



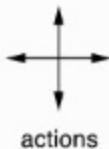
$$\begin{aligned} p(7, -1 | 6, \text{right}) &= p(5, -1 | 6, \text{left}) \\ &= p(10, -1 | 6, \text{down}) = p(2, -1 | 6, \text{up}) = 1 \\ p(9, -1 | 6, \text{left}) &= p(10, -1 | 6, \text{left}) \\ &= p(13, -1 | 6, \text{left}) = p(14, -1 | 6, \text{left}) = 0 \end{aligned}$$

Lets solve this with the help of two arrays of size 14.

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')] \end{aligned}$$

Bellman Equation

# Policy evaluation and improvement



$R = -1$   
on all transitions

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma v_k(s')], \end{aligned}$$

States (s)	$v_0$	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	
13	0	
14	0	

# Iterative Policy Evaluation

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], \end{aligned}$$

Start with some random policy  $v_0, v_1$ , and so on.

Input  $\pi$ , the policy to be evaluated  
 Initialize an array  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$   
 Repeat

$\Delta \leftarrow 0$   
 For each  $s \in \mathcal{S}$ :  
 $v \leftarrow V(s)$   
 $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$   
 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

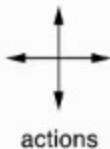
until  $\Delta < \theta$  (a small positive number)

Output  $V \approx v_\pi$

→ Don't update the values of terminal states.

Pseudocode

# Policy evaluation and improvement



$R = -1$   
on all transitions

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma v_k(s')], \end{aligned}$$

States (s)	$v_0$	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	
13	0	
14	0	

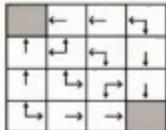
# Policy evaluation and improvement



States (s)	v <sub>1</sub>	v <sub>2</sub>	.	.	.	v <sub>∞</sub>
1	-1	-1.7				-14
2	-1	-2				-20
3	-1	-2				-22
4	-1	-1.7				-14
5	-1	-2				-18
6	-1	-2				-20
7	-1	-2				-20
8	-1	-2				-20
9	-1	-2				-20
10	-1	-2				-18
11	-1	-1.7				-14
12	-1	-2				-22
13	-1	-2				-20
14	-1	-1.7				-14

Optimal values and policy

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



- In general, improvements are guaranteed.
- This is the special case where improved policy is also the optimal one.

# Policy Iteration

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

E – policy evaluation

I – policy improvement

1. Initialization

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

2. Policy Evaluation

Repeat

$$\Delta \leftarrow 0$$

For each  $s \in \mathcal{S}$ :

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until  $\Delta < \theta$  (a small positive number)

3. Policy Improvement

*policy-stable*  $\leftarrow$  true

For each  $s \in \mathcal{S}$ :

$$\underline{\pi} \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

If  $a \neq \pi(s)$ , then *policy-stable*  $\leftarrow$  false

If *policy-stable*, then stop and return  $V$  and  $\pi$ ; else go to 2

- Policy Evaluation: Estimate  $v_\pi$  by iterative policy evaluation
- Policy Improvement: Generate  $\pi' \geq \pi$  by greedy policy improvement.

# Exercise



0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

- Let's add one more state 15
- Actions at state 15:
  - Left, next state: 7
  - Right, next state: 15
  - Down, next state: 11
  - UP, next state: 3
- Values and transition of previous states is unchanged
- Calculate  $v_{\pi}(15)$  for equiprobable random policy ?



# Solution

$$\begin{aligned}
 v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
 &= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma v_k(s')], 
 \end{aligned}$$

	1	2	3	
4	5	6	7	15
8	9	10	11	
12	13	14		

$$\begin{aligned}
 v_{k+1}(15) &= (1/4)p(15, -1 | 15, right)[-1 + v_k(15)] \\
 &\quad + (1/4)p(3, -1 | 15, top)[-1 + -22] \\
 &\quad + (1/4)p(11, -1 | 15, down)[-1 + -14] \\
 &\quad + (1/4)p(7, -1 | 15, left)[-1 + -20] \\
 \\ 
 &= (1/4)[-4 - 22 - 14 - 20 + v_k(15)]
 \end{aligned}$$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Here we may assume  $v_k(15) = v_{k+1}(15)$ . In some places  $v_k(15) = 0$  is also assumed.

$$v_{k+1}(15) = -20$$

We are assuming  $\gamma = 1$

# Iterative Policy Evaluation: Example



	1	2	3
4	5	6	7
8		10	11
12	13	14	

$R = -1$   
on all transitions

Transition Probabilities:

$$\begin{aligned} p(7, -1 | 6, \text{right}) &= p(5, -1 | 6, \text{left}) \\ &= p(10, -1 | 6, \text{down}) = p(2, -1 | 6, \text{up}) = 1 \\ p(9, -1 | 6, \text{left}) &= p(10, -1 | 6, \text{left}) \\ &= p(13, -1 | 6, \text{left}) = p(14, -1 | 6, \text{left}) = 0 \end{aligned}$$

Lets solve this with the help of two arrays of size 14.

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{\substack{s', r \\ \text{by}}} p(s', r | s, a) [r + \gamma v_k(s')] \end{aligned}$$

Bellman Equation

# Iterative Policy Evaluation

$$\begin{aligned}
 v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
 &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], 
 \end{aligned}$$

Start with some random policy  $v_0, v_1$ , and so on.

Input  $\pi$ , the policy to be evaluated  
 Initialize an array  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$   
 Repeat

```

 $\Delta \leftarrow 0$ 
For each  $s \in \mathcal{S}$ :
   $v \leftarrow V(s)$ 
   $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ 
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number)
Output  $V \approx v_\pi$ 
  
```

→ Don't update the values of terminal states.

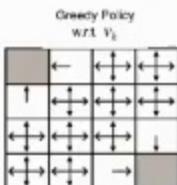
Pseudocode

# Policy evaluation and improvement



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



Since  
 $v_0(\text{terminal state}) = 0$ ,  
Greedy policy at state 1 is  
to go left, at state 4 is to  
go up, at state 11 is to go  
down and at state 14 is to  
go right.

$$\begin{aligned}v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')],\end{aligned}$$

$$\begin{aligned}v_2(1) &= (1/4)p(2, -1|1, right)[-1+1] \\&\quad + (1/4)p(1, -1|1, top)[-1+1] \\&\quad + (1/4)p(1, -1|1, down)[-1+1] \\&\quad + (1/4)p(T, -1|1, left)[-1+0] \\&= -1.75\end{aligned}$$

States (s)	$v_1$	$v_2$
1	-1	
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	
8	-1	
9	-1	
10	-1	
11	-1	
12	-1	
13	-1	
14	-1	

# Solution

$$\begin{aligned}
 v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
 &= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) \left[ r + \gamma v_k(s') \right],
 \end{aligned}$$

	1	2	3	
4	5	6	7	15
8	9	10	11	
12	13	14		

$$\begin{aligned}
 v_{k+1}(15) &= (1/4)p(15, -1 | 15, right)[-1 + v_k(15)] \\
 &\quad + (1/4)p(3, -1 | 15, top)[-1 + -22] \\
 &\quad + (1/4)p(11, -1 | 15, down)[-1 + -14] \\
 &\quad + (1/4)p(7, -1 | 15, left)[-1 + -20] \\
 &= (1/4)[-4 - 22 - 14 - 20 + v_k(15)]
 \end{aligned}$$

Here we may assume  $v_k(15) = v_{k+1}(15)$ . In some places  $v_k(15) = 0$  is also assumed.

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$$v_{k+1}(15) = -20$$

We are assuming  $\gamma = 1$

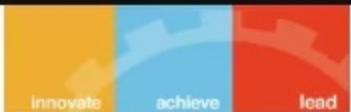


# Reminder: Project Step 1

## Deadline: 25<sup>th</sup> Sep

- 
- Please follow the CMS post.

Dr. PARESH SAXENA  
(BITS Pilani)  
psaxena@hyderabad.bits-pilani.ac.in



# Reminder: Project Step 1

## Deadline: 25<sup>th</sup> Sep

- 
- Please follow the CMS post.

Dr. PARESH SAXENA  
(BITS Pilani)  
psaxena@hyderabad.bits-pilani.ac.in

# Policy Improvement Theorem

For all states:  $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ .

A new policy  $\pi'$  must be good as, or better than policy  $\pi$ . Meaning for all states:

$$v_{\pi'}(s) \geq v_\pi(s).$$

Select a new greedy policy  $\pi'$ , such as:

$$\begin{aligned}\pi'(s) &\doteq \arg \max_a q_\pi(s, a) \\ &= \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \arg \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_\pi(s')],\end{aligned}$$

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

←	←	↑	
↑	↖	↑	↓
↑	↑	↗	↓
↑	→	→	↓

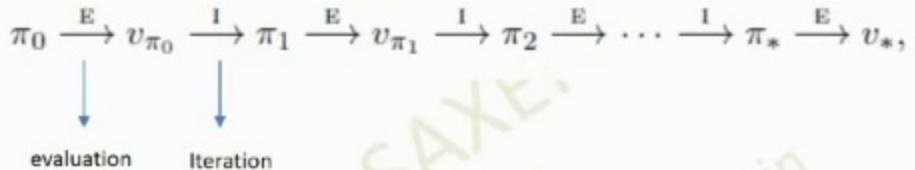


With greedy policy, the state values will be either -1, -2 or -3 for all the states.

Greedy Policy Meets the conditions of the Policy Improvement Theorem.

Previous Example

# Policy Iteration



For a finite MDP with finite number of policies, this process will converge to optimal policy !!



# Policy Improvement Theorem

For all states:  $q_{\pi}(s, \pi'(s)) \geq v_{\pi}(s)$ .

A new policy  $\pi'$  must be good as, or better than policy  $\pi$ . Meaning for all states:

$$v_{\pi'}(s) \geq v_{\pi}(s).$$

Select a new greedy policy  $\pi'$ , such as:

$$\begin{aligned}\pi'(s) &\doteq \arg \max_a q_{\pi}(s, a) \\ &= \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \arg \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_{\pi}(s')],\end{aligned}$$

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

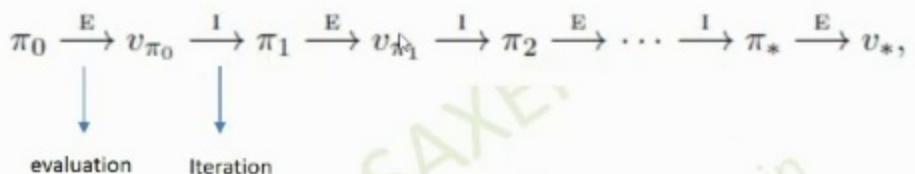
←	←	↖	
↑	↖	↖	↓
↑	↑	↖	↓
↑	→	→	↓

With greedy policy, the state values will be either -1, -2 or -3 for all the states.

Greedy Policy Meets the conditions of the Policy Improvement Theorem.

Previous Example

# Policy Iteration



For a finite MDP with finite number of policies, this process will converge to optimal policy !!

# Policy Iteration: PseudoCode

```

1. Initialization
 $V(s) \in \mathbb{R}$  and  $\pi(s) \in \mathcal{A}(s)$  arbitrarily for all  $s \in \mathcal{S}$ 

2. Policy Evaluation
    Repeat
         $\Delta \leftarrow 0$ 
        For each  $s \in \mathcal{S}$ :
             $v \leftarrow V(s)$ 
             $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$ 
             $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
        until  $\Delta < \theta$  (a small positive number)

3. Policy Improvement
     $policy-stable \leftarrow true$ 
    For each  $s \in \mathcal{S}$ :
         $a \leftarrow \pi(s)$ 
         $\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ 
        If  $a \neq \pi(s)$ , then  $policy-stable \leftarrow false$ 
    If  $policy-stable$ , then stop and return  $V$  and  $\pi$ ; else go to 2
  
```

## Policy Iteration

Drawback: Each policy iteration involves policy evaluation !! – computationally expensive.  
 Possible solution – stop the policy evaluation after one sweep through all the states (also known as value iteration algorithm)

# Value Iteration: Pseudocode

## 1. Initialization

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

## 2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

    For each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

    until  $\Delta < \theta$  (a small positive number)

## 3. Policy Improvement

*policy-stable*  $\leftarrow$  true

For each  $s \in \mathcal{S}$ :

$a \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

    If  $a \neq \pi(s)$ , then *policy-stable*  $\leftarrow$  false

If *policy-stable*, then stop and return  $V$  and  $\pi$ ; else go to 2

## Policy Iteration

Combine policy improvement and truncated policy evaluation.

### Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold  $\theta > 0$  determining accuracy of estimation  
Initialize  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop:

$\Delta \leftarrow 0$

    Loop for each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

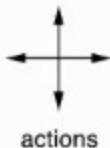
    until  $\Delta < \theta$

Output a deterministic policy,  $\pi \approx \pi_*$ , such that  

$$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

## Value Iteration

# Value Iteration: Example



$R = -1$   
on all transitions

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t=s, A_t=a] \\ &= \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], \end{aligned}$$

States (s)	$v_0$
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0

# Value Iteration: Pseudocode

## 1. Initialization

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

## 2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

    For each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

    until  $\Delta < \theta$  (a small positive number)

## 3. Policy Improvement

*policy-stable*  $\leftarrow$  true

For each  $s \in \mathcal{S}$ :

$a \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

    If  $a \neq \pi(s)$ , then *policy-stable*  $\leftarrow$  false

If *policy-stable*, then stop and return  $V$  and  $\pi$ ; else go to 2

## Policy Iteration

Combine policy improvement and truncated policy evaluation.

### Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold  $\theta > 0$  determining accuracy of estimation  
Initialize  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop:

$\Delta \leftarrow 0$

    Loop for each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

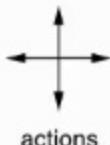
    until  $\Delta < \theta$

Output a deterministic policy,  $\pi \approx \pi_*$ , such that

$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

## Value Iteration

# Value Iteration: Example



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$   
on all transitions

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t=s, A_t=a] \\ &= \max_a \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')], \end{aligned}$$



States (s)	$v_0$	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	
13	0	
14	0	

# Value Iteration: Example



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$   
on all transitions

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t=s, A_t=a] \\ &= \max_a \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')], \end{aligned}$$

$$\begin{aligned} v_2(1) &= \max[p(2, -1|1, right)[-1+1] \\ &\quad + p(1, -1|1, top)[-1+1] \\ &\quad + p(1, -1|1, down)[-1+1] \\ &\quad + p(T, -1|1, left)[-1+0]] \end{aligned}$$

$$= -1$$

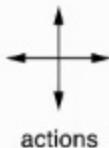
$$\begin{aligned} v_2(2) &= p(3, -1|2, right)[-1+1] \\ &\quad + p(2, -1|2, top)[-1+1] \\ &\quad + p(6, -1|2, down)[-1+1] \\ &\quad + p(1, -1|2, left)[-1+1] \end{aligned}$$

$$= -2$$

Compare with the Value Iteration Example. We are converging to the optimal policy.

States (s)	V <sub>1</sub>	V <sub>2</sub>
1	-1	-1
2	-1	-2
3	-1	-2
4	-1	-1
5	-1	-2
6	-1	-2
7	-1	-2
8	-1	-2
9	-1	-2
10	-1	-2
11	-1	-1
12	-1	-2
13	-1	-2
14	-1	-1

# Value Iteration: Example



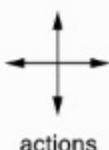
$R = -1$   
on all transitions

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t=s, A_t=a] \\ &= \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], \end{aligned}$$

$$\begin{aligned} v_1(1) &= \max[p(2,-1|1,right)[-1+0] \\ &\quad ,p(1,-1|1,top)[-1+0] \\ &\quad ,p(1,-1|1,down)[-1+0] \\ &\quad ,p(T,-1|1,left)[-1+0]] \\ &= -1 \end{aligned}$$

States (s)	$v_0$	$v_1$
1	0	-1
2	0	-1
3	0	-1
4	0	-1
5	0	-1
6	0	-1
7	0	-1
8	0	-1
9	0	-1
10	0	-1
11	0	-1
12	0	-1
13	0	-1
14	0	-1

# Value Iteration: Example



$R = -1$   
on all transitions

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t=s, A_t=a] \\ &= \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], \end{aligned}$$

$$\begin{aligned} v_2(1) &= \max[p(2, -1|1, right)[-1+1] \\ &\quad + p(1, -1|1, top)[-1+1] \\ &\quad + p(1, -1|1, down)[-1+1] \\ &\quad + p(T, -1|1, left)[-1+0]] \end{aligned}$$

$$= -1$$

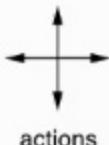
$$\begin{aligned} v_2(2) &= p(3, -1|2, right)[-1+1] \\ &\quad + p(2, -1|2, top)[-1+1] \\ &\quad + p(6, -1|2, down)[-1+1] \\ &\quad + p(1, -1|2, left)[-1+1] \end{aligned}$$

$$= -2$$

Compare with the Value Iteration Example. We are converging to the optimal policy.

States (s)	V <sub>1</sub>	V <sub>2</sub>
1	-1	-1
2	-1	-2
3	-1	-2
4	-1	-1
5	-1	-2
6	-1	-2
7	-1	-2
8	-1	-2
9	-1	-2
10	-1	-2
11	-1	-1
12	-1	-2
13	-1	-2
14	-1	-1

# Value Iteration: Example



$R = -1$   
on all transitions

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t=s, A_t=a] \\ &= \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], \end{aligned}$$

$$\begin{aligned} v_1(1) &= \max[p(2,-1|1,right)[-1+0] \\ &\quad ,p(1,-1|1,top)[-1+0] \\ &\quad ,p(1,-1|1,down)[-1+0] \\ &\quad ,p(T,-1|1,left)[-1+0]] \end{aligned}$$

$$= -1$$

States (s)	$v_0$	$v_1$
1	0	-1
2	0	-1
3	0	-1
4	0	-1
5	0	-1
6	0	-1
7	0	-1
8	0	-1
9	0	-1
10	0	-1
11	0	-1
12	0	-1
13	0	-1
14	0	-1

# Value Iteration: Example



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$   
on all transitions

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')], \end{aligned}$$

$$\begin{aligned} v_2(1) &= \max[p(2, -1|1, right)[-1+1] \\ &\quad + p(1, -1|1, top)[-1+1] \\ &\quad + p(1, -1|1, down)[-1+1] \\ &\quad + p(T, -1|1, left)[-1+0]] \\ &\stackrel{?}{=} -1 \end{aligned}$$

$$\begin{aligned} v_2(2) &= p(3, -1|2, right)[-1+1] \\ &\quad + p(2, -1|2, top)[-1+1] \\ &\quad + p(6, -1|2, down)[-1+1] \\ &\quad + p(1, -1|2, left)[-1+1] \\ &= -2 \end{aligned}$$

Compare with the Value Iteration Example. We are converging to the optimal policy.

States (s)	$V_1$	$V_2$
1	-1	-1
2	-1	-2
3	-1	-2
4	-1	-1
5	-1	-2
6	-1	-2
7	-1	-2
8	-1	-2
9	-1	-2
10	-1	-2
11	-1	-1
12	-1	-2
13	-1	-2
14	-1	-1



# Value Iteration: Example



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$   
on all transitions

$$\begin{aligned}v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\&= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')],\end{aligned}$$

$$\begin{aligned}v_2(1) &= \max[p(2, -1|1, right)[-1+1] \\&\quad , p(1, -1|1, top)[-1+1] \\&\quad , p(1, -1|1, down)[-1+1] \\&\quad , p(T, -1|1, left)[-1+0]]\end{aligned}$$

$$= -1$$

$$\begin{aligned}v_2(2) &= p(3, -1|2, right)[-1+1] \\&\quad , p(2, -1|2, top)[-1+1] \\&\quad , p(6, -1|2, down)[-1+1] \\&\quad , p(1, -1|2, left)[-1+1]\end{aligned}$$

$$= -2$$

Compare with the Value Iteration Example. We are converging to the optimal policy.

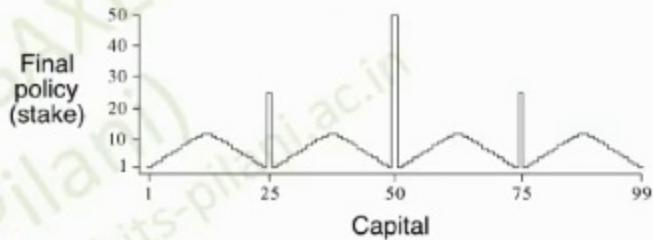
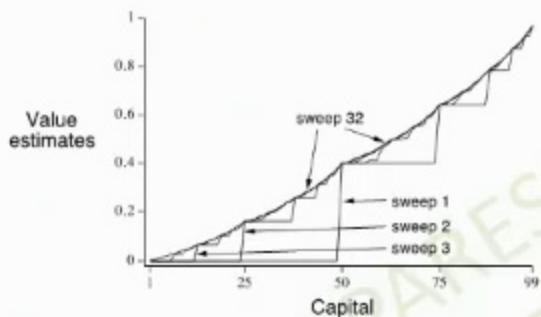
States (s)	V <sub>1</sub>	V <sub>2</sub>
1	-1	-1
2	-1	-2
3	-1	-2
4	-1	-1
5	-1	-2
6	-1	-2
7	-1	-2
8	-1	-2
9	-1	-2
10	-1	-2
11	-1	-1
12	-1	-2
13	-1	-2
14	-1	-1

# Gambler's Problem

- Gambler flips the coin.
- Assuming  $p_h$  and  $p_t$  can be different.
- If at a flip turn, his stake is  $x$ , and if it is heads, he wins  $x$  more. If it is tails, he losses  $x$ .
- Goal is to reach 100\$
- Reward: 0 on all transitions and +1 when gambler earns 100\$
- What will be the states ?
- What will be the actions ?



# Results from the Book



- Here  $p_h=0.4$ .
- Why are there spikes ?

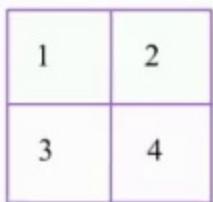


# Efficiency of DP

---

- A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of policies is  $m^n$ , m and n are no. of states and actions resp.
- DP is faster than any direct search in policy space.
- DP methods can be used with today's computers to solve MDPs with millions of states.
- Both policy and value iterations are widely used and it is not clear which is better in general.
- On problems with large state spaces, asynchronous DP methods (out of scope of this course) with GPIs are often preferred.

# DP: Problem Solving Day



- 4 states
- 4 actions: L, R, U and D
- Loopback: From 1 take up and reach 3, from 1 take left and reach 2 and so on.
- Reward: 1 if there is no loopback, 0 for all other cases.

Q. Perform policy evaluation for equiprobable random policy.

- (a) Find the state-value estimates for each state after 5 iterations. Assume the initial values are 0. Use  $\gamma = 0.8$ .
- (b) Find the state-value estimates for each state after infinite iterations. Assume the initial values are 0. Use  $\gamma = 0.8$ .

Use this:

$$\begin{aligned}v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma v_k(s')],\end{aligned}$$

# DP: Solution - 1

1	2
3	4

$$\begin{aligned}v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')],\end{aligned}$$

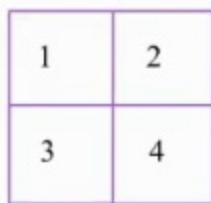
$$\begin{aligned}v_1(1) &= (1/4)p(2,1|1,right)[1+0] \\&\quad + (1/4)p(3,0|1,top)[0+0] \\&\quad + (1/4)p(3,1|1,down)[1+0] \\&\quad + (1/4)p(2,0|1,left)[0+0]\end{aligned}$$

$$\begin{aligned}v_1(1) &= (1/4)(1) [1+0] \\&\quad + (1/4)(1) [0+0] \\&\quad + (1/4)(1) [1+0] \\&\quad + (1/4)(1) [0+0]\end{aligned}$$

$$v_1(1) = 1/4 + 1/4 = 1/2$$



# DP: Problem Solving Day



- 4 states
- 4 actions: L, R, U and D
- Loopback: From 1 take up and reach 3, from 1 take left and reach 2 and so on.
- Reward: 1 if there is no loopback, 0 for all other cases.

Q. Perform policy evaluation for equiprobable random policy.

- (a) Find the state-value estimates for each state after 5 iterations. Assume the initial values are 10. Use  $\gamma = 0.8$ .
- (b) Find the state-value estimates for each state after infinite iterations. Assume the initial values are 10. Use  $\gamma = 0.8$ .

Use this:

$$\begin{aligned}v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma v_k(s')],\end{aligned}$$



# DP: Problem Solving Day



- 4 states
- 4 actions: L, R, U and D
- Loopback: From 1 take up and reach 3, from 1 take left and reach 2 and so on.
- Reward: 1 if there is no loopback, 0 for all other cases.

Q. Perform policy evaluation for equiprobable random policy.

- (a) Find the state-value estimates for each state after 5 iterations. Assume the initial values are 10. Use  $\gamma = 0.8$ .
- (b) Find the state-value estimates for each state after infinite iterations. Assume the initial values are 10. Use  $\gamma = 0.8$ .

Use this:

$$\begin{aligned}v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')],\end{aligned}$$

# Solution

Solve this similar to the previous problem, you will get the same

To derive a general expression, consider,  $u_n = au_{n-1} + b$   
 $\Rightarrow u_n = a(au_{n-2} + b) + b = a(a(au_{n-3} + b) + b) + b$   
 $\Rightarrow u_n = a^3u_{n-3} + b + ab + a^2b$

We can extrapolate that:

$$u_n = a^n u_0 + \frac{b \cdot (1 - a^n)}{1 - a}$$

and

$$u_\infty = \frac{b}{1 - a}$$

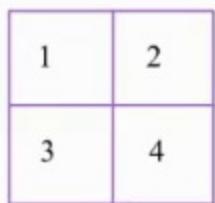
Here,  $a=0.8$ ,  $b=0.5$ ,  $u_0=10$

After 5 iterations:  $0.8^5 \cdot 10 + [0.5 (1-0.8)^5] / (1-0.8) = 4.9576$

After infinite iterations:  $0.5 / 1 - 0.8 = 2.5$



# DP: Problem Solving Day



- 4 states
- 4 actions: L, R, U and D
- Loopback: From 1 take up and reach 3, from 1 take left and reach 2 and so on.
- Reward: 1 if there is no loopback, 0 for all other cases.

Q. Perform policy evaluation for equiprobable random policy.

- (a) Find the state-value estimates for each state after infinite iterations. Use  $\gamma = 0.8$ . Assume the initial values are:  $v_0(1) = 8, v_0(2) = 5, v_0(3) = -2, v_0(4) = -5$

Use this:

$$\begin{aligned}v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')],\end{aligned}$$

# Solution

Solve this similar to the previous problem, you will get the same

To derive a general expression, consider,  $u_n = au_{n-1} + b$   
 $\implies u_n = a(au_{n-2} + b) + b = a(a(au_{n-3} + b) + b) + b$   
 $\implies u_n = a^3u_{n-3} + b + ab + a^2b$

We can extrapolate that:

$$u_n = a^n u_0 + \frac{b \cdot (1 - a^n)}{1 - a}$$

and

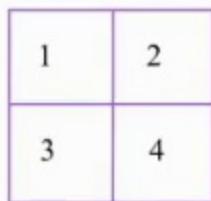
$$u_\infty = \frac{b}{1 - a}$$

Here,  $a=0.8$ ,  $b=0.5$ ,  $u_0=10$

After 5 iterations:  $0.8^5 \cdot 10 + [0.5 (1-0.8)^5] / (1-0.8) = 4.9576$

After infinite iterations:  $0.5 / 1 - 0.8 = 2.5$

# DP: Problem Solving Day



- 4 states
- 4 actions: L, R, U and D
- Loopback: From 1 take up and reach 3, from 1 take left and reach 2 and so on.
- Reward: 1 if there is no loopback, 0 for all other cases.

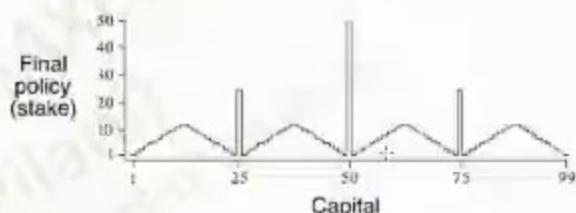
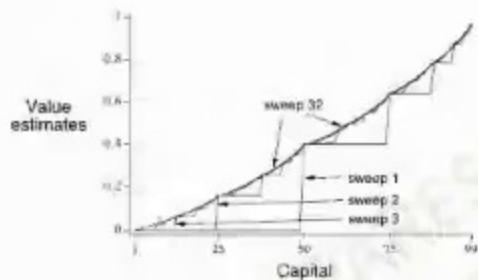
Q. Perform policy evaluation for equiprobable random policy.

- (a) Find the state-value estimates for each state after infinite iterations. Use  $\gamma = 0.8$ . Assume the initial values are:  $v_0(1) = 8, v_0(2) = 5, v_0(3) = -2, v_0(4) = -5$

Use this:

$$\begin{aligned}v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')],\end{aligned}$$

# Results from the Book



- Here  $p_h=0.4$ .
- Why are there spikes ?

# Policy Iteration

 $\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_k \xrightarrow{E} v_k$ 

## 1. Initialization

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

## 2. Policy Evaluation

Repeat

 $\Delta \leftarrow 0$ For each  $s \in \mathcal{S}$ : $v \leftarrow V(s)$  $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$  $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ until  $\Delta < \theta$  (a small positive number)

## 3. Policy Improvement

 $policy-stable \leftarrow true$ For each  $s \in \mathcal{S}$ : $a \leftarrow \pi(s)$  $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ If  $a \neq \pi(s)$ , then  $policy-stable \leftarrow false$ If  $policy-stable$ , then stop and return  $V$  and  $\pi$ ; else go to 2

E – policy evaluation

I – policy improvement

- Policy Evaluation: Estimate  $v_\pi$  by iterative policy evaluation
- Policy Improvement: Generate  $\pi' \geq \pi$  by greedy policy improvement.

# DP: Solution - 1

1	2
3	4

$$\begin{aligned}v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_k(s')],\end{aligned}$$

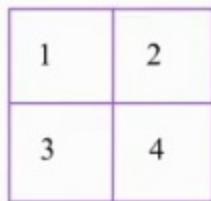
$$\begin{aligned}v_1(1) &= (1/4)p(2,1|1,right)[1+0] \\&\quad + (1/4)p(3,0|1,top)[0+0] \\&\quad + (1/4)p(3,1|1,down)[1+0] \\&\quad + (1/4)p(2,0|1, left)[0+0]\end{aligned}$$

$$\begin{aligned}v_1(1) &= (1/4)(1) [1+0] \\&\quad + (1/4)(1) [0+0] \\&\quad + (1/4)(1) [1+0] \\&\quad + (1/4)(1) [0+0]\end{aligned}$$

$$v_1(1) = 1/4 + 1/4 = 1/2$$



# DP: Problem Solving Day



- 4 states
- 4 actions: L, R, U and D
- Loopback: From 1 take up and reach 3, from 1 take left and reach 2 and so on.
- Reward: 1 if there is no loopback, 0 for all other cases.

Q. Perform policy evaluation for equiprobable random policy.

- (a) Find the state-value estimates for each state after infinite iterations. Use  $\gamma = 0.8$ . Assume the initial values are:  $v_0(1) = 8, v_0(2) = 5, v_0(3) = -2, v_0(4) = -5$

Use this:

$$\begin{aligned} I \quad v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')], \end{aligned}$$

# Reminder: Project

## STEP 1 (SUBMISSION OF GROUP DETAILS)

- 4-5 students in one group
- Send the group details to TA - Nida Fatima ([p20190504@hyderabad.bits.pilani.ac.in](mailto:p20190504@hyderabad.bits.pilani.ac.in)) by Saturday, 25th Sep 2021, 17:00. Only one group member shall send the email, keeping in cc all the other group members, and include the names and IDs of all the group members.
- Please keep the subject of the mail as: "Reinforcement Learning: Project Group Details"

# Reminder: Project

## STEP 1 (SUBMISSION OF GROUP DETAILS)

- 4-5 students in one group
- Send the group details to TA - Nida Fatima ([p20190504@hyderabad.bits.pilani.ac.in](mailto:p20190504@hyderabad.bits.pilani.ac.in)) by Saturday, 25th Sep 2021, 17:00. Only one group member shall send the email, keeping in cc all the other group members, and include the names and IDs of all the group members.
- Please keep the subject of the mail as: "Reinforcement Learning: Project Group Details"

# Example: Blackjack

Ace worth 1 or 11



All face cards count is 10.



Player

Dealer

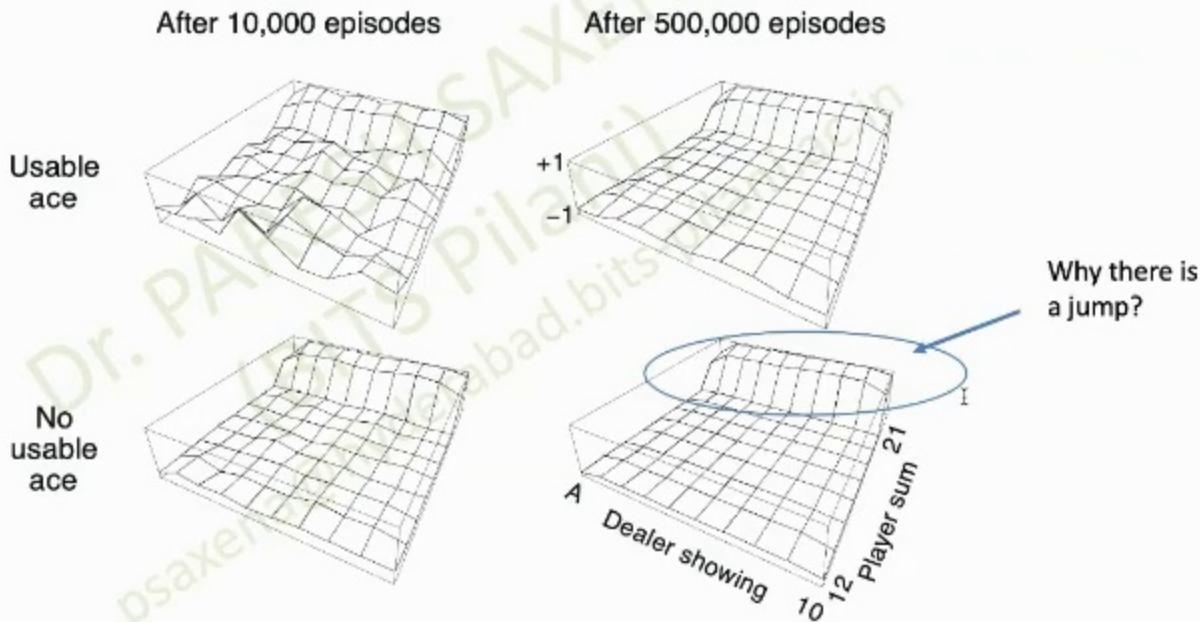
First player samples card from deck (hit) until stop (stick)

Then, dealer samples card from deck (hit) until sum > 16 (stick)

- Player loses (-1 reward) if busts (card sum > 21)
- Player wins (+1 reward) if dealer bust or player sum > dealer sum
- Player draw (0 reward) if player sum = dealer sum
- Rewards are 0 within the game

# Example: Evaluation of Policy

Policy: Player sticks if sum is 20 or 21 or else hit



# Example: Blackjack – as MDP

Ace worth 1 or 11



All face cards count is 10.



Player

Dealer

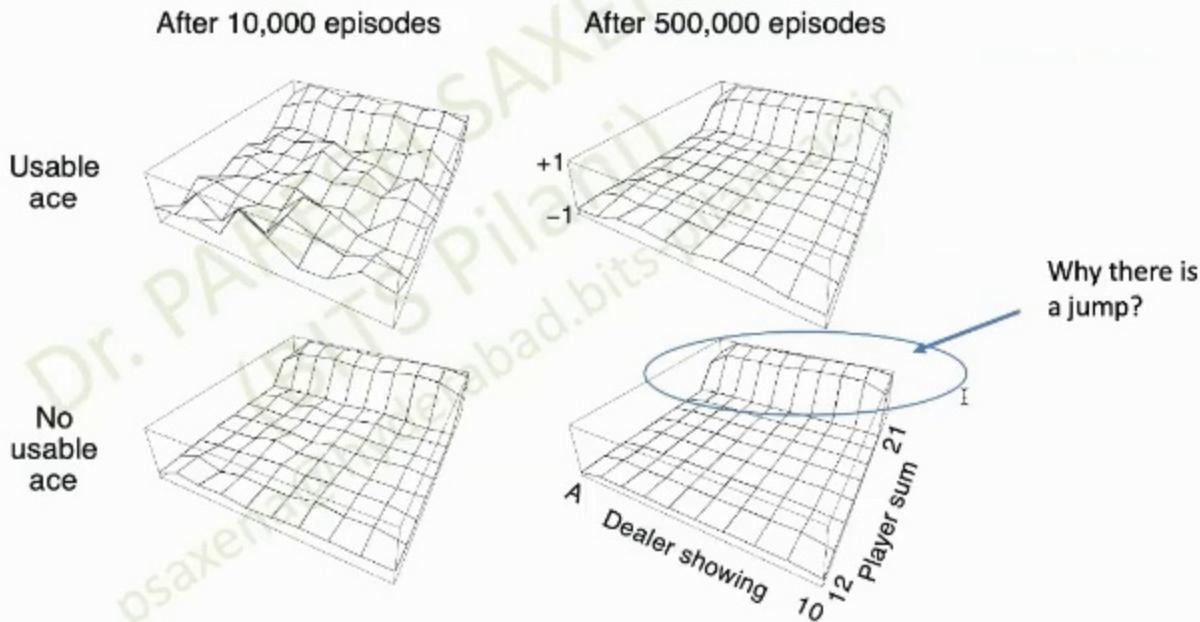
First player samples card from deck (hit) until stop (stick)

Then, dealer samples card from deck (hit) until sum > 16 (stick)

- Total states = 200 3 tuple: (player sum, dealer's card, holding of ace)
- Action = {hit, stick}
- Rewards = {-1, 0, 1}

# Example: Evaluation of Policy

Policy: Player sticks if sum is 20 or 21 or else hit



# Example: Blackjack – as MDP

Ace worth 1 or 11



All face cards count is 10.



Player

Dealer

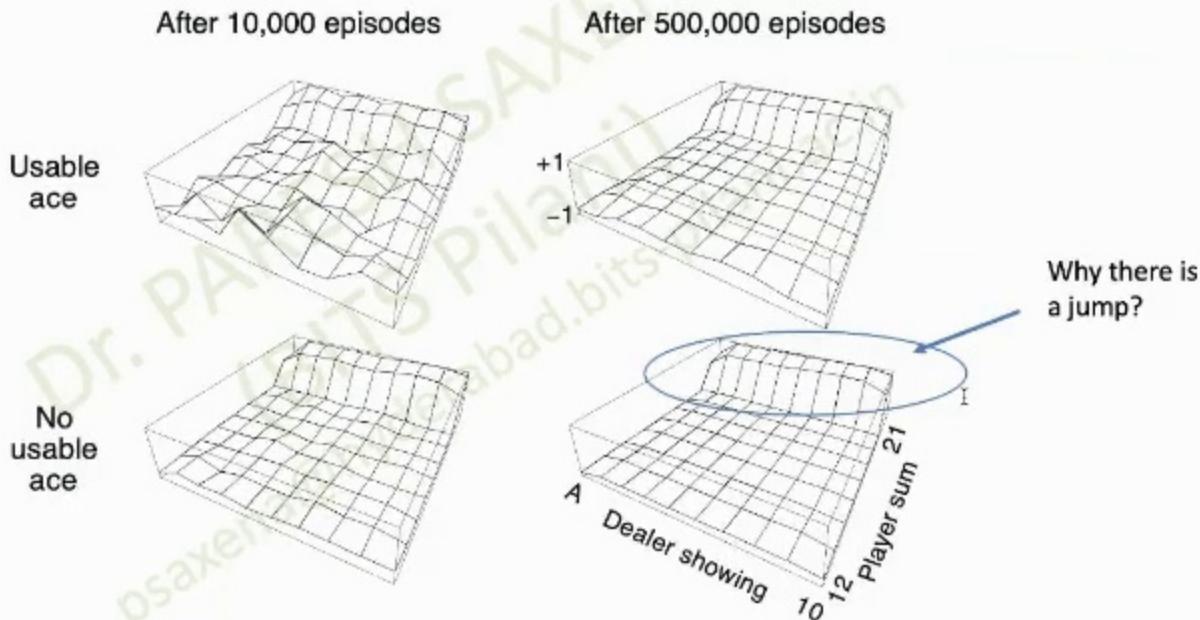
First player samples card from deck (hit) until stop (stick)

Then, dealer samples card from deck (hit) until sum > 16 (stick)

- Total states = 200 3 tuple: (player sum, dealer's card, holding of ace)
- Action = {hit, stick}
- Rewards = {-1, 0, 1}

# Example: Evaluation of Policy

Policy: Player sticks if sum is 20 or 21 or else hit



# Blackjack and DP

- In general, we can know complete MDP for blackjack and transition probabilities
- But computing transition probabilities can be complicated and hence DP can be tricky.
- MC methods will employ simulations and independent episodes
- MC can evaluate policy without knowledge of probabilities.

I

# Example: Blackjack – Total States

innovate

achieve

lead

Ace worth 1 or 11



All face cards count is 10.



Player

Dealer

First player samples card from deck (hit) until stop (stick)

Then, dealer samples card from deck (hit) until sum > 16 (stick)

- If we formulate blackjack as MDP, what are the number of states ? (assume that if player has sum less than 11 then it will always ask for hit, so need not to count that as states).

# Moving towards optimal policies: MC for action-value vs state-value

- In previous chapter, we assume complete knowledge of environment hence transition probabilities.
- In previous chapter, using state value estimates and transition probability, we estimated the optimal action.
- However, the complete model is not known in MC.
- Policy evaluation problem is therefore to estimate  $q_{\pi}(s, a)$ , the expected return when starting in state  $s$ , taking action  $a$  and following policy  $\pi$ .

1

psaxena@hyderabad.bits-pilani.ac.in

# Some issues with MC methods

Lack of Exploration while considering Episodes !!

Assume a Deterministic Policy for a Grid Problem.

- **Deterministic Policy:**

- First Preference – going down until edge of grid
- Second Preference - going left until edge of grid
- Third Preference: going right until edge of grid
- Fourth Preference: going up until edge of grid

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

# Some issues with MC methods

Lack of Exploration while considering Episodes !!

I

Assume a Deterministic Policy for a Grid Problem.

- Deterministic Policy:
  - First Choice – To go down until edge, then go left until edge, then go up until edge and then go right until edge
  - Starting from state 1 (with only one action), only one episode is being generated
- Every time if we start with 1, only 1, 5, 9, 13, 12, 8 and 4 will be visited. What about the other states ?
- At each state only one action is being taken.
- What we need to consider other state-action pairs ?

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	



## blem

the opportunity to make bets on the outcomes of a sequence of coin flips.

comes up heads, he wins as many dollar as he has staked on that flip;

, he loses his stake.

ends when the gambler wins by reaching his goal of \$100, or loses by running out of money.

flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars.

the gambler's capital  $s = \{0, 1, 2, 3, \dots, 100\}$

stakes  $a = \{1, 2, \dots, \min(s, 100-s)\}$

zero on all transitions except those on which the gambler reaches his goal, when it is +1.

the probability of the coin coming up heads

function then gives the probability of winning from each state.

mapping from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal.

ray  $V$  arbitrarily (e.g.,  $V(s) = 0$  for all  $s \in S^+$ )

$s \in S:$

$V(s)$

## Gambler's Problem



File Insert Runtime Tools Help All changes saved

## Comment

ment

## Problem

→ ↑ ↓ ⌂

: the opportunity to make bets on the outcomes of a sequence of coin flips.

If he comes up heads, he wins as many dollar as he has staked on that flip;

is, he loses his stake.

The game ends when the gambler wins by reaching his goal of \$100, or loses by running out of money.

flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars.

The gambler's capital  $s = \{0, 1, 2, 3, \dots, 100\}$

The stakes  $a = \{1, 2, \dots, \min(s, 100-s)\}$

zero on all transitions except those on which the gambler reaches his goal, when it is +1.

the probability of the coin coming up heads.

a function that gives the probability of winning from each state.

<sup>1</sup> ranging from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal.

draw  $V$  arbitrarily (e.g.,  $V(s) \equiv 0$  for all  $s \in S^+$ )

$$s \in S$$

V(%)

$$\leftarrow \max_a \sum_{s' \in \mathcal{S}} p(s', r | s, a) [r + \gamma V(s')]$$

## Gambler's Problem

v Insert Runtime Tools Help All changes saved

Comment

Connect

: the opportunity to make bets on the outcomes of a sequence of coin flips.

When it comes up heads, he wins as many dollar as he has staked on that flip;

tails, he loses his stake.

The game ends when the gambler wins by reaching his goal of \$100, or loses by running out of money.

At each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars.

The gambler's capital  $s = \{0, 1, 2, 3, \dots, 100\}$

He stakes  $a = \{1, 2, \dots, \min(s, 100-s)\}$

Probability is zero on all transitions except those on which the gambler reaches his goal, when it is +1.

The probability of the coin coming up heads

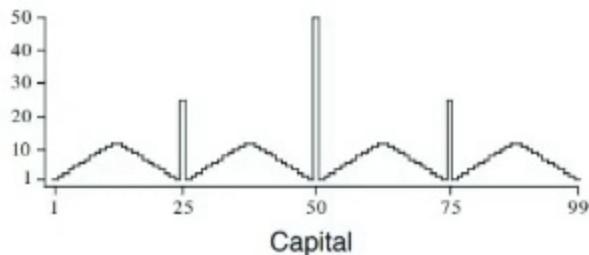
The function then gives the probability of winning from each state.

Mapping from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal.

Set initial value of array  $V$  arbitrarily (e.g.,  $V(s) = 0$  for all  $s \in \mathcal{S}^+$ )

$s \in \mathcal{S}$ :

$V(s)$   
 $\leftarrow \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$   
 $\max(\Delta, |v - V(s)|)$

Final  
policy  
(stake)

Expected Output

25	50	75
25	50	25
Stake - 25	Stake - 50	Stake - 25
Left - 0	Left - 0	Left - 50
Win - 25+25=50	Win-50+50=100	Win -75+25=100
Lose - 0	Lose - 0	Lose - 50

	27	43	55	70	80	93
2	7	5	5	5	7	
-12	Stake - 2	Stake - 7	Stake - 5	Stake - 5	Stake - 7	
-1	Left - 25	Left - 36	Left - 50	Left - 65	Left - 75	Left - 86
-1	Win-27+2 = 29	Win-43+7 = 50	Win- 55+5=60	Win- 70+5=75	Win-80+5=90	Win-93+7=100
-1	Lose - 25	Lose - 36	Lose - 50	Lose-70-5=65	Lose-75	Lose-86

## 3: Gambler's Problem ☆

Comment

View Insert Runtime Tools Help All changes saved

ext

Connect

Lose - 1	Lose - 25	Lose - 36	Lose - 50	Lose-70-5=65	Lose-75	Lose-86
----------	-----------	-----------	-----------	--------------	---------	---------

tion

```
numpy as np
sys
matplotlib.pyplot as plt
```

ment Value Iteration

```
, v = value_iteration_for_gamblers(0.25)

"Optimized Policy:")
policy)
"")

"Optimized Value Function:")
v)
"")
```

ting Final Policy (action stake) vs State (Capital)  
graph shows the value function found by successive sweeps of value iteration

### 3: Gambler's Problem

View Insert Runtime Tools Help All changes saved

Comment

ext

has the opportunity to make bets on the outcomes or a sequence of coin flips.

Connect

↑ ↓ ↻

coin comes up heads, he wins as many dollar as he has staked on that flip;

tails, he loses his stake.

Game ends when the gambler wins by reaching his goal of \$100, or loses by running out of money.

At each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars.

the gambler's capital  $s = \{0, 1, 2, 3, \dots, 100\}$

possible stakes  $a = \{1, 2, \dots, \min(s, 100-s)\}$

is zero on all transitions except those on which the gambler reaches his goal, when it is +1.

Note the probability of the coin coming up heads

value function then gives the probability of winning from each state.

mapping from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal.

array  $V$  arbitrarily (e.g.,  $V(s) = 0$  for all  $s \in S^+$ )

such  $s \in S$ :  
 $- V(s)$



### 3: Gambler's Problem

View Insert Runtime Tools Help

Comment

ext

RAM  
Disk

#### problem

has the opportunity to make bets on the outcomes of a sequence of coin flips.

coin comes up heads, he wins as many dollar as he has staked on that flip;

tails, he loses his stake.

ame ends when the gambler wins by reaching his goal of \$100, or loses by running out of money.

ch flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars.

the gambler's capital  $s \in \{0,1,2,3,\dots,100\}$

are stakes  $a \in \{1, 2, \dots, \min(s, 100-s)\}$

is zero on all transitions except those on which the gambler reaches his goal, when it is +1.

value function then gives the probability of winning from each state.

mapping from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal.

ote the probability of the coin coming up heads.

e

array  $V$  arbitrarily (e.g.,  $V(s) = 0$  for all  $s \in S^c$ )

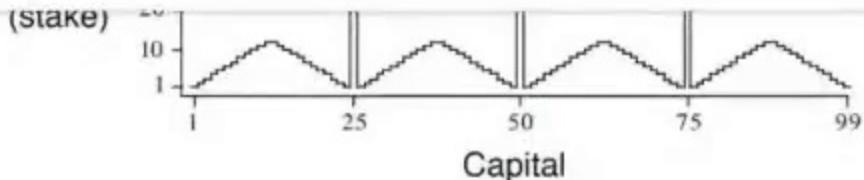
## 3: Gambler's Problem

Comment

View Insert Runtime Tools Help

RAM Disk

ext



of Expected Output

25	50	75
25	50	25
Stake - 25 Left - 0	Stake - 50 Left - 0	Stake - 25 Left - 50
Win - 25+25=50 Lose - 0	Win-50+50=100 Lose - 0	Win -75+25=100 Lose - 50

13	27	43	55	70	80	93
12	2	7	5	5	5	7
Stake - 12 Left - 1	Stake - 2 Left - 25	Stake - 7 Left - 36	Stake -5 Left - 50	Stake - 5 Left- 65	Stake - 5 Left - 75	Stake - 7 Left - 86
Win-13+12= 25 Lose - 1	Win-27+2 = 29 Lose - 25	Win-43+7= 50 Lose - 36	Win- 55+5=60 Lose - 50	Win- 70+5=75 Lose-70-5=65	Win-80+5=85 Lose-75	Win-93+7=100 Lose-86

✓ 0s completed at 12:31 PM

## 3: Gambler's Problem ★

Comment

View Insert Runtime Tools Help All changes saved

Disk

ext

RAM

"Optimized Value Function:")

v)

")

zed Policy:

```
1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17.  
6. 20. 21. 3. 23. 24. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.  
2. 38. 11. 10. 9. 42. 7. 44. 5. 46. 47. 48. 49. 50. 1. 2. 3.  
5. 6. 7. 8. 9. 10. 11. 12. 13. 11. 10. 9. 17. 7. 19. 5. 21.  
3. 24. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 12. 11.  
9. 8. 7. 6. 5. 4. 3. 2. 1.]
```

zed Value Function:

```
00000e+00 7.24792480e-05 2.89916992e-04 6.95257448e-04  
10383e-03 7.76906586e-03 2.78102979e-03 4.03504074e-03  
14120e-03 5.59997559e-03 7.08471239e-03 9.03964043e-03  
41192e-02 1.56793594e-02 1.61464431e-02 1.69517994e-02  
12806e-02 1.98249817e-02 2.24047303e-02 2.73845196e-02  
88495e-02 3.04937363e-02 3.61633897e-02 3.84953022e-02  
64767e-02 6.25000000e-02 6.27174377e-02 6.33700779e-02  
57723e-02 6.59966059e-02 6.78135343e-02 7.08430894e-02  
98323e-02 7.64884604e-02 7.93035477e-02 8.37541372e-02  
54232e-02 9.50723575e-02 1.09538078e-01 1.10939329e-01  
69151e-01 1.18457374e-01 1.21977661e-01 1.29716997e-01  
53559e-01 1.47520113e-01 1.53983246e-01 1.70990169e-01  
87434e-01 1.95990576e-01 2.50000000e-01 2.58217438e-01  
70078e-01 2.52085772e-01 2.53496606e-01 2.55313534e-01  
43089e-01 2.62109832e-01 2.63988460e-01 2.66803548e-01  
54137e-01 2.77122542e-01 2.83372357e-01 2.97038078e-01  
39329e-01 3.00860151e-01 3.05957374e-01 3.09477661e-01  
16907e-01 3.32153559e-01 3.35020113e-01 3.41483246e-01  
90169e-01 3.65487434e-01 3.83490576e-01 4.37500000e-01
```

0s completed at 12:31 PM

## 3: Gambler's Problem ★

View Insert Runtime Tools Help All changes saved

Comment



ext

RAM Disk

↑ ↓ ⏪ ⏩

25	50	75
25	50	25
Stake - 25 Left - 0	Stake - 50 Left - 0	Stake - 25 Left - 50
Win - 25+25=50 Lose - 0	Win-50+50=100 Lose - 0	Win - 75+25=100 Lose - 50

13	27	43	55	70	80	93
12	2	7	5	5	5	7
Stake - 12 Left - 1	Stake - 2 Left - 25	Stake - 7 Left - 36	Stake - 5 Left - 50	Stake - 5 Left - 65	Stake - 5 Left - 75	Stake - 7 Left - 86
Win-13+12= 25 Lose - 1	Win-27+2 = 29 Lose - 25	Win-43+7= 50 Lose - 36	Win- 55+5=60 Lose - 50	Win- 70+5+75 Lose-70-5=65	Win-80+5=90 Lose-75	Win-93+7=100 Lose-86

```
numpy as np
sys
matplotlib.pyplot as plt
```

```
lue_iteration_for_gamblers(p_h, theta=0.0001, discount_factor=1.0):
```

```
wards = np.zeros(101) #reward is zero for all transitions
wards[100] = 1 #except when gambler reach goal
= np.zeros(101) #Value of each state is initialized as 0 in the beginning
```

✓ 0s completed at 12:31 PM

## 3: Gambler's Problem ★

Comment

View Insert Runtime Tools Help All changes saved

ext

✓ RAM Disk

```
turn policy, v
, v = value_iteration_for_gamblers(0.25)

"Optimized Policy:")
policy)
"")

"Optimized Value Function:")
v)
"")

zed Policy:
1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17.
6. 20. 21. 3. 23. 24. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.
2. 38. 11. 10. 9. 42. 7. 44. 5. 46. 47. 48. 49. 50. 1. 2. 3.
5. 6. 7. 8. 9. 18. 11. 12. 13. 11. 10. 9. 17. 7. 19. 5. 21.
3. 24. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 11.
9. 8. 7. 6. 5. 4. 3. 2. 1.]
```

zed Value Function:

	0.00000e+00	7.24792480e-05	2.89916992e-04	6.95257448e-04	1.0383e-03	1.76906586e-03	2.78102979e-03	4.03504074e-03	5.59997559e-03	7.08471239e-03	9.03964043e-03	1.1209e-02	1.56793594e-02	1.61464431e-02	1.69517994e-02	1.2806e-02	1.98249817e-02	2.24047303e-02	2.73845196e-02	8.8495e-02	3.04937363e-02	3.61633897e-02	3.84953022e-02	6.4767e-02	6.25000000e-02	6.27174377e-02	6.33700779e-02	5.7723e-02	6.59966059e-02	6.78135343e-02	7.08430894e-02	9.8323e-02	7.64884604e-02	7.93035477e-02	8.37541372e-02
--	-------------	----------------	----------------	----------------	------------	----------------	----------------	----------------	----------------	----------------	----------------	------------	----------------	----------------	----------------	------------	----------------	----------------	----------------	------------	----------------	----------------	----------------	------------	----------------	----------------	----------------	------------	----------------	----------------	----------------	------------	----------------	----------------	----------------

✓ 0s completed at 12:31 PM

## 3: Gambler's Problem

Comment

View Insert Runtime Tools Help All changes saved

RAM Disk

ext

| LOGO - v | LOGO - v | LOGO - w |

13	27	43	55	70	80	93
12	2	7	5	5	5	7
Stake - 12	Stake - 2	Stake - 7	Stake - 5	Stake - 5	Stake - 7	Stake - 7
Left - 1	Left - 25	Left - 36	Left - 50	Left - 65	Left - 75	Left - 86
Win-13+12= 25	Win-27+2 = 29	Win-43+7= 50	Win- 55+5=60	Win- 70+5+75	Win-80+5=90	Win-93+7=100
Lose - 1	Lose - 25	Lose - 36	Lose - 50	Lose-70-5=65	Lose-75	Lose-86

```
numpy as np
sys
matplotlib.pyplot as plt
```

```
lue_iteration_for_gamblers(p_h, theta=0.0001, discount_factor=1.0):
```

```
wards = np.zeros(101) #reward is zero for all transitions
wards[100] = 1 #except when gambler reach goal
= np.zeros(101) #Value of each state is initialized as 0 in the beginning
```

```
f one_step_lookahead(s, v, rewards): #Bellman equation
A = np.zeros(101)
for a in range(1, min(s, 100-s)+1):
    A[a] = p_h * (rewards[s+a] + V[s+a]*discount_factor) + (1-p_h) * (rewards[s-a] + V[s-a]*discount_factor) #the sum of immediate rewards and the value of the
return A #Returns a vector containing the expected value of each action
```

✓ 0s completed at 12:31 PM

## 3: Gambler's Problem ★

Comment

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```
turn policy, v
    , v = value_iteration_for_gamblers(0.25)

"Optimized Policy:")
policy)
"")

"Optimized Value Function:")
v)
"")

zed Policy:
1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 12. 11. 15. 16. 17.
6. 20. 21. 3. 23. 24. 25. 1. 12. 3. 4. 5. 6. 7. 8. 9. 10.
2. 38. 11. 10. 9. 42. 7. 44. 5. 46. 47. 48. 49. 50. 1. 2. 3.
5. 6. 7. 8. 9. 10. 11. 12. 13. 11. 10. 9. 17. 7. 19. 5. 21.
3. 24. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 12. 11.
9. 8. 7. 6. 5. 4. 3. 2. 1.]
```

```
zed Value Function:
00000e+00 7.24792400e-05 2.89916992e-04 6.95257448e-04
10383e-03 1.76906586e-03 2.78102979e-03 4.03504074e-03
14120e-03 5.59997559e-03 7.08471239e-03 9.03964043e-03
41192e-02 1.56793594e-02 1.61464431e-02 1.69517994e-02
12806e-02 1.98249817e-02 2.24047303e-02 2.73845196e-02
88495e-02 3.04937363e-02 3.61633897e-02 3.84953022e-02
64767e-02 6.25000000e-02 6.27174377e-02 6.33700779e-02
57723e-02 6.59966059e-02 6.78135343e-02 7.08430894e-02
```

0s completed at 12:31 PM

## 3: Gambler's Problem

Comment

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

sys  
matplotlib.pyplot as plt

lue\_iteration\_for\_gamblers(p\_h, theta=0.0001, discount\_factor=1.0):

wards = np.zeros(101) #reward is zero for all transitions

wards[100] = 1 #except when gambler reach goal

= np.zeros(101) #Value of each state is initialized as 0 in the beginning

f one\_step\_lookahead(s, V, rewards): #Bellman equation

A = np.zeros(101)

for a in range(1, min(s, 100-s)+1):

A[a] = p\_h \* (rewards[s+a] + V[s+a]\*discount\_factor) + (1-p\_h) \* (rewards[s-a] + V[s-a]\*discount\_factor) #the sum of immediate rewards and the value of the next state

return A #Returns a vector containing the expected value of each action

ile True:

delta = 0

for s in range(1, 100):

best\_action\_value = V[s]

V[s] = np.max(one\_step\_lookahead(s, V, rewards))

delta = max(delta, np.abs(best\_action\_value - V[s]))

if delta &lt; theta:

break

licy = np.zeros(100)

r s in range(1, 100):

policy[s] = np.argmax(one\_step\_lookahead(s, V, rewards))

0s completed at 12:33 PM

### 3: Gambler's Problem

Comment

View Insert Runtime Tools Help All changes saved

RAM

Disk

ext

```
policy[s] = np.argmax(one_step_lookahead(s, v, rewards))
```

turn policy, v

↑ ↓ ⌂

```
, v = value_iteration_for_gamblers(0,1)
```

"Optimized Policy:")

policy

")

"Optimized Value Function:")

v

")

zed Policy:

```
1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 9. 17.  
9. 20. 21. 22. 23. 24. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.  
2. 38. 39. 40. 9. 8. 43. 44. 45. 4. 47. 2. 1. 59. 1. 2. 3.  
5. 6. 7. 8. 41. 10. 11. 12. 13. 14. 15. 34. 8. 18. 19. 20. 4.  
2. 1. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 11.  
9. 8. 7. 6. 5. 4. 3. 2. 1.]
```

zed Value Function:

```
0.00283162 0.00515507 0.00922512 0.01290018 0.01738288  
0.02781403 0.03227452 0.03787825 0.04346882 0.05035153  
0.05757 0.06521892 0.06953587 0.07442925 0.08068842 0.08664695  
0.1092 0.10912138 0.10065755 0.11596817 0.12587883 0.1335785  
0.1471 0.16 0.16309384 0.16774251 0.17383767 0.17936474  
0.0749 0.19459454 0.20172104 0.20841385 0.21652655 0.22519453  
0.1773 0.1869076 0.1976897 0.2087606 0.22588489 0.18143943
```

✓ 0s completed at 12:33 PM

## 3: Gambler's Problem



View Insert Runtime Tools Help All changes saved

Comment



ext

✓ RAM Disk

```
lue_iteration_for_gamblers(p_h, theta=0.0001, discount_factor=1.0):

wards = np.zeros(101) #reward is zero for all transitions
wards[100] = 1 #except when gambler reach goal
= np.zeros(101) #value of each state is initialized as 0 in the beginning

f one_step_lookahead(s, V, rewards): #Bellman equation
A = np.zeros(101)
for a in range(1, min(s, 100-s)+1):
    A[a] = p_h * (rewards[s+a] + V[s+a]*discount_factor) + (1-p_h) * (rewards[s-a] + V[s-a]*discount_factor) #the sum of immediate rewards and the value of the next state
return A #Returns a vector containing the expected value of each action

ile True:
delta = 0
for s in range(1, 100):
    best_action_value = V[s]
    V[s] = np.max(one_step_lookahead(s, V, rewards))
    delta = max(delta, np.abs(best_action_value - V[s]))
if delta < theta:
    break

licy = np.zeros(100)
r s in range(1, 100):
    policy[s] = np.argmax(one_step_lookahead(s, V, rewards))

turn policy, V
```

✓ 0s completed at 12:33 PM



## 3: Gambler's Problem

Comment

View Insert Runtime Tools Help All changes saved

RAM Disk

```

ext
v = np.zeros(101)      #Value of each state is initialized as 0 in the beginning

def one_step_lookahead(s, V, rewards):
    A = np.zeros(101)
    for a in range(1, min(s, 100-s)+1):
        A[a] = p_h * (rewards[s+a] + V[s+a]*discount_factor) + (1-p_h) * (rewards[s-a] + V[s-a]*discount_factor) #the sum of immediate rewards and the value of the next state
    return A      #Returns a vector containing the expected value of each action

while True:
    delta = 0
    for s in range(1, 100):
        best_action_value = V[s]
        V[s] = np.max(one_step_lookahead(s, V, rewards))
        delta = max(delta, np.abs(best_action_value - V[s]))
    if delta < theta:
        break

policy = np.zeros(100)
for s in range(1, 100):
    policy[s] = np.argmax(one_step_lookahead(s, V, rewards))

return policy, V
, v = value_iteration_for_gamblers(0.4)

"Optimized Policy:")
policy)
""")

```

✓ 0s completed at 12:33 PM

## 3: Gambler's Problem ★

Comment

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```

value_iteration_for_gamblers(p_h, theta=0.0001, discount_factor=1.0):

    rewards = np.zeros(101) #reward is zero for all transitions
    rewards[100] = 1 #except when gambler reach goal
    - np.zeros(101) #value of each state is initialized as 0 in the beginning

def one_step_lookahead(s, V, rewards):
    A = np.zeros(101)
    for a in range(1, min(s, 100-s)+1):
        A[a] = p_h * (rewards[s+a] + V[s+a]*discount_factor) + (1-p_h) * (rewards[s-a] + V[s-a]*discount_factor) #the sum of immediate rewards and the value of the next state
    return A #Returns a vector containing the expected value of each action

while True:
    delta = 0
    for s in range(1, 100):
        best_action_value = V[s]
        V[s] = np.max(one_step_lookahead(s, V, rewards))
        delta = max(delta, np.abs(best_action_value - V[s]))
    if delta < theta:
        break

policy = np.zeros(100)
for s in range(1, 100):
    policy[s] = np.argmax(one_step_lookahead(s, V, rewards))

return policy, V

```

✓ 0s completed at 12:33 PM

## 3: Gambler's Problem ★

Comment

View Insert Runtime Tools Help All changes saved

RAM Disk

ext

```
9. 20. 21. 22. 23. 24. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.
2. 38. 39. 40. 9. 8. 43. 44. 45. 4. 47. 2. 1. 50. 1. 2. 3.
5. 6. 7. 8. 41. 18. 11. 12. 13. 14. 15. 34. 8. 18. 19. 20. 4.
2. 1. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 12. 11.
9. 8. 7. 6. 5. 4. 3. 2. 1.]
```

zed Value Function:

```
0.00203162 0.00515507 0.00922512 0.01290418 0.01738208
0.02781483 0.03227457 0.03767825 0.04346082 0.05035153
0.06521897 0.06953507 0.07442925 0.08068842 0.08660695
0.1092 0.10313138 0.10865755 0.11596417 0.12587883 0.1335785
0.1471 0.16 0.16309304 0.16774251 0.17383767 0.17936474
0.17649 0.19459454 0.20172104 0.20841305 0.21652655 0.22519453
0.2523 0.24648826 0.25785582 0.2643026 0.27164589 0.28103263
0.31592 0.30131638 0.31471349 0.32298754 0.33394956 0.3488281
0.36974 0.37622184 0.4 0.40309304 0.40774251 0.41383767
0.36474 0.42607649 0.43459454 0.44172104 0.44841305 0.45652655
0.19453 0.4755273 0.48648826 0.49785582 0.5043026 0.51164589
0.03263 0.52991593 0.54131638 0.55471349 0.56298754 0.57394956
0.8281 0.60036974 0.61622184 0.64 0.6446455 0.65161885
0.75673 0.66984783 0.67911672 0.69189296 0.70258156 0.71261958
0.78983 0.73779252 0.75329686 0.7697331 0.7867873 0.79645404
0.47003 0.82154894 0.8348739 0.85197811 0.87207238 0.88448282
0.92434 0.92324343 0.9405546 0.96433276 0. ]
```

ting Final Policy (action stake) vs State (Capital)

```
title('Final Policy (action stake) vs State (Capital)') # title to the graph
xlabel('Capital') # naming the x axis
ylabel('Value Estimates') # naming the y axis
```

✓ 0s completed at 12:33 PM

## 3: Gambler's Problem

View Insert Runtime Tools Help All changes saved

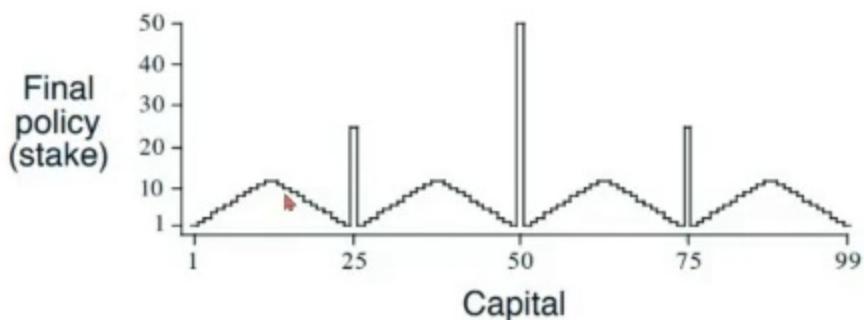
Comment



RAM Disk

Figure 4.5: Value iteration.

Output



of Expected Output

25	50	75
25	50	25

✓ 0s completed at 12:33 PM

## 3: Gambler's Problem

Comment

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```

def value_iteration_for_gamblers(p_h, theta=0.0001, discount_factor=1.0):
    rewards = np.zeros(101) #reward is zero for all transitions
    rewards[100] = 1 #except when gambler reach goal
    v = np.zeros(101) #value of each state is initialized as 0 in the beginning

    def one_step_lookahead(s, V, rewards):
        A = np.zeros(101)
        for a in range(1, min(s, 100-s)+1):
            A[a] = p_h * (rewards[s+a] + V[s+a]*discount_factor) + (1-p_h) * (rewards[s-a] + V[s-a]*discount_factor) #the sum of immediate rewards and the value of the next state
        return A #Returns a vector containing the expected value of each action

    i = True
    delta = 0
    for s in range(1, 100):
        best_action_value = v[s]
        V[s] = np.max(one_step_lookahead(s, v, rewards))
        delta = max(delta, np.abs(best_action_value - V[s]))
    if delta < theta:
        break

    policy = np.zeros(100)
    for s in range(1, 100):
        policy[s] = np.argmax(one_step_lookahead(s, v, rewards))

    return policy, v

```

0s completed at 12:33 PM

## 3: Gambler's Problem ★

Comment

View Insert Runtime Tools Help All changes saved

ext

RAM Disk

```

value_iteration_for_gamblers(p_h, theta=0.0001, discount_factor=1.0):

rewards = np.zeros(101)    #reward is zero for all transitions
rewards[100] = 1            #except when gambler reach goal
v = np.zeros(101)           #value of each state is initialized as 0 in the beginning

def one_step_lookahead(s, v, rewards):
    """Bellman equation"""
    A = np.zeros(101)
    for a in range(1, min(s, 100-s)+1):
        A[a] = p_h * (rewards[s+a] + v[s+a]*discount_factor) + (1-p_h) * (rewards[s-a] + v[s-a]*discount_factor) #the sum of immediate rewards and the value of the next state
    return A      #Returns a vector containing the expected value of each action

while True:
    delta = 0
    for s in range(1, 100):
        best_action_value = v[s]
        v[s] = np.max(one_step_lookahead(s, v, rewards))
        delta = max(delta, np.abs(best_action_value - v[s]))
    if delta < theta:
        break

policy = np.zeros(100)
for s in range(1, 100):
    policy[s] = np.argmax(one_step_lookahead(s, v, rewards))

return policy, v

```

✓ 0s completed at 12:33 PM

## 3: Gambler's Problem

View Insert Runtime Tools Help All changes saved

Comment



ext

47003 0.82154894 0.8348739 0.85197811 0.87207238 0.88448282  
92434 0.92324343 0.9405546 0.96433276 0. ]

RAM

Disk



ting Final Policy (action stake) vs State (Capital)

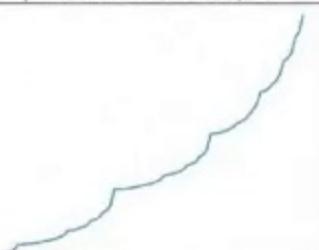
```
title('Final Policy (action stake) vs State (Capital)' ) # title to the graph
label('Capital') # naming the x axis
label('Value Estimates') # naming the y axis

range(100) # x axis values
:100] # y axis values

plot(x, y) # plotting the points

ow() # function to show the plot
```

Final Policy (action stake) vs State (Capital)



✓ 0s completed at 12:33 PM

## 3: Gambler's Problem

Comment

View Insert Runtime Tools Help All changes saved

RAM Disk

ext

```

5273 0.24648826 0.25785582 0.2643026 0.27164589 0.28103263
91593 0.90131638 0.31471349 0.32298758 0.33394956 0.3488281
36974 0.37622184 0.4 0.40309394 0.40774251 0.41383767
36474 0.426687649 0.43459454 0.44172184 0.44841305 0.45652655
19453 0.4755273 0.48648826 0.49785582 0.5043026 0.51164589
01263 0.52991593 0.54131638 0.55471389 0.56298758 0.57394956
8281 0.68036974 0.61622184 0.64 0.6446455 0.65161885
75673 0.66984783 0.67911672 0.69189296 0.70258156 0.71261958
78983 0.73779252 0.75329686 0.7697331 0.7867873 0.79645404
47003 0.82154894 0.8348739 0.85197811 0.87207256 0.88447282
92434 0.92324143 0.9405546 0.96433276 0. ]
```

ting Final Policy (action stake) vs State (capital)

```

title('Final Policy (action stake) vs state (Capital)') # title to the graph
xlabel('Capital') # naming the x axis
ylabel('Value Estimates') # naming the y axis
xrange(100) # x axis values
yrange(100) # y axis values
plot(x, y) # plotting the points
show() # function to show the plot
```

Final Policy (action stake) vs State (Capital)



✓ 0s completed at 12:33 PM

## 3: Gambler's Problem

View Insert Runtime Tools Help All changes saved

Comment

ext

RAM Disk

```
, v = value_iteration_for_gamblers(0.4)

"Optimized Policy:"
policy
"")

"Optimized Value Function:"
v)
"")

zed Policy:
1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 11. 15. 9. 17.
9. 20. 21. 22. 23. 24. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.
2. 38. 39. 40. 9. 8. 43. 44. 45. 4. 47. 2. 1. 58. 1. 2. 3.
5. 6. 7. 8. 41. 10. 11. 12. 13. 14. 15. 34. 8. 18. 19. 20. 4.
2. 1. 25. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 12. 11.
9. 8. 7. 6. 5. 4. 3. 2. 1.]
```

```
zed Value Function:
0.00283162 0.00515507 0.00922512 0.01290418 0.01738208
0.0279 0.02781403 0.03227457 0.03767825 0.04346082 0.05035153
0.06521897 0.06953507 0.07442925 0.08060842 0.08660695
0.1092 0.10313138 0.10865755 0.11596417 0.12587883 0.1335785
0.1471 0.16 0.16309304 0.16774251 0.17383767 0.17936474
0.17649 0.19459454 0.20172104 0.20841305 0.21652655 0.22519453
0.2273 0.24648826 0.25785582 0.2643026 0.27164589 0.28103263
0.29593 0.30131638 0.31471349 0.32298754 0.33394956 0.3488281
0.36974 0.37622184 0.4 0.40309304 0.40774251 0.41383767
0.36474 0.42607649 0.43459454 0.44172104 0.44841305 0.45652655
```

✓ 0s completed at 12:33 PM



# Some issues with MC methods

innovate

achieve

lead

Lack of Exploration while considering Episodes !!

Assume a Deterministic Policy for a Grid Problem.

- **Deterministic Policy:**

- First Preference – going down until edge of grid
- Second Preference - going left until edge of grid
- Third Preference: going right until edge of grid
- Fourth Preference: going up until edge of grid

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

# Some issues with MC methods

Lack of Exploration while considering Episodes !!

Assume a Deterministic Policy for a Grid Problem.

- Deterministic Policy:
    - First Choice – To go down until edge, then go left until edge, then go up until edge and then go right until edge
    - Starting from state 1 (with only one action), only one episode is being generated
- |    |    |    |    |
|----|----|----|----|
|    | 1  | 2  | 3  |
| 4  | 5  | 6  | 7  |
| 8  | 9  | 10 | 11 |
| 12 | 13 | 14 |    |

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	
- Every time if we start with 1, only 1, 5, 9, 13, 12, 8 and 4 will be visited. What about the other states ?
  - At each state only one action is being taken.
  - What we need to consider other state-action pairs ?

# Some issues with MC methods

innovate

achieve

lead

Lack of Exploration while considering Episodes !!

Assume a Deterministic Policy for a Grid Problem.

- **Deterministic Policy:**

- First Preference – going down until edge of grid
- Second Preference - going left until edge of grid
- Third Preference: going right until edge of grid
- Fourth Preference: going up until edge of grid

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

# Some issues with MC methods

Lack of Exploration while considering Episodes !!

Assume a Deterministic Policy for a Grid Problem.

- Deterministic Policy:
    - First Choice – To go down until edge, then go left until edge, then go up until edge and then go right until edge
    - Starting from state 1 (with only one action), only one episode is being generated
- |    |    |    |    |
|----|----|----|----|
|    | 1  | 2  | 3  |
| 4  | 5  | 6  | 7  |
| 8  | 9  | 10 | 11 |
| 12 | 13 | 14 |    |

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	
- Every time if we start with 1, only 1, 5, 9, 13, 12, 8 and 4 will be visited. What about the other states ?
  - At each state only one action is being taken.
  - What we need to consider other state-action pairs ?

# Exploring Start

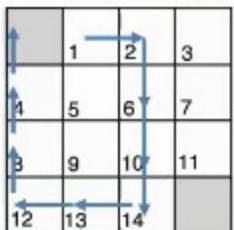
- Initialize with one (state, action) pair
- All (state, action) pairs have non-zero probability to be selected. Each time start with a different (state, action) pair and then evaluate the policy.
- Lets evaluate the policy again. Deterministic Policy:
  - First Choice – To go down until edge, then go left until edge, then go up until edge and then go right until edge

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

- 14 states
- 4 actions
- Total 56 pairs

# Exploring Start – Example (Episode 1)

- Initial  $(s,a) \rightarrow 1$ , right
- Policy: First Choice – To go down until edge, then go left until edge, then go up until edge and then go right until edge



	L	R	U	D
1		X		
2				X
3				
4				X
5				
6				X
7				
8				X
9				
10				X
11				
12	X			X
13	X			
14	X			X

# Some issues with MC methods

innovate

achieve

lead

Lack of Exploration while considering Episodes !!

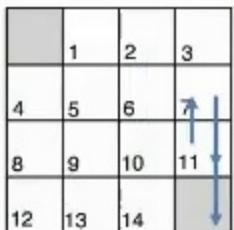
Assume a Deterministic Policy for a Grid Problem.

- Deterministic Policy:
    - First Choice – To go down until edge, then go left until edge, then go up until edge and then go right until edge
    - Starting from state 1 (with only one action), only one episode is being generated
  - Every time if we start with 1, only 1, 5, 9, 13, 12, 8 and 4 will be visited. What about the other states ?
  - At each state only one action is being taken.
  - What we need to consider other state-action pairs ?
- 
- |    |    |    |    |
|----|----|----|----|
|    | 1  | 2  | 3  |
| 4  | 5  | 6  | 7  |
| 8  | 9  | 10 | 11 |
| 12 | 13 | 14 |    |
- |    |    |    |    |
|----|----|----|----|
|    | 1  | 2  | 3  |
| 4  | 5  | 6  | 7  |
| 8  | 9  | 10 | 11 |
| 12 | 13 | 14 |    |
- |    |    |    |    |
|----|----|----|----|
|    | 1  | 2  | 3  |
| 4  | 5  | 6  | 7  |
| 8  | 9  | 10 | 11 |
| 12 | 13 | 14 |    |

# Exploring Start – Example (Episode 2)

innovate achieve lead

- Initial  $(s,a) \rightarrow 11$ , up
- Policy: First Choice – To go down until edge, then go left until edge, then go up until edge and then go right until edge



	L	R	U	D
1			X	
2				X
3				
4				X
5				
6				X
7				X1
8			X	
9				
10				X
11			X1	X1
12	X			X
13	X			
14	X			X

# Monte-Carlo Control - ES

- Optimal Policy Estimate by Evaluation and Improvement.

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  $s \in \mathcal{S}, a \in \mathcal{A}(s)$

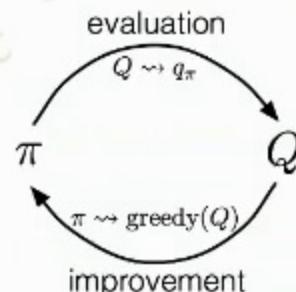
Loop forever (for each episode):

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  $\pi$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$   
 $G \leftarrow 0$

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

$G \leftarrow \gamma G + R_{t+1}$   
Unless the pair  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ :  
Append  $G$  to  $Returns(S_t, A_t)$   
 $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$   
 $\pi(S_t) \leftarrow \arg \max_a Q(S_t, a)$

Improving the policy for  
each episode.

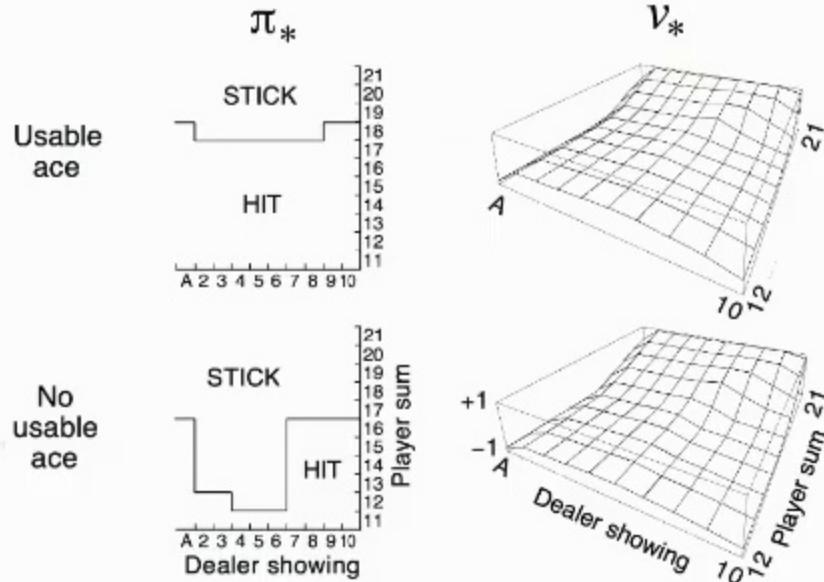


# Optimal policy for Blackjack with MC-ES

innovate

achieve

lead



# MC without exploring starts: On-policy vs Off-policy methods

---

- Exploring start can not be generalized.
- On-policy methods: Improve the policy that is used to make decisions, e.g., Monte Carlo methods
- Off-policy methods: Improve the policy that is different than the policy that is used to generate the data.

Dr. Shekhar Saxena  
(BITS Pilani Hyderabad Campus)

# MC without exploring starts: On-policy vs Off-policy methods

- On-policy: Choose a “soft-policy”, i.e,  $\pi(a|s) > 0$  – meaning each action has a non-zero probability to be chosen. Gradually shift towards a deterministic policy.
- Use  $\epsilon$  – *greedy* approach, meaning most of the time choose an action with maximum estimated value and the other times choose a random action with probability  $\epsilon$ .
- Non-greedy actions are selected with probability  $\frac{\epsilon}{|A(s)|}$
- $\epsilon$  – *greedy* policies are among  $\epsilon$  – *soft* policies.

# On-Policy MC Method vs MC ES

innovate

achieve

lead

On-policy first-visit MC control (for  $\epsilon$ -soft policies), estimates  $\pi \approx \pi_*$ .

Algorithm parameter: small  $\epsilon > 0$

Initialize:

$\pi \leftarrow$  an arbitrary  $\epsilon$ -soft policy

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

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

Repeat forever (for each episode):

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

$G \leftarrow 0$

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

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

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

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

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

$A^* \leftarrow \arg\max_a Q(S_t, a)$  (with ties broken arbitrarily)

For all  $a \in \mathcal{A}(S_t)$ :

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon / |\mathcal{A}(S_t)| & \text{if } a = A^* \\ \epsilon / |\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

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

$G \leftarrow 0$

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

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

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

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

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

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

Moving towards  
 $\epsilon$  – greedy policy.

# MC without exploring starts: On-policy vs Off-policy methods

- On-policy: Choose a “soft-policy”, i.e,  $\pi(a|s) > 0$  – meaning each action has a non-zero probability to be chosen. Gradually shift towards a deterministic policy.
- Use  $\epsilon$  – *greedy* approach, meaning most of the time choose an action with maximum estimated value and the other times choose a random action with probability  $\epsilon$ .
- Non-greedy actions are selected with probability  $\frac{\epsilon}{|A(s)|}$
- $\epsilon$  – *greedy* policies are among  $\epsilon$  – *soft* policies.

# On-Policy MC Method vs MC ES

innovate

achieve

lead

On-policy first-visit MC control (for  $\epsilon$ -soft policies), estimates  $\pi \approx \pi_*$ .

Algorithm parameter: small  $\epsilon > 0$

Initialize:

$\pi \leftarrow$  an arbitrary  $\epsilon$ -soft policy

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

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

Repeat forever (for each episode):

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

$G \leftarrow 0$

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

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

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

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

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

$A^* \leftarrow \arg\max_a Q(S_t, a)$  (with ties broken arbitrarily)

For all  $a \in \mathcal{A}(S_t)$ :

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon / |\mathcal{A}(S_t)| & \text{if } a = A^* \\ \epsilon / |\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

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

$G \leftarrow 0$

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

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

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

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

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

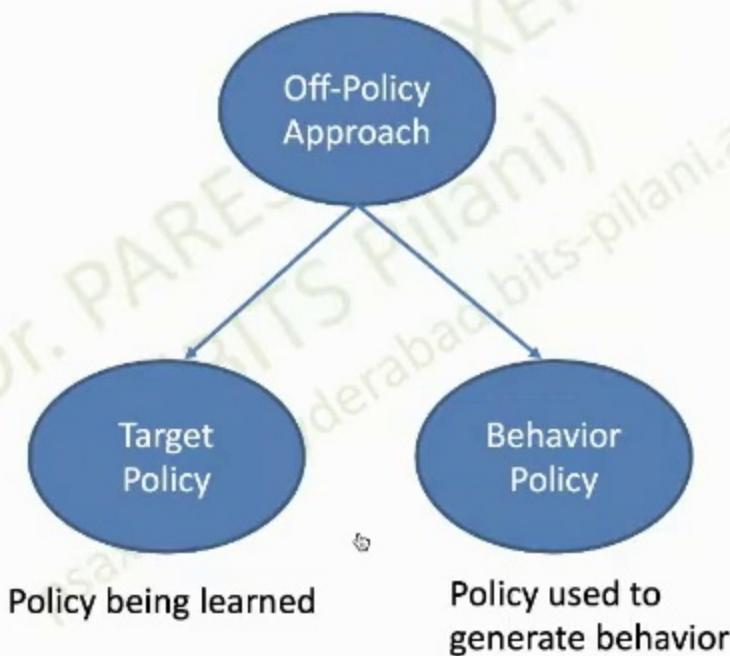
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Moving towards  
 $\epsilon$  - greedy policy.



# Monte Carlo: Off-Policy Prediction

- On-policy approaches generally learn action values for near-optimal policy.



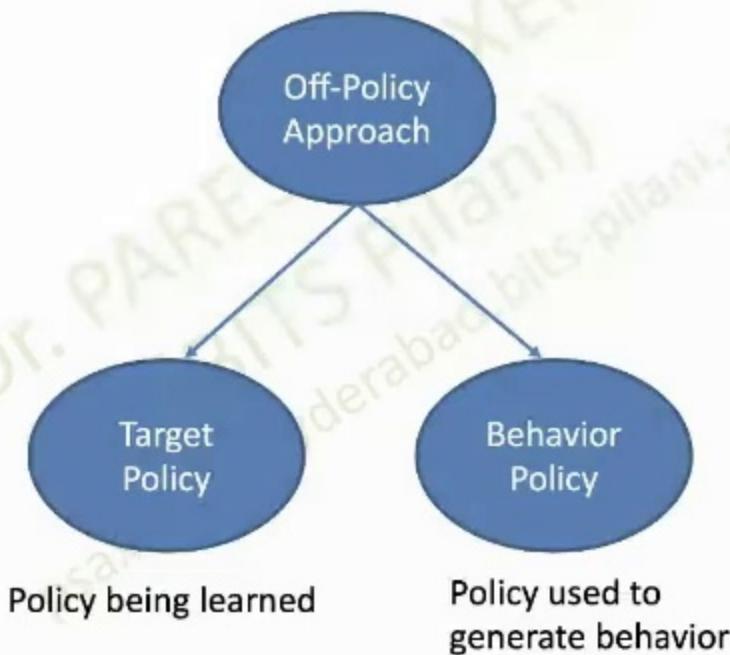
Learning is  
from data off  
the target  
policy !!

# MC without exploring starts: On-policy vs Off-policy methods

- Exploring start can not be generalized.
- On-policy methods: Improve the policy that is used to make decisions, e.g., Monte Carlo methods
- Off-policy methods: Improve the policy that is different than the policy that is used to generate the data.

# Monte Carlo: Off-Policy Prediction

- On-policy approaches generally learn action values for near-optimal policy.



Learning is from data off the target policy !!

# Monte Carlo: Off-Policy Prediction



- Off-policy methods
- Pros: Powerful, Generalized, have on-policy as special case, targeting for optimal solution
- Cons: Greater Variance and Slower to Converge

Dr. P. SAXENA  
(BITS Pilani)  
psaxena@hyderabad.bits-pilani.ac.in

# Off-policy learning

- Estimate  $q_\pi$  based on experience/episodes generated with behavioral policy  $b$  where  $b \neq \pi$ .
- Requires **coverage**, i.e., if  $\pi(a|s) > 0$  then  $b(a|s) > 0$ , i.e., every action under  $\pi$  is also taken under  $b$  at least occasionally.
- Behavioral policy must be stochastic – exploration !
- Target policy may be deterministic.
- Off-policy prediction problem:  $\pi$  is unchanging and given.

# Off-policy learning: Important Sampling Ratio

$$\begin{aligned} \Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\ = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\ = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k), \end{aligned}$$

Probability of  
state-action  
trajectory

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

I

Importance Sampling ratio – Independent of state-transitions which are generally unknown

# Why Importance Sampling Ratio ?



- We have: returns  $G_t$  due to behavioral policy (since episodes are being generated by behavioral policy)
- We want: returns from target policy:

$$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s).$$

- Estimate value function of target policy by observing episodes and returns of behavioral policy using importance sampling ratio !!!

# Ordinary and weighted Importance Sampling

**Ordinary IS (unbiased but unbounded variance) and estimating  $V(s)$  for MC:**

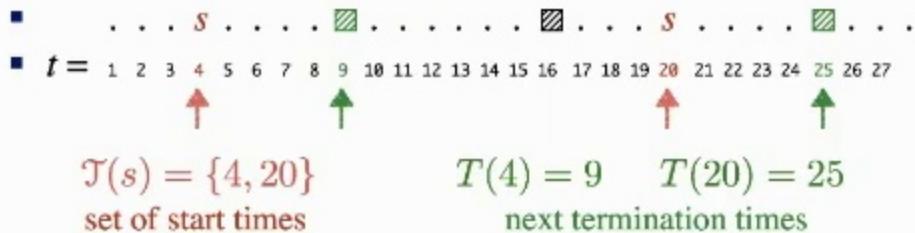
$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}.$$

Generalized for first-visit and every-visit.

**Weighted IS (biased but variance converges to 0) and estimating V(s) for MC:**

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}},$$

- $\rho_{t:T(t)-1}$  - importance sampling ratio
  - $\Gamma(s)$  – time stamps that were first visits to state  $s$  within their episodes.
  - $T(t)$  – first time of termination following time  $t$



# Why Importance Sampling Ratio ?



- We have: returns  $G_t$  due to behavioral policy (since episodes are being generated by behavioral policy)
- We want: returns from target policy:

$$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s).$$

- Estimate value function of target policy by observing episodes and returns of behavioral policy using importance sampling ratio !!!

# Off-policy learning: Important Sampling Ratio

$$\begin{aligned} \Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\ = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\ = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k), \end{aligned}$$

Probability of  
state-action  
trajectory

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

I

Importance Sampling ratio – Independent of state-transitions which are generally unknown

# Why Importance Sampling Ratio ?



- We have: returns  $G_t$  due to behavioral policy (since episodes are being generated by behavioral policy)
- We want: returns from target policy:

$$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s).$$

- Estimate value function of target policy by observing episodes and returns of behavioral policy using importance sampling ratio !!!

# Ordinary and weighted Importance Sampling

**Ordinary IS (unbiased but unbounded variance) and estimating  $V(s)$  for MC:**

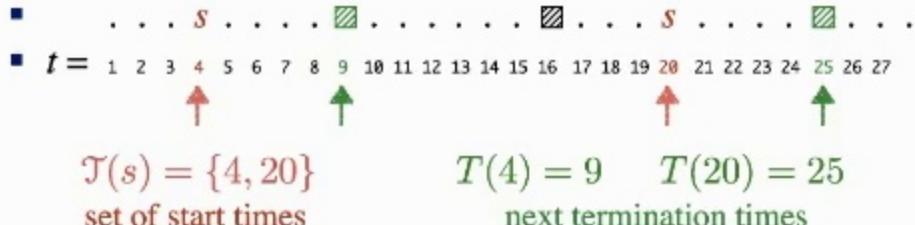
$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

Generalized for first-visit and every-visit.

**Weighted IS (biased but variance converges to 0) and estimating  $V(s)$  for MC:**

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

- $\rho_{t:T(t)-1}$  - importance sampling ratio
- $\mathcal{T}(s)$  – time stamps that were first visits to state  $s$  within their episodes.
- $T(t)$  – first time of termination following time  $t$



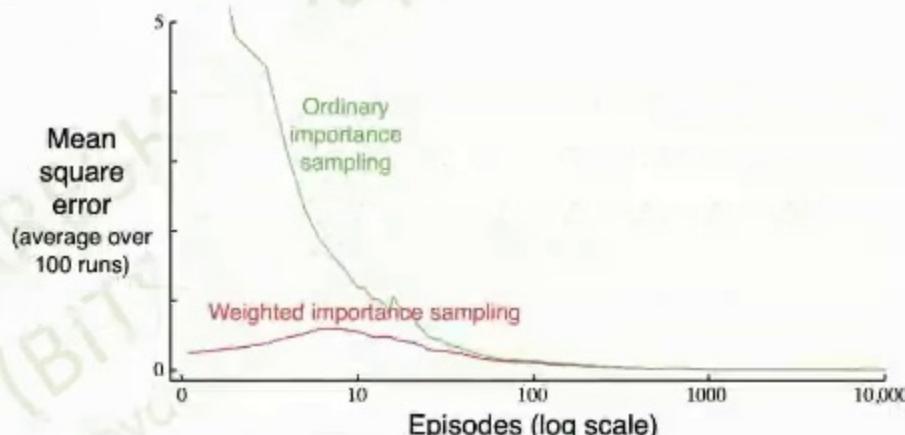
# Off-Policy Evaluation in BlackJack

innovate

achieve

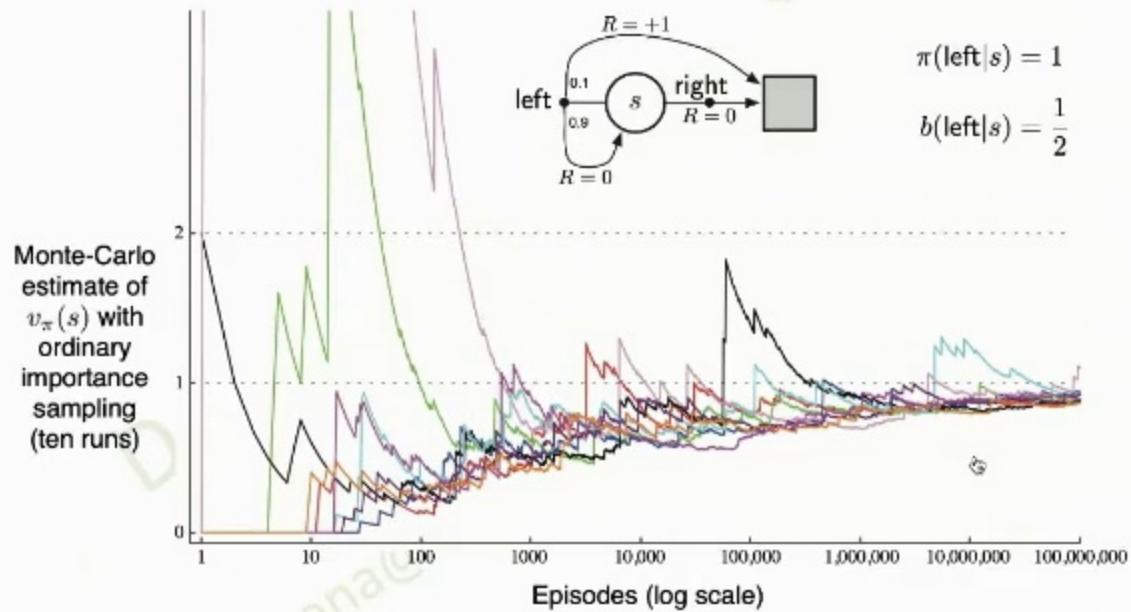
lead

- $\pi$  (target): stick if player sum is 20 or 21
- $b$  (behavioral): uniformly random
- $s$ : player sum 13 (Usable ace and 2), dealer shows 2
- $v_\pi(s) \approx -0.27726$



High variance for Ordinary IS in the beginning.

# Ordinary IS: Infinite Variance (Trajectory with loops)



A mathematical proof of variance to be infinite is given in the textbook.

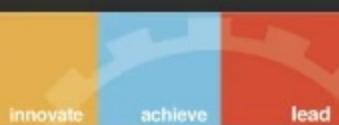


# Upcoming Project Deadline for Step 2 and Step 3



1. Send group details (**completed**)
  2. Pick a problem that interests you
  3. Identify 8-10 relevant research papers that you plan to review (8 for group with 4 members, 10 for group with 5 members). Each student will review two papers in next steps.
- Submit one page on (1), (2) and (3) to [p20190504@hyderabad.bits-pilani.ac.in](mailto:p20190504@hyderabad.bits-pilani.ac.in) by Friday, 8<sup>th</sup> Oct 2021, 17:00.
  - This activity carries 5% weightage of final grade.

# Upcoming Project Deadline for Step 2 and Step 3



1. Send group details (**completed**)
  2. Pick a problem that interests you
  3. Identify 8-10 relevant research papers that you plan to review (8 for group with 4 members, 10 for group with 5 members). Each student will review two papers in next steps.
- Submit one page on (1), (2) and (3) to [p20190504@hyderabad.bits-pilani.ac.in](mailto:p20190504@hyderabad.bits-pilani.ac.in) by Friday, 8<sup>th</sup> Oct 2021, 17:00.
  - This activity carries 5% weightage of final grade.

# Off-policy MC prediction

- Off-policy MC prediction: Average of Weighted Returns
- Let us consider  $G_1, G_2, G_3, \dots, G_{n-1}$  as a sequence of returns with a corresponding random weight  $W_i$  ( $W_i = \rho_{t_i:T(t_i)-1}$ ). To update:

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2,$$

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1,$$

I

and,

$$C_{n+1} \doteq C_n + W_{n+1},$$

# Iterative Implementation of Weighted IS

innovate

achieve

lead

Off-policy MC prediction (policy evaluation) for estimating  $Q \approx q_\pi$

Input: an arbitrary target policy  $\pi$

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$Q(s, a) \in \mathbb{R}$  (arbitrarily)

$C(s, a) \leftarrow 0$

Loop forever (for each episode):

$b \leftarrow$  any policy with coverage of  $\pi$

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

$G \leftarrow 0$

$W \leftarrow 1$

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

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

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$W \leftarrow W \frac{\pi(A_t | S_t)}{b(A_t | S_t)}$

← cumulative sum

I

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2,$$

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1,$$

$$C_{n+1} \doteq C_n + W_{n+1},$$

# Off-policy MC prediction

- Off-policy MC prediction: Average of Weighted Returns
- Let us consider  $G_1, G_2, G_3, \dots, G_{n-1}$  as a sequence of returns with a corresponding random weight  $W_i$  ( $W_i = \rho_{t_i:T(t_i)-1}$ ). To update:

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2,$$

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1,$$

and,

$$C_{n+1} \doteq C_n + W_{n+1},$$

# Iterative Implementation of Weighted IS

innovate

achieve

lead

Off-policy MC prediction (policy evaluation) for estimating  $Q \approx q_\pi$

Input: an arbitrary target policy  $\pi$

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$$Q(s, a) \in \mathbb{R} \text{ (arbitrarily)}$$

$$C(s, a) \leftarrow 0$$

Loop forever (for each episode):

$b \leftarrow$  any policy with coverage of  $\pi$

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

$$G \leftarrow 0$$

$$W \leftarrow 1$$

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

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

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$W \leftarrow W \frac{\pi(A_t | S_t)}{b(A_t | S_t)}$$

← cumulative sum

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2,$$

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1,$$

$$C_{n+1} \doteq C_n + W_{n+1},$$

# Off-policy MC prediction

- Off-policy MC prediction: Average of Weighted Returns
- Let us consider  $G_1, G_2, G_3, \dots, G_{n-1}$  as a sequence of returns with a corresponding random weight  $W_i$  ( $W_i = \rho_{t_i:T(t_i)-1}$ ). To update:

$$\bar{V}_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2,$$

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1,$$

and,

$$C_{n+1} \doteq C_n + W_{n+1},$$

# Iterative Implementation of Weighted IS

innovate

achieve

lead

Off-policy MC prediction (policy evaluation) for estimating  $Q \approx q_\pi$

Input: an arbitrary target policy  $\pi$

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$$Q(s, a) \in \mathbb{R} \text{ (arbitrarily)}$$

$$C(s, a) \leftarrow 0$$

Loop forever (for each episode):

$b \leftarrow$  any policy with coverage of  $\pi$

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

$$G \leftarrow 0$$

$$W \leftarrow 1$$

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

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

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$W \leftarrow W \frac{\pi(A_t | S_t)}{b(A_t | S_t)}$$

← cumulative sum

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2,$$

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1,$$

$$C_{n+1} \doteq C_n + W_{n+1},$$

I

# Off-Policy Monte Carlo Control

Off-policy MC control, for estimating  $\pi \approx \pi_*$

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$Q(s, a) \in \mathbb{R}$  (arbitrarily)

$C(s, a) \leftarrow 0$

$\pi(s) \leftarrow \arg \max_a Q(s, a)$  (with ties broken consistently)

Loop forever (for each episode):

$b \leftarrow$  any soft policy

Generate an episode using  $b$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

$W \leftarrow 1$

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

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

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a)$  (with ties broken consistently)

If  $A_t \neq \pi(S_t)$  then exit inner Loop (proceed to next episode)

$W \leftarrow W \frac{1}{b(A_t | S_t)}$

Target policy can be greedy and behavior policy (**soft**) can explore.

To explore

Greedy Policy

# Off-policy MC prediction

- Off-policy MC prediction: Average of Weighted Returns
- Let us consider  $G_1, G_2, G_3, \dots, G_{n-1}$  as a sequence of returns with a corresponding random weight  $W_i$  ( $W_i = \rho_{t_i:T(t_i)-1}$ ). To update:

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2,$$

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1,$$

and,

$$C_{n+1} \doteq C_n + W_{n+1},$$

# Iterative Implementation of Weighted IS

innovate

achieve

lead

Off-policy MC prediction (policy evaluation) for estimating  $Q \approx q_\pi$

Input: an arbitrary target policy  $\pi$

Initialize, for all  $s \in S, a \in A(s)$ :

$$Q(s, a) \in \mathbb{R} \text{ (arbitrarily)}$$

$$C(s, a) \leftarrow 0$$

Loop forever (for each episode):

$b \leftarrow$  any policy with coverage of  $\pi$

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

$$G \leftarrow 0$$

$$W \leftarrow 1$$

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

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

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$W \leftarrow W \frac{\pi(A_t | S_t)}{b(A_t | S_t)}$$

← cumulative sum

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2,$$

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1,$$

$$C_{n+1} \doteq C_n + W_{n+1},$$

# Off-Policy Monte Carlo Control

Off-policy MC control, for estimating  $\pi \approx \pi_*$ .

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$Q(s, a) \in \mathbb{R}$  (arbitrarily)

$C(s, a) \leftarrow 0$

$\pi(s) \leftarrow \arg \max_a Q(s, a)$  (with ties broken consistently)

Loop forever (for each episode):

$b \leftarrow$  any soft policy

Generate an episode using  $b$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

$W \leftarrow 1$

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

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

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a)$  (with ties broken consistently)

If  $A_t \neq \pi(S_t)$  then exit inner Loop (proceed to next episode)

$W \leftarrow W \frac{1}{b(A_t | S_t)}$

Target policy can be greedy and behavior policy (**soft**) can explore.

To explore

Greedy Policy

# Off-Policy Monte Carlo Control

Off-policy MC control, for estimating  $\pi \approx \pi_*$

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$Q(s, a) \in \mathbb{R}$  (arbitrarily)

$C(s, a) \leftarrow 0$

$\pi(s) \leftarrow \arg \max_a Q(s, a)$  (with ties broken consistently)

Loop forever (for each episode):

$b \leftarrow$  any soft policy

Generate an episode using  $b$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

$W \leftarrow 1$

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

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

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \arg \max_a Q(S_t, a)$  (with ties broken consistently)

If  $A_t \neq \pi(S_t)$  then exit inner Loop (proceed to next episode)

$W \leftarrow W \frac{1}{b(A_t | S_t)}$

Target policy can be greedy and behavior policy (**soft**) can explore.

To explore

Greedy Policy

# Monte Carlo: Problem Session

Problem:

- Thief (each day decides whether to steal or retire)
- If thief steals, there is some probability of him to get caught and some probability of not getting caught
- If thief steals, he steals either 1 million dollars, or 2 million dollars or 4 million dollars
- If thief gets caught, he has to pay a fine of 10 million dollars and he will no longer steal
- Discount factor  $\gamma = 0.8$
- Q. 1 Define the states, actions and rewards for the problem.



# Monte Carlo: Problem Session

Problem:

- Thief (each day decides whether to steal or retire)
- If thief steals, there is some probability of him to get caught and some probability of not getting caught
- If thief steals, he steals either 1 million dollars, or 2 million dollars or 4 million dollars
- If thief gets caught, he has to pay a fine of 10 million dollars and he will no longer steal
- Discount factor  $\gamma = 0.8$
- Q. 1 Define the states, actions and rewards for the problem.



# Monte Carlo: Problem Session

Problem:

- Thief (each day decides whether to steal or retire)
- If thief steals, there is some probability of him to get caught and some probability of not getting caught
- If thief steals, he steals either 1 million dollars, or 2 million dollars or 4 million dollars
- If thief gets caught, he has to pay a fine of 10 million dollars and he will no longer steal
- Discount factor  $\gamma = 0.8$
- Q. 1 Define the states, actions and rewards for the problem.

# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}

# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S,A)$

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

**Initialization:**

$\pi(s) \in \mathcal{A}(s)$  (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  $\pi$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$   
 $G \leftarrow 0$

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

$G \leftarrow \gamma G + R_{t+1}$   
Unless the pair  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ :  
Append  $G$  to  $Returns(S_t, A_t)$   
 $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$   
 $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$

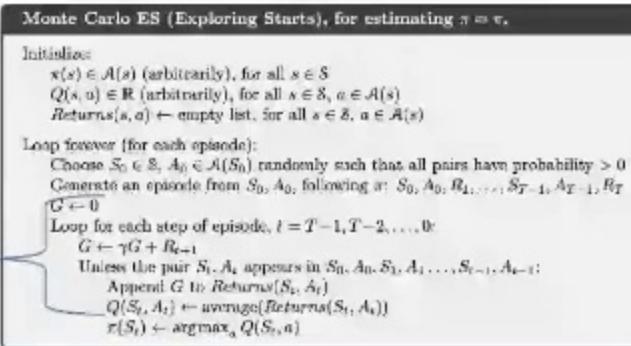


# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$



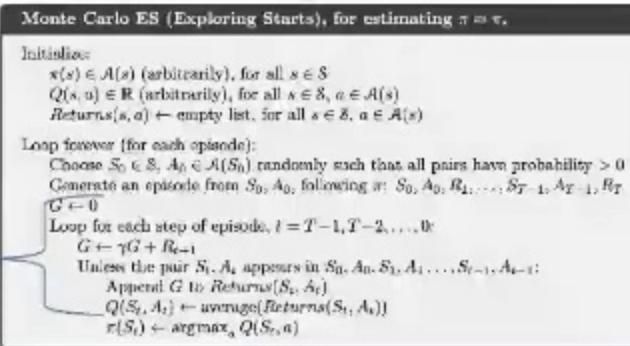


# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$





# Monte Carlo: Problem Session

innovate

achieve

lead

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$

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

Initialize:  
 $v(s) \in A(s)$  (arbitrarily), for all  $s \in S$   
 $Q(s, a) \in \mathbb{R}$  (arbitrarily), for all  $s \in S, a \in A(s)$   
 $Returns(s, a) \leftarrow$  empty list, for all  $s \in S, a \in A(s)$

Loop forever (for each episode):  
Choose  $S_0 \in S, A_0 \in A(S_0)$  randomly such that all pairs have probability  $> 0$   
Generate an episode from  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$   
 $G \leftarrow 0$

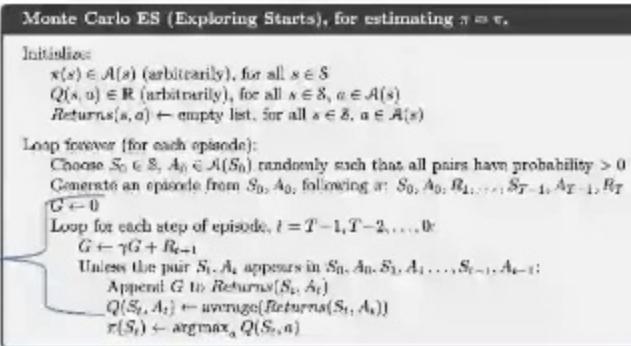
Loop for each step of episode,  $t = T-1, T-2, \dots, 0$ :  
 $G \leftarrow \gamma G + R_{t+1}$   
Unless the pair  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ :  
 $Append G to Returns(S_t, A_t)$   
 $Q(S_t, A_t) \leftarrow \text{avg}(Returns(S_t, A_t))$   
 $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$

# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$



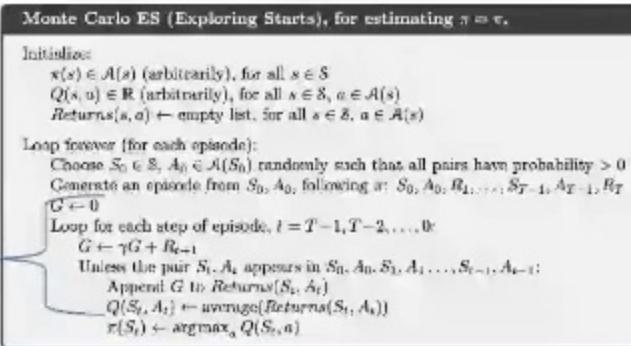


# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$





# Monte Carlo: Problem Session

innovate

achieve

lead

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$

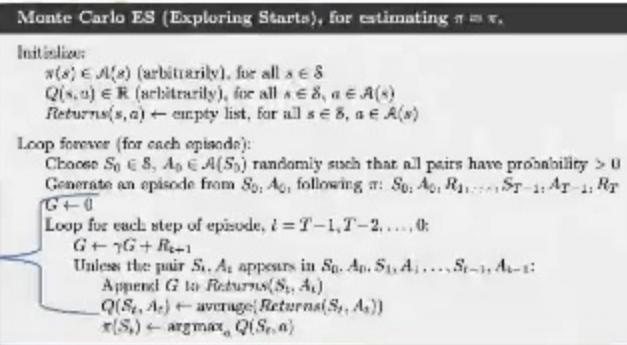
```
Monte Carlo ES (Exploring Starts), for estimating  $\pi = \pi_*$ .  
Initialize:  
     $\pi(s) \in \mathcal{A}(s)$  (arbitrarily), for all  $s \in \mathcal{S}$   
     $Q(s, a) \in \mathbb{R}$  (arbitrarily), for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$   
     $Returns[s, a] \leftarrow$  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  $\pi$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$   
     $G \leftarrow 0$   
    Loop for each step of episode,  $t = T-1, T-2, \dots, 0$ :  
         $G \leftarrow \gamma G + R_{t+1}$   
        Unless the pair  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ :  
            Append  $G$  to  $Returns[S_t, A_t]$   
             $Q(S_t, A_t) \leftarrow \text{average}(Returns[S_t, A_t])$   
             $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$ 
```

# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$

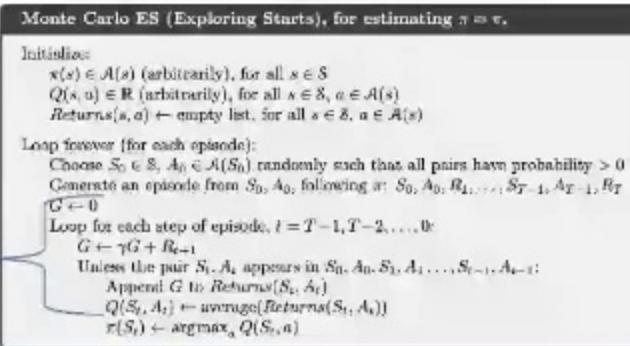


# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$



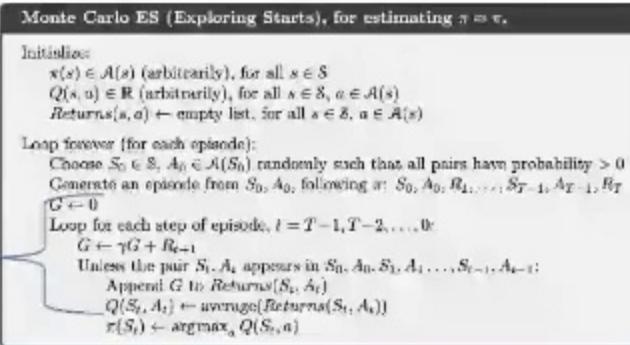


# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$





# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$

```

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  $\pi$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ 
   $G \leftarrow 0$ 
  Loop for each step of episode,  $t = T-1, T-2, \dots, 0$ :
     $G \leftarrow \gamma G + R_{t+1}$ 
    Unless the pair  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ :
      Append  $G$  to  $Returns[S_t, A_t]$ 
       $Q(S_t, A_t) \leftarrow \text{average}(Returns[S_t, A_t])$ 
       $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$ 
  
```

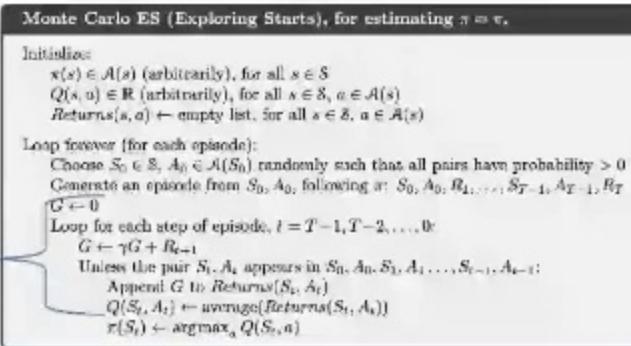


# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

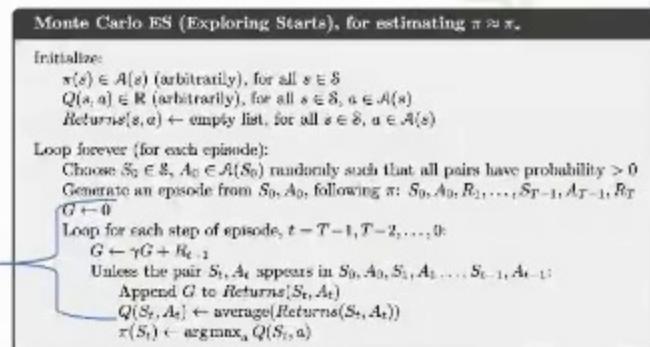
Estimate  
 $Q(S, A)$



# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T

Estimate  
 $Q(S, A)$



- $G = 0$
- $G = -10$
- $G = 2 + 0.8 * (-10) = -6$
- $G = 1 + 0.8 * (-6) = -3.8$
- $G = 4 + 0.8 * (-3.8) = 0.96$
- $G = 2 + 0.8 * (0.96) = 2.768$

After 1<sup>st</sup> episode  $Q(S|D) = 2.768$

# Monte Carlo: Problem Session



- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

After 6 episodes:

$$Q(S|D) = [2.768 + (-7) + 0 + (-1.2) + 4.2 + 5.55]/6 = 0.72$$

$$Q(R|D) = 0$$

# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 4( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T

Estimate  
 $Q(S, A)$

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  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

Uniqe the pair  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ :

Append  $G$  to  $Returns[S_t, A_t]$

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

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

- $G = 0$
- $G = -10$
- $G = 4 + 0.8 * (-10) = -4$
- $G = 2 + 0.8 * (-4) = -1.2$

After 4<sup>th</sup> episode  $Q(S|D) = -1.2$

# Monte Carlo: Problem Session

innovate

achieve

lead

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .



After 6 episodes:

$$Q(S|D) = [2.768 + (-7) + 0 + (-1.2) + 4.2 + 5.55]/6 = 0.72$$

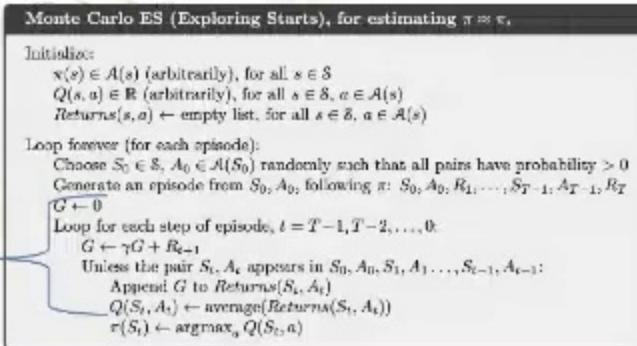
$$Q(R|D) = 0$$

# Monte Carlo: Problem Session

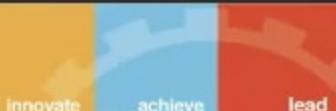
- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S, A)$



# Monte Carlo: Problem Session



- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

After 6 episodes:

$$Q(S|D) = [2.768 + (-7) + 0 + (-1.2) + 4.2 + 5.55]/6 = 0.72$$

$$Q(R|D) = 0$$

RL class - p20190504@hyderabad

Meet - brc-ssbd-pzj

cym/blackjack.py at master · ops/Blackjack using On Policy Monte Carlo - Collaboratory colab.research.google.com

You're presenting to everyone

Blackjack using On Policy Monte Carlo - Collaboratory colab.research.google.com

YouTube Maps

You're presenting your screen

Stop presenting Ignore

infinity mirror, don't share your entire screen or browser window. Share different window instead.

meet.google.com is sharing your screen. Stop sharing Hide

Google Chrome

You're presenting to Click here to return to when you're ready to meet.google.com

paresh saxena ANJEL PATEL

S Saloni Singh NEIL PARESH MEH...

RUBAN S 10 others

30°C Rain showers

policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-sbpd-pj x | gym/blackjack.py at master · open x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1clJcpS6D6Dw2C9dYdbI8d3lkXWxD\_Vr6#scrollTo=XlY8Z1lvNjM7

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help All changes saved

ext Connect ▾

Blackjack Gym: [https://github.com/openai/gym/blob/master/gym/envs/toy\\_text/blackjack.py](https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py)

### first-visit MC control algorithm for epsilon-soft policies

, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$\pi(s, a) \leftarrow$  arbitrary

$ns(s, a) \leftarrow$  empty list

an arbitrary  $\varepsilon$ -soft policy

forever:

generate an episode using  $\pi$

for each pair  $s, a$  appearing in the episode:

$R \leftarrow$  return following the first occurrence of  $s, a$

Append  $R$  to  $Returns(s, a)$

$Q(s, a) \leftarrow$  average( $Returns(s, a)$ )

for each  $s$  in the episode:

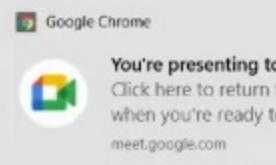
$a^* \leftarrow \arg \max_a Q(s, a)$

For all  $a \in \mathcal{A}(s)$ :

$$\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

### policy and state value function for blackjack

meet.google.com is sharing your screen. Stop sharing Hide



RL class - p20190504@hyderabad | Meet - brc-ssbd-pzj | gym/blackjack.py at master · op... | Blackjack using On Policy Monte | +

meet.google.com/brc-ssbd-pzj?authuser=0&pli=1

You're presenting to everyone

The screenshot shows a Google Meet interface. On the left, a presentation slide is displayed with a dark background and various text elements. In the center, a grid of participant thumbnails is shown. The participants listed are:

- paresh saxena
- ANJEL PATEL
- Saloni Singh
- NEIL PARESH MEH...
- RUBAN S
- 9 others

At the bottom of the screen, there is a message from Google Meet stating: "Inity mirror, don't share your entire screen or browser window. Share different window instead." Below this message are buttons for "Stop presenting", "Ignore", and "Hide".

At the very bottom of the image, a taskbar is visible with icons for search, file, mail, calendar, and other applications. A weather widget indicates "30°C Rain showers".

policy Monte Carlo x RL class - p20190504@hyderabad x Meet - brc-ssbd-pj x gym/blackjack.py at master · op... x Blackjack using On Policy Monte Carlo x +

lab.research.google.com/drive/1ciJcpS6D6Dw2C9dYdbI8d3kXWxD\_Vr6#scrollTo=XlYZ1lvNjM7

YouTube Maps

Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help All changes saved

ext

Comment Connect

blackjack Gym: [https://github.com/openai/gym/blob/master/gym/envs/toy\\_text/blackjack.py](https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py)

first-visit MC control algorithm for epsilon-soft policies

, for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ :

$\pi(s, a) \leftarrow$  arbitrary

$ns(s, a) \leftarrow$  empty list

an arbitrary  $\varepsilon$ -soft policy

forever:

generate an episode using  $\pi$

for each pair  $s, a$  appearing in the episode:

$R \leftarrow$  return following the first occurrence of  $s, a$

Append  $R$  to  $Returns(s, a)$

$Q(s, a) \leftarrow$  average( $Returns(s, a)$ )

for each  $s$  in the episode:

$a^* \leftarrow \arg \max_a Q(s, a)$

For all  $a \in \mathcal{A}(s)$ :

$$\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

policy and state value function for blackjack

oole.com...

to search

Loading...

x

Google Chrome



You're presenting to  
Click here to return to  
when you're ready to  
meet.google.com

meet.google.com is sharing your screen. Stop sharing Hide

30°C Rain showers

By 15 contributors

128 lines (95 sloc) | 4.33 KB

[Raw](#) [Blame](#) [Copy](#) [Edit](#) [Delete](#)

```
1 import gym
2 from gym import spaces
3 from gym.utils import seeding
4
5
6 def cmp(a, b):
7     return float(a > b) - float(a < b)
8
9
10 # 1 = Ace, 2-10 = Number, cards, Jack/Queen/King = 10
11 deck = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]
12
13
14 def draw_card(np_random):
15     return int(np_random.choice(deck))
16
17
18 def draw_hand(np_random):
19     return [draw_card(np_random), draw_card(np_random)]
20
21
22 def usable_ace(hand): # Does this hand have a usable ace?
23     return 1 in hand and sum(hand) + 10 <= 21
24
25
26 def sum_hand(hand): # Return current hand total
27     if usable_ace(hand):
28         return sum(hand) + 10
29     return sum(hand)
```

meet.google.com is sharing your screen. [Stop sharing](#) [Hide](#)

```
41     return sorted(hand) == [1, 10]
42
43
44 class BlackjackEnv(gym.Env):
45     """Simple blackjack environment
46
47     Blackjack is a card game where the goal is to obtain cards that sum to as
48     near as possible to 21 without going over. They're playing against a fixed
49     dealer.
50
51     Face cards (Jack, Queen, King) have point value 10.
52     Aces can either count as 11 or 1, and it's called 'usable' at 11.
53     This game is played with an infinite deck (or with replacement).
54     The game starts with dealer having one face up and one face down card, while
55     player having two face up cards. (Virtually for all Blackjack games today).
56
57     The player can request additional cards (hit=1) until they decide to stop
58     (stick=0) or exceed 21 (bust).
59
60     After the player sticks, the dealer reveals their facedown card, and draws
61     until their sum is 17 or greater. If the dealer goes bust the player wins.
62
63     If neither player nor dealer busts, the outcome (win, lose, draw) is
64     decided by whose sum is closer to 21. The reward for winning is +1,
65     drawing is 0, and losing is -1.
66
67     The observation of a 3-tuple of: the players current sum,
68     the dealers one showing card (1-10 where 1 is ace),
69     and whether or not the player holds a usable ace (0 or 1).
70
71     This environment corresponds to the version of the blackjack problem
72     described in Example 5.1 in Reinforcement Learning: An Introduction
73     by Sutton and Barto.
74     http://incompleteideas.net/book/the-book-2nd.html
75
76     def __init__(self, natural=False, sab=False):
```

meet.google.com is sharing your screen. Stop sharing Hide



YouTube Maps

```
99         done = True
100        reward = -1.0
101    else:
102        done = False
103        reward = 0.0
104    else: # stick: play out the dealers hand, and score
105        done = True
106        while sum_hand(self.dealer) < 17:
107            self.dealer.append(draw_card(self.np_random))
108        reward = cmp(score(self.player), score(self.dealer))
109        if self.sab and is_natural(self.player) and not is_natural(self.dealer):
110            # Player automatically wins. Rules consistent with S&B
111            reward = 1.0
112        elif (
113            not self.sab
114            and self.natural
115            and is_natural(self.player)
116            and reward == 1.0
117        ):
118            # Natural gives extra points, but doesn't autowin. Legacy implementation
119            reward = 1.5
120        return self._get_obs(), reward, done, {}
121
122    def _get_obs(self):
123        return (sum_hand(self.player), self.dealer[0], usuable_ace(self.player))
124
125    def reset(self):
126        self.dealer = draw_hand(self.np_random)
127        self.player = draw_hand(self.np_random)
128        return self._get_obs()
```



Policy Monte Carlo - RL class - p20190504@hyderabad | Meet - brc-sabd-pj | gym/blackjack.py at master · opencog/gym | Blackjack Using On Policy Monte Carlo

YouTube Maps

Blackjack Using On Policy Monte Carlo

Comment Connect

## Blackjack Using On Policy Monte Carlo

View Insert Runtime Tools Help All changes saved

ext

Blackjack Gym: [https://github.com/openai/gym/blob/master/gym/envs/toy\\_text/blackjack.py](https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py)

### First-visit MC control algorithm for epsilon-soft policies

, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$\pi(s)$   $\leftarrow$  arbitrary

$Returns(s, a)$   $\leftarrow$  empty list

an arbitrary  $\varepsilon$ -soft policy

forever:

generate an episode using  $\pi$

for each pair  $s, a$  appearing in the episode:

$R \leftarrow$  return following the first occurrence of  $s, a$

Append  $R$  to  $Returns(s, a)$

$Q(s, a) \leftarrow$  average( $Returns(s, a)$ )

for each  $s$  in the episode:

$a^* \leftarrow \arg \max_a Q(s, a)$

For all  $a \in \mathcal{A}(s)$ :

$$\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon / |\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon / |\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

### Optimal policy and state value function for blackjack

meet.google.com is sharing your screen.

Stop sharing

Hide

to search



30°C Rain showers

RL class - p20190504@hyderabad

Meet - brc-ssbd-pzj

cym/blackjack.py at master · ops/Blackjack using On Policy Monte Carlo - Collaboratory colab.research.google.com

You're presenting to everyone

Blackjack using On Policy Monte Carlo - Collaboratory colab.research.google.com

YouTube Maps

You're presenting your screen

Stop presenting Ignore

infinity mirror, don't share your entire screen or browser window. Share different window instead.

meet.google.com is sharing your screen. Stop sharing Hide

Google Chrome

You're presenting to Click here to return to when you're ready to meet.google.com

paresh saxena ANJEL PATEL

S Saloni Singh NEIL PARESH MEH...

RUBAN S 10 others

30°C Rain showers

policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-sbpd-pj x | gym/blackjack.py at master · open x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1clJcpS6D6Dw2C9dYdbI8d3lkXWxD\_Vr6#scrollTo=XlY8Z1lvNjM7

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help All changes saved

ext Connect ▾

blackjack Gym: [https://github.com/openai/gym/blob/master/gym/envs/toy\\_text/blackjack.py](https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py)

### first-visit MC control algorithm for epsilon-soft policies

, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$\pi(s, a) \leftarrow$  arbitrary

$ns(s, a) \leftarrow$  empty list

an arbitrary  $\varepsilon$ -soft policy

forever:

generate an episode using  $\pi$

for each pair  $s, a$  appearing in the episode:

$R \leftarrow$  return following the first occurrence of  $s, a$

Append  $R$  to  $Returns(s, a)$

$Q(s, a) \leftarrow$  average( $Returns(s, a)$ )

for each  $s$  in the episode:

$a^* \leftarrow \arg \max_a Q(s, a)$

For all  $a \in \mathcal{A}(s)$ :

$$\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

### policy and state value function for blackjack

meet.google.com is sharing your screen.

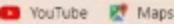
Stop sharing

Hide

Google Chrome

You're presenting to  
Click here to return to  
when you're ready to  
meet.google.com

meet.google.com/birc-ssbd-pzj?authuser=0&pli=1



You're presenting to everyone

A screenshot of a video conferencing interface, likely from a platform like Zoom or Google Meet. The main area shows a grid of participant thumbnails and names. On the far left, there's a message about screen sharing. At the bottom, a prominent blue button says "Stop presenting".

Inity mirror, don't share your entire screen or browser window. Share different window instead.

## Stop presenting

ignore

asb:d-pzi

meet.google.com is sharing your screen. [Stop sharing](#) [Hide](#)

policy Monte Carlo x RL class - p20190504@hyderabad x Meet - brc-ssbd-pj x gym/blackjack.py at master · op... x Blackjack using On Policy Monte Carlo x +

lab.research.google.com/drive/1ciJcpS6D6Dw2C9dYdbI8d3kXWxD\_Vr6#scrollTo=XlYZ1lvNjM7

YouTube Maps

Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help All changes saved

ext

Comment Connect

blackjack Gym: [https://github.com/openai/gym/blob/master/gym/envs/toy\\_text/blackjack.py](https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py)

first-visit MC control algorithm for epsilon-soft policies

, for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ :

$\pi(s, a) \leftarrow$  arbitrary

$ns(s, a) \leftarrow$  empty list

an arbitrary  $\varepsilon$ -soft policy

forever:

generate an episode using  $\pi$

for each pair  $s, a$  appearing in the episode:

$R \leftarrow$  return following the first occurrence of  $s, a$

Append  $R$  to  $Returns(s, a)$

$Q(s, a) \leftarrow$  average( $Returns(s, a)$ )

for each  $s$  in the episode:

$a^* \leftarrow \arg \max_a Q(s, a)$

For all  $a \in \mathcal{A}(s)$ :

$$\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

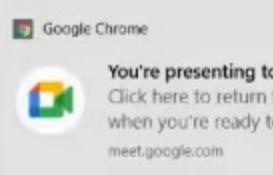
policy and state value function for blackjack

oole.com...

to search

Loading...

x



meet.google.com is sharing your screen. Stop sharing Hide

30°C Rain showers

By 15 contributors

128 lines (95 sloc) | 4.33 KB

Raw Blame   

```
1 import gym
2 from gym import spaces
3 from gym.utils import seeding
4
5
6 def cmp(a, b):
7     return float(a > b) - float(a < b)
8
9
10 # 1 = Ace, 2-10 = Number, cards, Jack/Queen/King = 10
11 deck = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]
12
13
14 def draw_card(np_random):
15     return int(np_random.choice(deck))
16
17
18 def draw_hand(np_random):
19     return [draw_card(np_random), draw_card(np_random)]
20
21
22 def usable_ace(hand): # Does this hand have a usable ace?
23     return 1 in hand and sum(hand) + 10 <= 21
24
25
26 def sum_hand(hand): # Return current hand total
27     if usable_ace(hand):
28         return sum(hand) + 10
29     return sum(hand)
```

meet.google.com is sharing your screen. Stop sharing Hide

```
41     return sorted(hand) == [1, 10]
42
43
44 class BlackjackEnv(gym.Env):
45     """Simple blackjack environment
46
47     Blackjack is a card game where the goal is to obtain cards that sum to as
48     near as possible to 21 without going over. They're playing against a fixed
49     dealer.
50
51     Face cards (Jack, Queen, King) have point value 10.
52     Aces can either count as 11 or 1, and it's called 'usable' at 11.
53     This game is played with an infinite deck (or with replacement).
54     The game starts with dealer having one face up and one face down card, while
55     player having two face up cards. (Virtually for all Blackjack games today).
56
57     The player can request additional cards (hit=1) until they decide to stop
58     (stick=0) or exceed 21 (bust).
59
60     After the player sticks, the dealer reveals their facedown card, and draws
61     until their sum is 17 or greater. If the dealer goes bust the player wins.
62
63     If neither player nor dealer busts, the outcome (win, lose, draw) is
64     decided by whose sum is closer to 21. The reward for winning is +1,
65     drawing is 0, and losing is -1.
66
67     The observation of a 3-tuple of: the players current sum,
68     the dealers one showing card (1-10 where 1 is ace),
69     and whether or not the player holds a usable ace (0 or 1).
70
71     This environment corresponds to the version of the blackjack problem
72     described in Example 5.1 in Reinforcement Learning: An Introduction
73     by Sutton and Barto.
74     http://incompleteideas.net/book/the-book-2nd.html
75
76     def __init__(self, natural=False, sab=False):
```

meet.google.com is sharing your screen. Stop sharing Hide



YouTube Maps

```
99         done = True
100        reward = -1.0
101    else:
102        done = False
103        reward = 0.0
104    else: # stick: play out the dealers hand, and score
105        done = True
106        while sum_hand(self.dealer) < 17:
107            self.dealer.append(draw_card(self.np_random))
108        reward = cmp(score(self.player), score(self.dealer))
109        if self.sab and is_natural(self.player) and not is_natural(self.dealer):
110            # Player automatically wins. Rules consistent with S&B
111            reward = 1.0
112        elif (
113            not self.sab
114            and self.natural
115            and is_natural(self.player)
116            and reward == 1.0
117        ):
118            # Natural gives extra points, but doesn't autowin. Legacy implementation
119            reward = 1.5
120        return self._get_obs(), reward, done, {}
121
122    def _get_obs(self):
123        return (sum_hand(self.player), self.dealer[0], usuable_ace(self.player))
124
125    def reset(self):
126        self.dealer = draw_hand(self.np_random)
127        self.player = draw_hand(self.np_random)
128        return self._get_obs()
```



Policy Monte Carlo - RL class - p20190504@hyderabad | Meet - brc-sabd-pj | gym/blackjack.py at master · opencog/gym | Blackjack Using On Policy Monte Carlo

YouTube Maps

Blackjack Using On Policy Monte Carlo

Comment Connect

## Blackjack Using On Policy Monte Carlo

View Insert Runtime Tools Help All changes saved

ext

Blackjack Gym: [https://github.com/openai/gym/blob/master/gym/envs/toy\\_text/blackjack.py](https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py)

### First-visit MC control algorithm for epsilon-soft policies

, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$\pi(s)$   $\leftarrow$  arbitrary

$Returns(s, a)$   $\leftarrow$  empty list

an arbitrary  $\varepsilon$ -soft policy

forever:

generate an episode using  $\pi$

for each pair  $s, a$  appearing in the episode:

$R \leftarrow$  return following the first occurrence of  $s, a$

Append  $R$  to  $Returns(s, a)$

$Q(s, a) \leftarrow$  average( $Returns(s, a)$ )

for each  $s$  in the episode:

$a^* \leftarrow \arg \max_a Q(s, a)$

For all  $a \in \mathcal{A}(s)$ :

$$\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon / |\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon / |\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

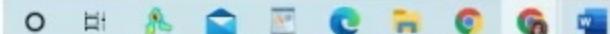
### Optimal policy and state value function for blackjack

meet.google.com is sharing your screen.

Stop sharing

Hide

to search



30°C Rain showers

policy Monte | RL class - p20190504@hyderabad | Meet - brc-sabd-pj | gym/blackjack.py at master · opencpu | Blackjack using On Policy Monte | +

lab.research.google.com/drive/1clJcpS6D6Dw2C9dYdbI8d3kXWxD\_Vr6#scrollTo=XlY8Z1lvNjM7

YouTube Maps

## Blackjack using On Policy Monte Carlo

View Insert Runtime Tools Help All changes saved

ext

return P  
turn policy

```
_pol_mc_control_learn(env, episodes, discount_factor, epsilon):  
    """  
    Monte Carlo Control using Epsilon-Greedy policies.  
    Finds an optimal epsilon-greedy policy.  
  
    Args:  
        env: Environment.  
        episodes: Number of episodes to sample.  
        discount_factor: Gamma discount factor.  
        epsilon: Chance the sample a random action. Float between 0 and 1.  
  
    Returns:  
        A tuple (Q, policy).  
        Q is a dictionary mapping state to action values.  
        Policy is the trained policy that returns action probabilities  
    """  
  
    Keeps track of sum and count of returns for each state  
    An array could be used to save all returns but that's memory inefficient.  
    defaultdict used so that the default value is stated if the observation(key) is not found  
    turns_sum = defaultdict(float)  
    turns_count = defaultdict(float)  
  
    The final action-value function.  
    A nested dictionary that maps state -> (action -> action-value).  
    = defaultdict(lambda: np.zeros(env.action_space.n))  
  
    The policy we're following
```

Comment Connect

30°C Rain showers

policy Monte | RL class - p20190504@hyderabad | Meet - brc-sabd-pj | gym/blackjack.py at master · opencog/gym | Blackjack using On Policy Monte | +

lab.research.google.com/drive/1clJcpS6D6Dw2C9dYdbi8d3kXWxD\_Vr6#scrollTo=XIY8Z1lvNjM7

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help All changes saved

ext

Monte Carlo control using Epsilon-Greedy policies.

and an optimal epsilon-greedy policy.

gs:

```
env: Environment.  
episodes: Number of episodes to sample.  
discount_factor: Gamma discount factor.  
epsilon: Chance the sample a random action. Float between 0 and 1.
```

turns:

A tuple (Q, policy).  
Q is a dictionary mapping state to action values.  
Policy is the trained policy that returns action probabilities

"

Keeps track of sum and count of returns for each state  
An array could be used to save all returns but that's memory inefficient.  
defaultdict used so that the default value is stated if the observation(key) is not found

```
turns_sum = defaultdict(float)  
turns_count = defaultdict(float)
```

The final action-value function.  
A nested dictionary that maps state -> (action -> action-value).  
= defaultdict(lambda: np.zeros(env.action\_space.n))

The policy we're following

```
l = create_epsilon_greedy_action_policy(env,Q,epsilon)
```

r i in range(1, episodes + 1):  
# Print out which episode we're on  
if i% 1000 == 0:  
print("\rEpisode {}/{}, ".format(i, episodes), end="")  
clear\_output(wait=True)

to search

○ Hi ! Mail Internet Explorer Google Chrome Google Photos Microsoft Edge

30°C Rain showers

policy Monte x RL class - p20190504@hyderabad x Meet - brc-ssbd-pj x gym/blackjack.py at master · opa x Blackjack using On Policy Monte x + lab.research.google.com/drive/1clJcpS6D6DwzC9dYdbI8d3kXWxD\_Vr6#scrollTo=XlY8Z1lvNjM7

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help All changes saved

ext

turns:  
A tuple (Q, policy).  
Q is a dictionary mapping state to action values.  
Policy is the trained policy that returns action probabilities

"Keeps track of sum and count of returns for each state  
An array could be used to save all returns but that's memory inefficient.  
defaultdict used so that the default value is stated if the observation(key) is not found

```
turns_sum = defaultdict(float)
turns_count = defaultdict(float)
```

The final action-value function.  
A nested dictionary that maps state -> (action -> action-value).  
= defaultdict(lambda: np.zeros(env.action\_space.n))

The policy we're following

```
l = create_epsilon_greedy_action_policy(env,Q,epsilon)

for i in range(1, episodes + 1):
    # Print out which episode we're on
    if i%1000 == 0:
        print("\rEpisode {}/{}.".format(i, episodes), end="")
        clear_output(wait=True)

    # Generate an episode.
    # An episode is an array of (state, action, reward) tuples
    episode = []
    state = env.reset()
    for t in range(100):
        probs = pol(state)
        action = np.random.choice(np.arange(len(probs)), p=probs)
```

to search

30°C Rain showers

Click using On Policy Monte Carlo

[View](#) [Insert](#) [Runtime](#) [Tools](#) [Help](#) [All changes saved](#)

## Comment

10

ext

Connect

```

r i in range(1, episodes + 1):
    # Print out which episode we're on
    if i%1000 == 0:
        print("\rEpisode {} / {}".format(i, episodes), end="")
        clear_output(wait=True)

    # Generate an episode.
    # An episode is an array of (state, action, reward) tuples
    episode = []
    state = env.reset()
    for t in range(100):
        probs = pol(state)
        action = np.random.choice(np.arange(len(probs)), p=probs)
        next_state, reward, done, _ = env.step(action)
        episode.append((state, action, reward))
        if done:
            break
        state = next_state

    # Find all (state, action) pairs we've visited in this episode
    # We convert each state to a tuple so that we can use it as a dict key
    sa_in_episode = set([(tuple(x[0]), x[1]) for x in episode])
    for state, action in sa_in_episode:
        sa_pair = (state, action)
        #First Visit MC:
        # Find the first occurrence of the (state, action) pair in the episode
        first_occurrence_idx = next(i for i,x in enumerate(episode)
                                     if x[0] == state and x[1] == action)
        # Sum up all rewards since the first occurrence
        G = sum([x[2]*(discount_factor**i) for i,x in enumerate(episode[first_occurrence_idx:])])
        # Calculate average return for this state over all sampled episodes
        state_to_return[state] = G

```

policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-ssbd-pj x | gym/blackjack.py at master · opencpu x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1ciJcpS6D6Dw2C9dYdbi8d3kXWxD\_Vr6#scrollTo=7jBLkFyZNfcV

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

File Insert Runtime Tools Help All changes saved

ext

An array could be used to save all returns but that's memory inefficient.  
defaultdict used so that the default value is stated if the observation(key) is not found

```
turns_sum = defaultdict(float)
turns_count = defaultdict(float)
```

The final action-value function.  
A nested dictionary that maps state -> (action -> action-value).  
- defaultdict(lambda: np.zeros(env.action\_space.n))

The policy we're following

```
l = create_epsilon_greedy_action_policy(env,Q,epsilon)
```

```
r i in range(episodes + 1):
    # Print out which episode we're on
    if i%1000 == 0:
        print("\rEpisode {}/{}.".format(i, episodes), end="")
        clear_output(wait=True)

    # Generate an episode.
    # An episode is an array of (state, action, reward) tuples
    episode = []
    state = env.reset()
    for t in range(100):
        probs = pol(state)
        action = np.random.choice(np.arange(len(probs)), p=probs)
        next_state, reward, done, _ = env.step(action)
        episode.append((state, action, reward))
        if done:
            break
        state = next_state
```

to search

30°C Rain showers

## Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help All changes saved

Comment



ext

Connect

```
1 = create_epsilon_greedy_action_policy(env,Q,epsilon)

2 i in range(1, episodes + 1):
3     # Print out which episode we're on
4     if i% 1000 == 0:
5         print("\rEpisode {}/{}.".format(i, episodes), end="")
6         clear_output(wait=True)

7     # Generate an episode,
8     # An episode is an array of (state, action, reward) tuples
9     episode = []
10    state = env.reset()
11    for t in range(100):
12        probs = pol(state)
13        action = np.random.choice(np.arange(len(probs)), p=probs)
14        next_state, reward, done, _ = env.step(action)
15        episode.append((state, action, reward))
16        if done:
17            break
18        state = next_state

19    # Find all (state, action) pairs we've visited in this episode
20    # We convert each state to a tuple so that we can use it as a dict key
21    sa_in_episode = set([(tuple(x[0]), x[1]) for x in episode])
22    for state, action in sa_in_episode:
23        sa_pair = (state, action)
24        #First Visit MC:
25        # Find the first occurrence of the (state, action) pair in the episode
26        first_occurrence_idx = next(i for i,x in enumerate(episode)
27                                      if x[0] == state and x[1] == action)
28        # Sum up all rewards since the first occurrence
```

policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-ssbd-pj x | gym/blackjack.py at master · ope x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1clJcpS6D6Dw2C9dYdbi8d3kXWxD\_Vr6#scrollTo=7jBLkfYZNfcV

YouTube Maps

## Blackjack using On Policy Monte Carlo

View Insert Runtime Tools Help All changes saved

ext

```
print("\rEpisode {}/{}, end={}".format(i, episodes), end="")
clear_output(wait=True)

# Generate an episode.
# An episode is an array of (state, action, reward) tuples
episode = []
state = env.reset()
for t in range(100):
    probs = pol(state)
    action = np.random.choice(np.arange(len(probs)), p=probs)
    next_state, reward, done, _ = env.step(action)
    episode.append((state, action, reward))
    if done:
        break
    state = next_state

# Find all (state, action) pairs we've visited in this episode
# We convert each state to a tuple so that we can use it as a dict key
sa_in_episode = set([(tuple(x[0]), x[1]) for x in episode])
for state, action in sa_in_episode:
    sa_pair = (state, action)
    #First visit MC:
    # Find the first occurrence of the (state, action) pair in the episode
    first_occurrence_idx = next(i for i,x in enumerate(episode)
                                 if x[0] == state and x[1] == action)
    # Sum up all rewards since the first occurrence
    G = sum([x[2]*(discount_factor**i) for i,x in enumerate(episode[first_occurrence_idx:])])
    # Calculate average return for this state over all sampled episodes
    returns_sum[sa_pair] += G
    returns_count[sa_pair] += 1.0
    Q[state][action] = returns_sum[sa_pair] / returns_count[sa_pair]
```

Comment

Connect

30°C Rain showers

lab.research.google.com/drive/1clJcpS6D6DwzC9dYdbi8d3kXWxD\_Vr6#scrollTo=7jBLkfYZNfcV

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help All changes saved

Comment



```
ext
x_y = np.array([i,j] for i in V.keys())
range_x = np.arange(min_x, max_x + 1)
range_y = np.arange(min_y, max_y + 1)
X, Y = np.meshgrid(range_x, range_y)

Find value for all (x, y) coordinates
noace = np.apply_along_axis(lambda _: V[(_[0], _[1], False)], 2, np.dstack([X, Y]))
ace = np.apply_along_axis(lambda _: V[(_[0], _[1], True)], 2, np.dstack([X, Y]))
```

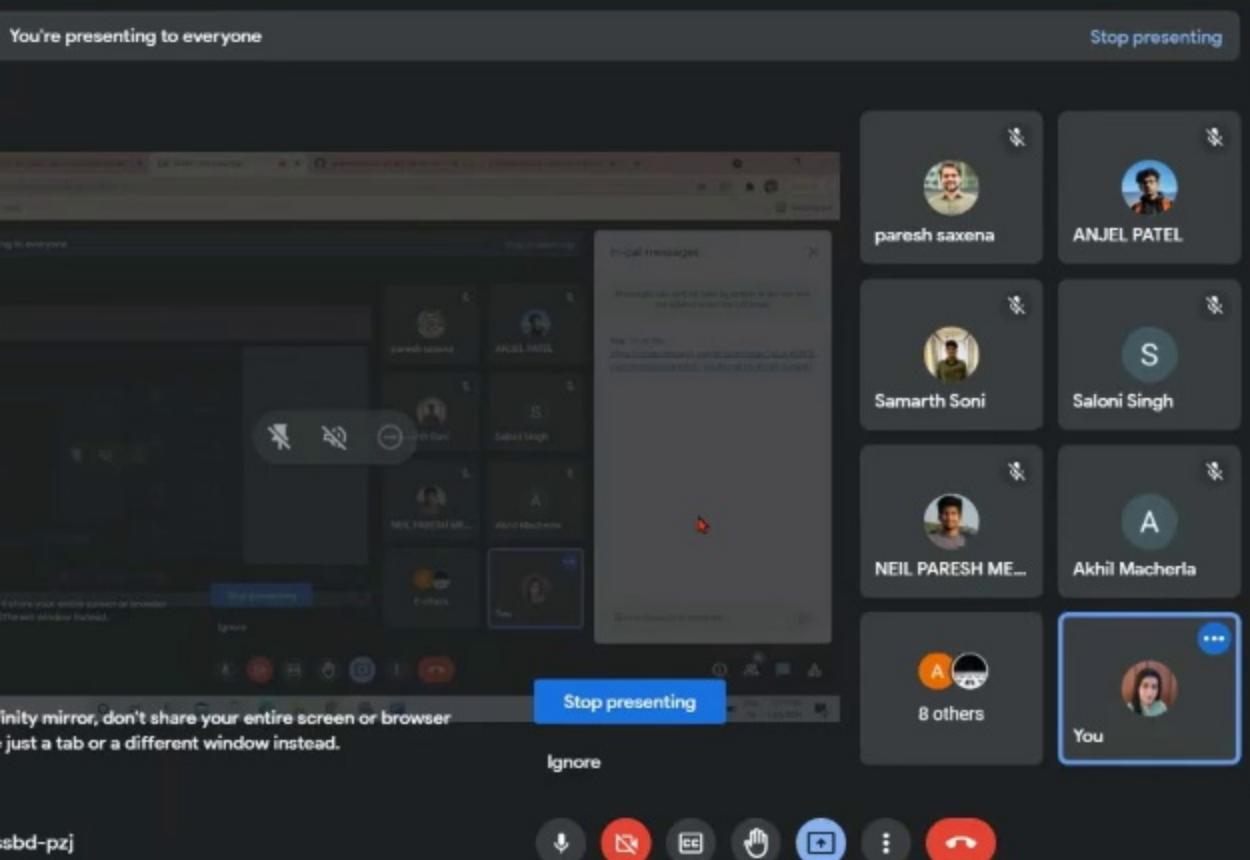
f plot\_surface(X, Y, Z, title):
 fig = plt.figure(figsize=(10,8))
 ax = fig.add\_subplot(111, projection='3d')
 surf = ax.plot\_surface(X, Y, Z, rstride=1, cstride=1,
 cmap=matplotlib.cm.coolwarm, vmin=-1.0, vmax=1.0)
 ax.set\_xlabel('Player Sum')
 ax.set\_ylabel('Dealer Showing')
 ax.set\_zlabel('Value')
 ax.set\_title(title)
 ax.view\_init(ax.elev, -120)
 fig.colorbar(surf)
 plt.show()

```
plot_surface(X, Y, Z_noace, "{} (No Usable Ace)".format(title))
plot_surface(X, Y, Z_ace, "{} (Usable Ace)".format(title))
```

```
create_epsilon_greedy_action_policy(env,Q,epsilon):
    "Create epsilon greedy action policy
gs:
    env: Environment
    Q: Q-table
```



meet.google.com/brc-ssbd-pzj?authuser=0&rpli=1



## In-call messages

Messages can only be seen by people who are deleted when the contact is removed.

You 12:03 PM  
<https://colab.research.google.com>  
DwzC9dYdbi8d3kXWxD\_Vrb#scroll

policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-sabd-pj x | gym/blackjack.py at master · op... x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1ciJcpS6D6DwzC9dYdbi8d3kXWxD\_Vr6#scrollTo=7jBLkFyZNfcV

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help All changes saved

ext Connect ▾

```
# Generate an episode.
# An episode is an array of (state, action, reward) tuples
episode = []
state = env.reset()
for t in range(100):
    probs = pol(state)
    action = np.random.choice(np.arange(len(probs)), p=probs)
    next_state, reward, done, _ = env.step(action)
    episode.append((state, action, reward))
    if done:
        break
    state = next_state

# Find all (state, action) pairs we've visited in this episode
# We convert each state to a tuple so that we can use it as a dict key
sa_in_episode = set([(tuple(x[0]), x[1]) for x in episode])
for state, action in sa_in_episode:
    sa_pair = (state, action)
    #First Visit MC:
    # Find the first occurrence of the (state, action) pair in the episode
    first_occurrence_idx = next(i for i, x in enumerate(episode)
                                 if x[0] == state and x[1] == action)
    # Sum up all rewards since the first occurrence
    G = sum([x[2]*(discount_factor**i) for i, x in enumerate(episode[first_occurrence_idx:])])
    # Calculate average return for this state over all sampled episodes
    returns_sum[sa_pair] += G
    returns_count[sa_pair] += 1.0
    Q[state][action] = returns_sum[sa_pair] / returns_count[sa_pair]

turn Q, pol
```

I

 Google Chrome  
You're presenting to [redacted]  
Click here to return to [redacted]  
when you're ready to [redacted]  
meet.google.com

to search

30°C Rain showers

policy Monte x | My RL class - p20190504@hyderabad x | Meet - b7c-ssbd-pj x | gym/blackjack.py at master · opencpu x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1icJcpS6D6DwzC9dYdbi8d3kXWxD\_Vr6#scrollTo=7jBLkfYZNfcV

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

File Insert Runtime Tools Help All changes saved

ext

Connect ▾

```
episode = []
state = env.reset()
for t in range(100):
    probs = pol(state)
    action = np.random.choice(np.arange(len(probs)), p=probs)
    next_state, reward, done, _ = env.step(action)
    episode.append((state, action, reward))
    if done:
        break
    state = next_state

# Find all (state, action) pairs we've visited in this episode
# We convert each state to a tuple so that we can use it as a dict key
sa_in_episode = set([(tuple(x[0]), x[1]) for x in episode])
for state, action in sa_in_episode:
    sa_pair = (state, action)
    # First Visit MC:
    # Find the first occurrence of the (state, action) pair in the episode
    first_occurrence_idx = next(i for i, x in enumerate(episode)
                                 if x[0] == state and x[1] == action)
    # Sum up all rewards since the first occurrence
    G = sum([x[2]*(discount_factor**i) for i, x in enumerate(episode[first_occurrence_idx:])])
    # Calculate average return for this state over all sampled episodes
    returns_sum[sa_pair] += G
    returns_count[sa_pair] += 1.0
    Q[state][action] = returns_sum[sa_pair] / returns_count[sa_pair]

turn Q, pol
```

Google Chrome

You're presenting to [redacted]  
Click here to return to [redacted]  
when you're ready to [redacted]  
meet.google.com

30°C Rain showers

Policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-ssbd-pj x | gym/blackjack.py at master · opencood/gym x Blackjack using On Policy Monte x +

lab.research.google.com/drive/1clJcpS6D6Dw2C9dYdbi8d3kXWxD\_Vr6#scrollTo=XlYZ1lvNjM7

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

File Insert Runtime Tools Help All changes saved

ext

```
episode.append((state, action, reward))
if done:
    break
state = next_state

# Find all (state, action) pairs we've visited in this episode
# We convert each state to a tuple so that we can use it as a dict key
sa_in_episode = set([(tuple(x[0]), x[1]) for x in episode])
for state, action in sa_in_episode:
    sa_pair = (state, action)
    #First visit MC:
    # Find the first occurrence of the (state, action) pair in the episode
    first_occurrence_idx = next(i for i,x in enumerate(episode)
                                 if x[0] == state and x[1] == action)
    # Sum up all rewards since the first occurrence
    G = sum([x[2]*(discount_factor**i) for i,x in enumerate(episode[first_occurrence_idx:])])
    # Calculate average return for this state over all sampled episodes
    returns_sum[sa_pair] += G
    returns_count[sa_pair] += 1.0
    Q[state][action] = returns_sum[sa_pair] / returns_count[sa_pair]

turn Q, pol
```

Comment Connect

30°C Rain showers

policy Monte... x | RL class - p20190504@hyderabad... x | Meet - brc-ssbd-p2j x | gym/blackjack.py at master · op... x | Blackjack using On Policy Monte... x +

lab.research.google.com/drive/1iciJcpS6D6DwzC9dYdbI8d3lkXWxD\_Vr6#scrollTo=XIY0Z1lvNjM7

YouTube Maps

## Blackjack using On Policy Monte Carlo ★

File Insert Runtime Tools Help All changes saved

ext

```
surf = ax.plot_surface(X, Y, Z_noace, rstride=1, cstride=1,
                       cmap=matplotlib.cm.coolwarm, vmin=-1.0, vmax=1.0)
ax.set_xlabel('Player Sum')
ax.set_ylabel('Dealer Showing')
ax.set_zlabel('Value')
ax.set_title(title)
ax.view_init(ax.elev, -120)
fig.colorbar(surf)
plt.show()
```

ot\_surface(X, Y, Z\_noace, "{} (No Usable Ace)".format(title))
ot\_surface(X, Y, Z\_ace, "{} (Usable Ace)".format(title))

```
ate_epsilon_greedy_action_policy(env,Q,epsilon):
    " Create epsilon greedy action policy
gs:
    env: Environment
    Q: Q_table
    epsilon: Probability of selecting random action instead of the 'optimal' action

turns:
    Epsilon-greedy-action Policy function with Probabilities of each action for each state
"
f policy(obs):
    P = np.ones(env.action_space.n, dtype=float) * epsilon / env.action_space.n #initiate with same prob for all actions
    best_action = np.argmax(Q[obs]) #get best action
    P[best_action] += (1.0 - epsilon)
    return P
turn policy
```

to search

30°C Rain showers

```
YouTube Maps
41     return self._hit(hand) == [1, 10]
42
43
44
45     class BlackjackEnv(gym.Env):
46         """Simple Blackjack environment.
47
48         Blackjack is a card game where the goal is to obtain cards that sum to as
49         near as possible to 21 without going over. They're playing against a fixed
50         dealer.
51         Face cards (Jack, Queen, King) have point value 10.
52         Aces can count as 1 or 11; and it's called 'usable' at 11.
53         This game is played with an infinite deck (or with replacement).
54         The game starts with dealer having one face up card, while
55         player having two face up cards. (Virtually for all Blackjack games today).
56
57         The player can request additional cards (hit=1) until they decide to stop
58         (stick=0) or exceed 21 (bust).
59
60         After the player sticks, the dealer reveals their facedown card, and draws
61         until their sum is 17 or greater. If the dealer goes bust the player wins.
62
63         If neither player nor dealer busts, the outcome (win, lose, draw) is
64         decided by whose sum is closer to 21. The reward for winning is +1,
65         drawing is 0, and losing is -1.
66
67         The observation of a 3-tuple of: the player's current sum,
68         the dealer's own showing card (1-10 where 1 is ace),
69         and whether or not the player holds a usable ace (0 or 1).
69
70
71         This environment corresponds to the version of the Blackjack problem
72         described in Example 5.1 in Reinforcement Learning: An Introduction
73         by Sutton and Barto.
74         http://incompleteideas.net/book/the-book-2nd.html
75
76         def __init__(self, natural=False, sab=False):
```

Policy Monte x | M RL class - p20190504@hyderabad x | Meet - brc-sabd-pj x | gym/blackjack.py at master · open x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1icJcpS6D6Dw2C9dYdbi8d3kXWxD\_Vr6#scrollTo=rdbtYKY60BWp

YouTube Maps

## Blackjack using On Policy Monte Carlo

View Insert Runtime Tools Help

ext

```
ym import envs
numpy as np
matplotlib
matplotlib.pyplot as plt
pl_toolkits.mplot3d import Axes3D
matplotlib as mpl
matplotlib import cm
collections import defaultdict
Python.display import clear_output
otlib inline
```

gym.make('Blackjack-v0')

+ Code + Text

```
pl_toolkits.axes_grid1 import make_axes_locatable
pl_toolkits.mplot3d import Axes3D
ot_policy(policy):

f get_z(player_hand, dealer_showing, usable_ace):
    if (player_hand, dealer_showing, usable_ace) in policy:
        return policy[player_hand, dealer_showing, usable_ace]
    else:
        return 1

f get_figure(usable_ace, ax):
    x_range = np.arange(1, 11)
    y_range = np.arange(11, 22)
    X, Y = np.meshgrid(x_range, y_range)
    Z = np.array([get_z(player_hand, dealer_showing, usable_ace) for dealer_showing in x_range for player_hand in range(1, 10, 1)])
    Os completed at 12:18 PM
```

30°C Rain showers



YouTube Maps

```
66     the observation of a 3-tuple of: the players current sum,  
67     the dealer's one showing card (1-10 where 1 is ace),  
68     and whether or not the player holds a usable ace (0 or 1).  
69  
70     This environment corresponds to the version of the blackjack problem  
71     described in Example 5.1 in Reinforcement Learning: An Introduction  
72     by Sutton and Barto.  
73     http://incompleteideas.net/book/the-book-2nd.html  
74     ....  
75  
76     def __init__(self, natural=False, sab=False):  
77         self.action_space = spaces.Discrete(2)  
78         self.observation_space = spaces.Tuple(  
79             (spaces.Discrete(32), spaces.Discrete(11), spaces.Discrete(2)))  
80     )  
81         self.seed()  
82  
83         # Flag to payout 1.5 on a "natural" blackjack win, like casino rules  
84         # Ref: http://www.bicyclecards.com/how-to-play/blackjack/  
85         self.natural = natural  
86  
87         # Flag for full agreement with the (Sutton and Barto, 2018) definition. Overrides self.natural  
88         self.sab = sab  
89  
90     def seed(self, seed=None):  
91         self.np_random, seed = seeding.np_random(seed)  
92         return [seed]  
93  
94     def step(self, action):  
95         assert self.action_space.contains(action)  
96         if action: # hit: add a card to players hand and return  
97             self.player.append(draw_card(self.np_random))  
98             if is_bust(self.player):  
99                 done = True  
100                 reward = -1.0  
101             else:
```

policy Monte x | RL class · p20190504@hyderabad x | Meet · brc-sabd-pj x | gym/blackjack.py at master · op... x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1ldJcpS6D6Dw2C9dYdbi8d3kXWxD\_Vr6#scrollTo=4-z2JD\_z0lyJ

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

File Insert Runtime Tools Help Saving failed since 12:19 PM

RAM Disk

```
ext
matplotlib
matplotlib.pyplot as plt
pl_toolkits.mplot3d import Axes3D
matplotlib as mpl
matplotlib import cm
collections import defaultdict
Python.display import clear_output
otlib inline

gym.make('Blackjack-v0')

servation_space

Discrete(32), Discrete(11), Discrete(2))

tion

pl_toolkits.axes_grid1 import make_axes_locatable
pl_toolkits.mplot3d import Axes3D
ot_policy(policy):

    f get_Z(player_hand, dealer_showing, usable_ace):
        if (player_hand, dealer_showing, usable_ace) in policy:
            return policy[player_hand, dealer_showing, usable_ace]
        else:
            return 1

Failed. This file was updated remotely or in another tab. Show diff
```

0s completed at 12:19 PM

30°C Rain showers

policy Monte Carlo x | RL class - p20190504@hyderabad x | Meet - brc-sabd-pj x | gym/blackjack.py at master · openui5/gym x Blackjack using On Policy Monte Carlo x + lab.research.google.com/drive/1clJcpS6D6Dw2C9dYdbi8d3kXWxD\_Vr6#scrollTo=jgnqat9f0577

YouTube Maps

## Blackjack using On Policy Monte Carlo

View Insert Runtime Tools Help Saving failed since 12:19 PM

ext  
einer  
, 10]

```
pl_toolkits.axes_grid1 import make_axes_locatable
pl_toolkits.mplot3d import Axes3D
ot_policy(policy):

def get_z(player_hand, dealer_showing, usable_ace):
    if (player_hand, dealer_showing, usable_ace) in policy:
        return policy[player_hand, dealer_showing, usable_ace]
    else:
        return 1

def get_figure(usable_ace, ax):
    x_range = np.arange(1, 11)
    y_range = np.arange(11, 22)
    X, Y = np.meshgrid(x_range, y_range)
    Z = np.array([[get_z(player_hand, dealer_showing, usable_ace) for dealer_showing in x_range] for player_hand in range(21, 10, -1)])
    surf = ax.imshow(Z, cmap=plt.get_cmap('Accent', 2), vmin=0, vmax=1, extent=[0.5, 10.5, 10.5, 21.5])
    plt.xticks(x_range, ('A', '2', '3', '4', '5', '6', '7', '8', '9', '10'))
    plt.yticks(y_range)
    ax.set_xlabel('Dealer Showing')
    ax.set_ylabel('Player Hand')
    ax.grid(color='black', linestyle='-', linewidth=1)
    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="5%", pad=0.1)
    cbar = plt.colorbar(surf, ticks=[0, 1], cax=cax)
    cbar.ax.set_yticklabels(['0 (STICK)', '1 (HIT)'])

failed. This file was updated remotely or in another tab. Show diff
```

✓ 0s completed at 12:23 PM

30°C Rain showers

policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-sabd-pj x | gym/blackjack.py at master · op... x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1ciJcpS6D6DwzC9dYdbi8d3lkXwxD\_Vr6#scrollTo=jgnqat9t0577

YouTube Maps

## Blackjack using On Policy Monte Carlo

View Insert Runtime Tools Help Saving failed since 12:19 PM

Comment

RAM Disk

```
f policy(obs):
    P = np.ones(env.action_space.n, dtype=float) * epsilon / env.action_space.n #initiate with same prob for all actions
    best_action = np.argmax(Q[obs]) #get best action
    P[best_action] += (1.0 - epsilon)
    return P

turn policy

def pol_mc_control_learn(env, episodes, discount_factor, epsilon):
    """
    Monte Carlo Control using Epsilon-Greedy policies.
    Finds an optimal epsilon-greedy policy.

    Parameters:
    env: Environment.
    episodes: Number of episodes to sample.
    discount_factor: Gamma discount factor.
    epsilon: Chance the sample a random action. Float between 0 and 1.

    Returns:
    A tuple (Q, policy).
    Q is a dictionary mapping state to action values.
    Policy is the trained policy that returns action probabilities
    """

    Keeps track of sum and count of returns for each state
    An array could be used to save all returns but that's memory inefficient.
    defaultdict used so that the default value is stated if the observation(key) is not found
    turns_sum = defaultdict(float)
    turns_count = defaultdict(float)

    Failed. This file was updated remotely or in another tab. Show diff
```

0s completed at 12:23 PM

30°C Rain showers

policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-sabd-pj x | gym/blackjack.py at master · open x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1ciJcpS6D6Dw2C9dYdbi8d3kXWxD\_Vr6#scrollTo=XlY8Z1lvNjM7

YouTube Maps

## Blackjack using On Policy Monte Carlo

View Insert Runtime Tools Help Saving failed since 12:19 PM

Comment

RAM Disk

```
P = np.ones(env.action_space.n, dtype=float) * epsilon / env.action_space.n #initiate with same prob for all actions
best_action = np.argmax(Q[obs]) #get best action
P[best_action] += (1.0 - epsilon)
return P
turn policy

def pol_mc_control_learn(env, episodes, discount_factor, epsilon):
    """
    Monte Carlo Control using Epsilon-Greedy policies.
    Finds an optimal epsilon-greedy policy.

    Args:
        env: Environment.
        episodes: Number of episodes to sample.
        discount_factor: Gamma discount factor.
        epsilon: Chance the sample a random action. Float between 0 and 1.

    Returns:
        A tuple (Q, policy).
        Q is a dictionary mapping state to action values.
        Policy is the trained policy that returns action probabilities
    """
    # Keeps track of sum and count of returns for each state
    # An array could be used to save all returns but that's memory inefficient.
    # defaultdict used so that the default value is stated if the observation(key) is not found
    turns_sum = defaultdict(float)
    turns_count = defaultdict(float)

    The final action-value function.
```

Failed. This file was updated remotely or in another tab. [Show diff](#)

✓ 0s completed at 12:23 PM

30°C Rain showers

policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-sabd-pj x | gym/blackjack.py at master · opa x | Blackjack using On Policy Monte x +

lab.research.google.com/drive/1clJcp56D6Dw2C9dYdbi8d3kXWxD\_Vr6#scrollTo=XIYZ1lvNjM7

YouTube Maps

## Blackjack using On Policy Monte Carlo

View Insert Runtime Tools Help Saving failed since 12:19 PM

ext

turns:

A tuple (Q, policy).  
Q is a dictionary mapping state to action values.  
Policy is the trained policy that returns action probabilities

"Keeps track of sum and count of returns for each state  
An array could be used to save all returns but that's memory inefficient.  
defaultdict used so that the default value is stated if the observation(key) is not found

```
turns_sum = defaultdict(float)
turns_count = defaultdict(float)
```

The final action-value function.  
A nested dictionary that maps state -> (action -> action-value).  
= defaultdict(lambda: np.zeros(env.action\_space.n))

The policy we're following

```
l = create_epsilon_greedy_action_policy(env,Q,epsilon)
```

r i in range(1, episodes + 1):  
# Print out which episode we're on  
if i% 1000 == 0:  
 print("\rEpisode {}/{}.".format(i, episodes), end="")  
 clear\_output(wait=True)

# Generate an episode.  
# An episode is an array of (state, action, reward) tuples  
episode = []  
state = env.reset()  
for t in range(100):

Failed. This file was updated remotely or in another tab. Show diff

RAM Disk

0s completed at 12:23 PM

30°C Rain showers

## Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help Saving failed since 12:19 PM

Comment



ext

RAM Disk

```
print("\rEpisode {}/{}, end={}".format(i, episodes), end="")
clear_output(wait=True)

# Generate an episode.
# An episode is an array of (state, action, reward) tuples
episode = []
state = env.reset()
for t in range(100):
    probs = pol(state)
    action = np.random.choice(np.arange(len(probs)), p=probs)
    next_state, reward, done, _ = env.step(action)
    episode.append((state, action, reward))
    if done:
        break
it def __call__(keep_kernel=False)

# Find all (state, action) pairs we've visited in this episode
# We convert each state to a tuple so that we can use it as a dict key
sa_in_episode = set([(tuple(x[0]), x[1]) for x in episode])
for state, action in sa_in_episode:
    sa_pair = (state, action)
    #First visit MC:
    # Find the first occurrence of the (state, action) pair in the episode
    first_occurrence_idx = next(i for i,x in enumerate(episode)
                                 if x[0] == state and x[1] == action)
    # Sum up all rewards since the first occurrence
    G = sum([x[2]**(discount_factor**i) for i,x in enumerate(episode[first_occurrence_idx:])])
    # Calculate average return for this state over all sampled episodes
    returns_sum[sa_pair] += G
    returns_count[sa_pair] += 1
```

Failed. This file was updated remotely or in another tab. Show diff

it[sa\_pair]

✓ 0s completed at 12:23 PM

Solving MDPs by Backpropagation Using On-Policy Monte Carlo

View Insert Runtime Tools Help Saving failed since 12:19 PM

## Comment

1

ext

```

# Find the first occurrence of the (state, action) pair in the episode
first_occurrence_idx = next(i for i,x in enumerate(episode)
                             if x[0] == state and x[1] == action)
# Sum up all rewards since the first occurrence
G = sum([x[2]*(discount_factor**i) for i,x in enumerate(episode[first_occurrence_idx:])])
# Calculate average return for this state over all sampled episodes
returns_sum[sa_pair] += G
returns_count[sa_pair] += 1.0
Q[state][action] = returns_sum[sa_pair] / returns_count[sa_pair]

```

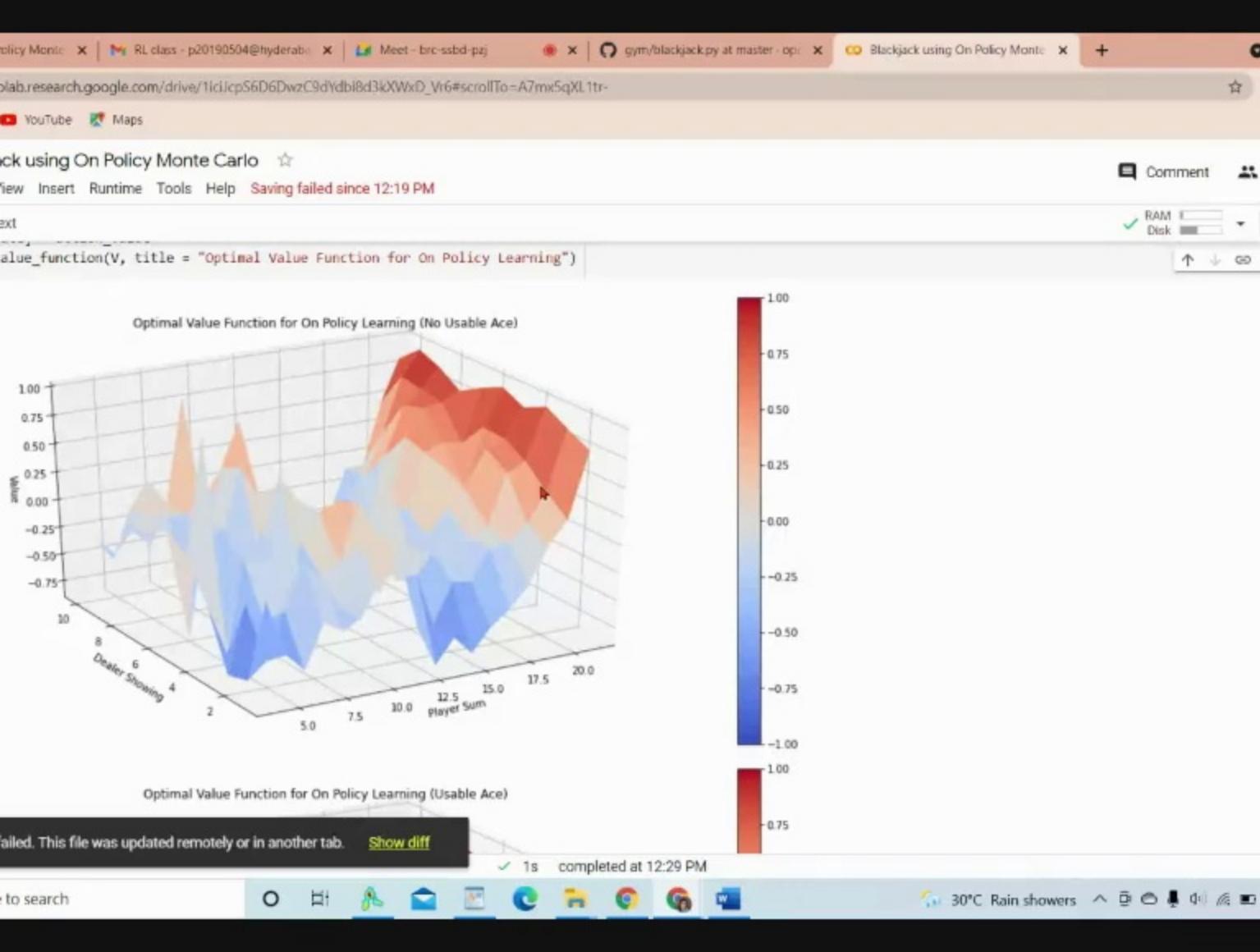
turn Q, pol

`set()`

failed. This file was updated remotely or in another tab. [Show diff](#)

Show diff

✓ 0s completed at 12:23 PM



policy Monte x | RL class - p20190504@hyderabad x | Meet - brc-sabd-pj x | gym/blackjack.py at master · openclosed/gym x Blackjack using On Policy Monte x +

lab.research.google.com/drive/1clJcpS6D6Dw2C9dYdbi8d3kXwxD\_Vr6#scrollTo=nMLqagD1OT7

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

File Insert Runtime Tools Help Saving failed since 12:19 PM

ext

turn Q, pol

set()

ol,On\_MC\_Learned\_Policy = On\_pol\_mc\_control\_learn(env, 5000, 0.9, 0.05)

e 5000/5000.

faultdict(float)

ate, actions in Q\_on\_pol.items():

on\_value = np.max(actions)

ate] = action\_value

alue\_function(V, title = "Optimal Value Function for On Policy Learning")

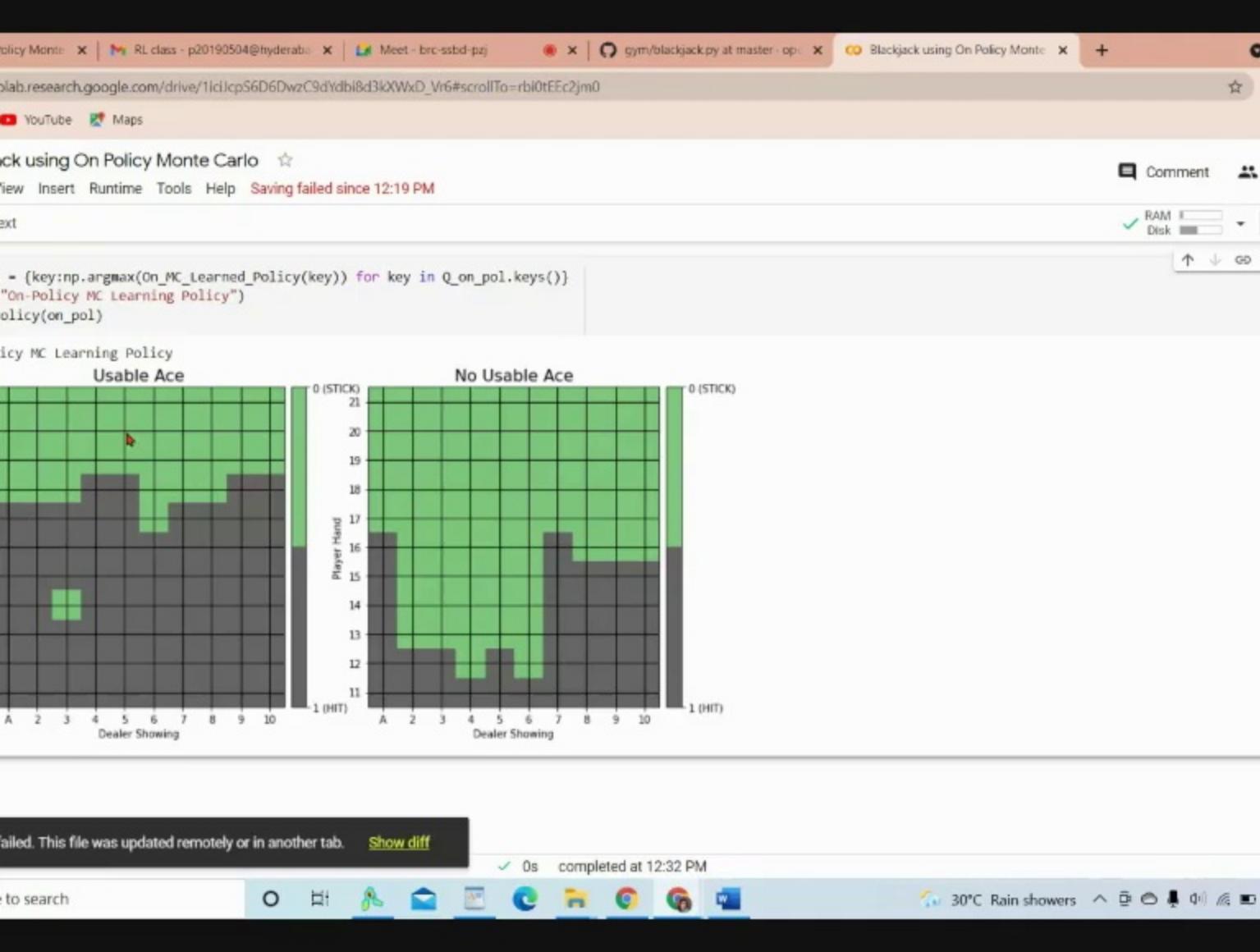
Optimal Value Function for On Policy Learning (No Usable Ace)

1s completed at 12:29 PM

This file was updated remotely or in another tab. Show diff

to search

30°C Rain showers



YouTube Maps

## Blackjack using On Policy Monte Carlo ⭐

View Insert Runtime Tools Help Saving failed since 12:19 PM

Comment



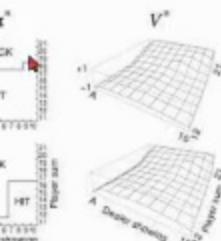
ext

RAM Disk

For all  $a \in \mathcal{A}(s)$ :

$$\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

## Optimal policy and state value function for blackjack



```
gym
ym import envs
numpy as np
matplotlib
matplotlib.pyplot as plt
pl_toolkits.mplot3d import Axes3D
matplotlib as mpl
matplotlib import cm
```

Failed. This file was updated remotely or in another tab. Show diff

✓ 0s completed at 12:32 PM

to search



30°C Rain showers

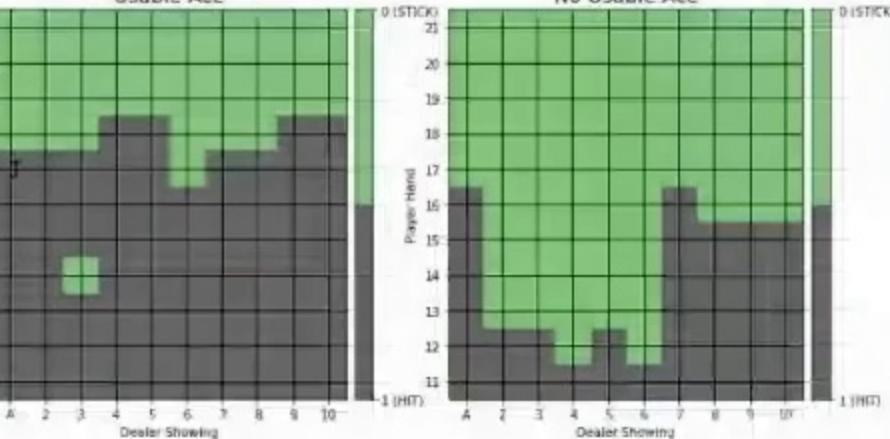
```

= {key:np.argmax(On_MC_Learned_Policy(key)) for key in Q_on_pol.keys()}
"On-Policy MC learning Policy")
olicy(on_pol)

```

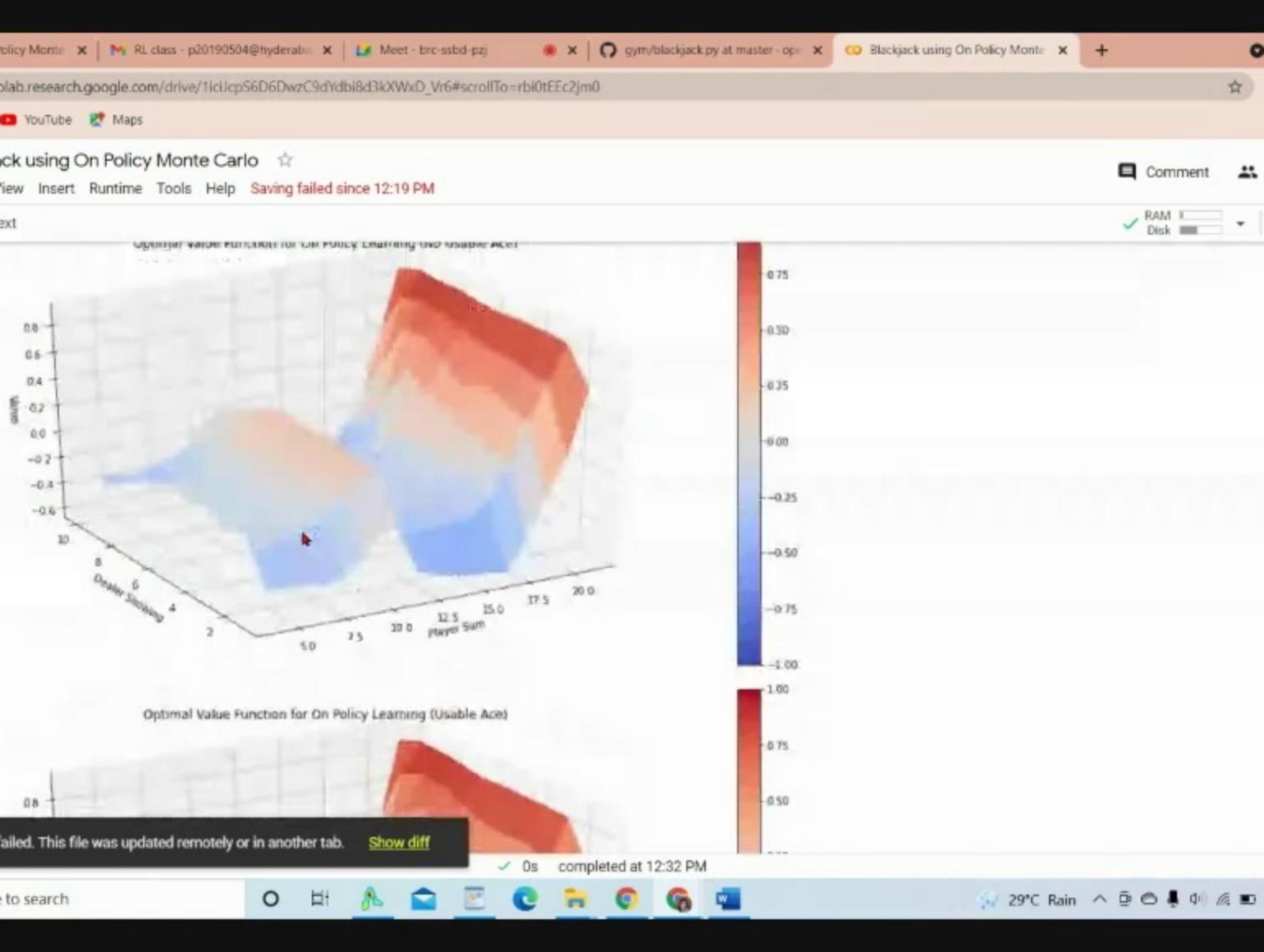
Policy MC Learning Policy

Usable Ace



Failed. This file was updated remotely or in another tab. [Show diff](#)

✓ 0s completed at 12:32 PM



RL class - p20190504@hyderabad | Meet - brc-ssbd-pj | gym/blackjack.py at master · opencpu/gym | Blackjack using On Policy Monte Carlo

sheet.google.com/brc-ssbd-pj?authuser=0&pli=1

You're presenting to everyone

Stop presenting

In-call messages

Messages can only be seen by people in the call. They are deleted when the call ends.

You 12:03 PM [https://colab.research.google.com/DwzC9dYdbi8d3kXWxD\\_Vr6#scrollTo](https://colab.research.google.com/DwzC9dYdbi8d3kXWxD_Vr6#scrollTo)

RUBAN S 12:20 PM why Tuple(Discrete(32), Discrete(1))

infinity mirror, don't share your entire screen or browser just a tab or a different window instead.

Stop presenting

Ignore

7 others

You

Send a message to everyone

csbd-pj

o II G E M F C G W

29°C Rain



YouTube Maps

```
56     The player can request additional cards (hit=1) until they decide to stop
57     (stick=0) or exceed 21 (bust).
58
59     After the player sticks, the dealer reveals their facedown card, and draws
60     until their sum is 17 or greater. If the dealer goes bust the player wins.
61
62     If neither player nor dealer busts, the outcome (win, lose, draw) is
63     decided by whose sum is closer to 21. The reward for winning is +1,
64     drawing is 0, and losing is -1.
65
66     The observation of a 3-tuple of: the players current sum,
67     the dealers one showing card (1-10 where 1 is ace),
68     and whether or not the player holds a usable ace (0 or 1).
69
70     This environment corresponds to the version of the blackjack problem
71     described in Example 5.1 in Reinforcement Learning: An Introduction
72     by Sutton and Barto.
73     http://incompleteideas.net/book/the-book-2nd.html
74     """
75
76     def __init__(self, natural=False, sab=False):
77         self.action_space = spaces.Discrete(2)
78         self.observation_space = spaces.Tuple(
79             (spaces.Discrete(32), spaces.Discrete(11), spaces.Discrete(2)))
80     )
81     self.seed()
82
83     # Flag to payout 1.5 on a "natural" blackjack win, like casino rules
84     # Ref: http://www.bicyclecards.com/how-to-play/blackjack/
85     self.natural = natural
86
87     # Flag for full agreement with the (Sutton and Barto, 2018) definition. Overrides self.natural
88     self.sab = sab
89
90     def seed(self, seed=None):
91         self.np_random, seed = seeding.np_random(seed)
```

meet.google.com is sharing your screen. Stop sharing Hide

YouTube Maps

## Blackjack using On Policy Monte Carlo ☆

View Insert Runtime Tools Help Saving failed since 12:19 PM

Comment



ext

RAM Disk

servation\_space

Discrete(32), Discrete(11), Discrete(2))

tion\_space.n

set()

, False)



+ Code

+ Text

ep(0)

8, False), -1.0, True, {})

ep(1)

8, False), -1.0, True, {})

aler

]

ailed. This file was updated remotely or in another tab.

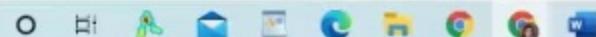
Show diff

meet.google.com is sharing your screen.

Stop sharing

Hide

to search



29°C Rain

YouTube Maps

```
56     The player can request additional cards (hit=1) until they decide to stop
57     (stick=0) or exceed 21 (bust).
58
59     After the player sticks, the dealer reveals their facedown card, and draws
60     until their sum is 17 or greater. If the dealer goes bust the player wins.
61
62     If neither player nor dealer busts, the outcome (win, lose, draw) is
63     decided by whose sum is closer to 21. The reward for winning is +1,
64     drawing is 0, and losing is -1.
65
66     The observation of a 3-tuple of: the players current sum,
67     the dealers one showing card (1-10 where 1 is ace),
68     and whether or not the player holds a usable ace (0 or 1).
69
70     This environment corresponds to the version of the blackjack problem
71     described in Example 5.1 in Reinforcement Learning: An Introduction
72     by Sutton and Barto.
73     http://incompleteideas.net/book/the-book-2nd.html
74     """
75
76     def __init__(self, natural=False, sab=False):
77         self.action_space = spaces.Discrete(2)
78         self.observation_space = spaces.Tuple(
79             (spaces.Discrete(32), spaces.Discrete(11), spaces.Discrete(2)))
80     )
81     self.seed()
82
83     # Flag to payout 1.5 on a "natural" blackjack win, like casino rules
84     # Ref: http://www.bicyclecards.com/how-to-play/blackjack/
85     self.natural = natural
86
87     # Flag for full agreement with the (Sutton and Barto, 2018) definition. Overrides self.natural
88     self.sab = sab
89
90     def seed(self, seed=None):
91         self.np_random, seed = seeding.np_random(seed)
```

meet.google.com is sharing your screen.

Stop sharing

Hide



YouTube Maps

```
77     self.action_space = spaces.Discrete(2)
78     self.observation_space = spaces.Tuple(
79         (spaces.Discrete(32), spaces.Discrete(11), spaces.Discrete(2)))
80     )
81     self.seed()
82
83     # Flag to payout 1.5 on a "natural" blackjack win, like casino rules
84     # Ref: http://www.bicyclecards.com/how-to-play/blackjack/
85     self.natural = natural
86
87     # Flag for full agreement with the (Sutton and Barto, 2018) definition. Overrides self.natural
88     self.sab = sab
89
90
91     def seed(self, seed=None):
92         self.np_random, seed = seeding.np_random(seed)
93         return [seed]
94
95     def step(self, action):
96         assert self.action_space.contains(action)
97         if action: # hit: add a card to players hand and return
98             self.player.append(draw_card(self.np_random))
99             if is_bust(self.player):
100                 done = True
101                 reward = -1.0
102             else:
103                 done = False
104                 reward = 0.0
105         else: # stick: play out the dealers hand, and score
106             done = True
107             while sum_hand(self.dealer) < 17:
108                 self.dealer.append(draw_card(self.np_random))
109             reward = cmp(score(self.player), score(self.dealer))
110             if self.sab and is_natural(self.player) and not is_natural(self.dealer):
111                 # Player automatically wins. Rules con
112                 reward = 1.0
113             elif (
```

meet.google.com is sharing your screen. Stop sharing Hide

RL class - p20190504@hyderabad | Meet - brc-ssbd-pzj | gym/blackjack.py at master · open | Blackjack using On Policy Monte Carlo Control · GitHub

sheet.google.com/brc-ssbd-pzj?authuser=0&pli=1

You're presenting to everyone

Stop presenting

RUBAN S paresh saxena

ANJEL PATEL Samarth Soni

Saloni Singh NEIL PARESH ME...

7 others You

In-call messages

Messages can only be seen by people in the call. They are deleted when the call ends.

You 12:03 PM <https://colab.research.google.com/> DwzC9dYdbi8d3kXWxD\_Vr6#scroll

RUBAN S 12:20 PM why Tuple(Discrete(32), Discrete(1))

infinity mirror, don't share your entire screen or browser just a tab or a different window instead.

Stop presenting

Ignore

meet.google.com is sharing your screen. Stop sharing Hide

29°C Rain

```

42
43
44 class BlackjackEnv(gym.Env):
45     """Simple Blackjack environment
46
47     Blackjack is a card game where the goal is to obtain cards that sum to as
48     near as possible to 21 without going over. They're playing against a fixed
49     dealer.
50     Face cards (Jack, Queen, King) have point value 10.
51     Aces can either count as 11 or 1, and it's called 'usable' at 11.
52     This game is played with an infinite deck (or with replacement).
53     The game starts with dealer having one face up and one face down card, while
54     player having two face up cards. (Virtually for all Blackjack games today).
55
56     The player can request additional cards (hit=1) until they decide to stop
57     (stick=0) or exceed 21 (bust).
58
59     After the player sticks, the dealer reveals their facedown card, and draws
60     until their sum is 17 or greater. If the dealer goes bust the player wins.
61
62     If neither player nor dealer busts, the outcome (win, lose, draw) is
63     decided by whose sum is closer to 21. The reward for winning is +1,
64     drawing is 0, and losing is -1.
65
66     The observation of a 3-tuple of: the players current sum,
67     the dealers one showing card (1-10 where 1 is ace),
68     and whether or not the player holds a usable ace (0 or 1).
69
70     This environment corresponds to the version of the blackjack problem
71     described in Example 5.1 in Reinforcement Learning: An Introduction
72     by Sutton and Barto.
73     http://incompleteideas.net/book/the-book-2nd.html
74
75
76     def __init__(self, natural=False, seed=False):
77         self.action_space = spaces.Discrete(2)

```

meet.google.com is sharing your screen. Stop sharing Hide

 Google Chrome

You're presenting to  
Click here to return to  
when you're ready to  
meet.google.com

```
67     the dealer's one showing card (1-10 where 1 is ace),  
68     and whether or not the player holds a usable ace (0 or 1).  
69  
70     This environment corresponds to the version of the blackjack problem  
71     described in Example 5.1 in Reinforcement Learning: An Introduction  
72     by Sutton and Barto.  
73     http://incompleteideas.net/book/the-book-2nd.html  
74     """  
75  
76     def __init__(self, natural=False, sab=False):  
77         self.action_space = spaces.Discrete(2)  
78         self.observation_space = spaces.Tuple(  
79             (spaces.Discrete(32), spaces.Discrete(11), spaces.Discrete(2))  
80         )  
81         self.seed()  
82  
83         # Flag to payout 1.5 on a "natural" blackjack win, like casino rules  
84         # Ref: http://www.bicyclecards.com/how-to-play/blackjack/  
85         self.natural = natural  
86  
87         # Flag for full agreement with the (Sutton and Barto, 2018) definition. Overrides self.natural  
88         self.sab = sab  
89  
90     def seed(self, seed=None):  
91         self.np_random, seed = seeding.np_random(seed)  
92         return [seed]  
93  
94     def step(self, action):  
95         assert self.action_space.contains(action)  
96         if action: # hit: add a card to players hand and return  
97             self.player.append(draw_card(self.np_random))  
98             if is_bust(self.player):  
99                 done = True  
100                reward = -1.0  
101            else:  
102                done = False
```

meet.google.com is sharing your screen. Stop sharing Hide

```
51     Aces can either count as 11 or 1, and it's called 'usable' at 11.  
52     This game is played with an infinite deck (or with replacement).  
53     The game starts with dealer having one face up and one face down card, while  
54     player having two face up cards. (Virtually for all Blackjack games today).  
55  
56     The player can request additional cards (hit=1) until they decide to stop  
57     (stick=0) or exceed 21 (bust).  
58  
59     After the player sticks, the dealer reveals their facedown card, and draws  
60     until their sum is 17 or greater. If the dealer goes bust the player wins.  
61  
62     If neither player nor dealer busts, the outcome (win, lose, draw) is  
63     decided by whose sum is closer to 21. The reward for winning is +1,  
64     drawing is 0, and losing is -1.  
65  
66     The observation of a 3-tuple of: the players current sum,  
67     the dealers one showing card (1-10 where 1 is ace),  
68     and whether or not the player holds a usable ace (0 or 1).  
69  
70     This environment corresponds to the version of the blackjack problem  
71     described in Example 5.1 in Reinforcement Learning: An Introduction  
72     by Sutton and Barto.  
73     http://incompleteideas.net/book/the-book-2nd.html  
74     ...  
75  
76     def __init__(self, natural=False, sab=False):  
77         self.action_space = spaces.Discrete(2)  
78         self.observation_space = spaces.Tuple(  
79             (spaces.Discrete(32), spaces.Discrete(11), spaces.Discrete(2)))  
80     )  
81     self.seed()  
82  
83     # Flag to payout 1.5 on a "natural" blackjack win, like casino rules  
84     # Ref: http://www.bicyclecards.com/how-to-play  
85     self.natural = natural
```

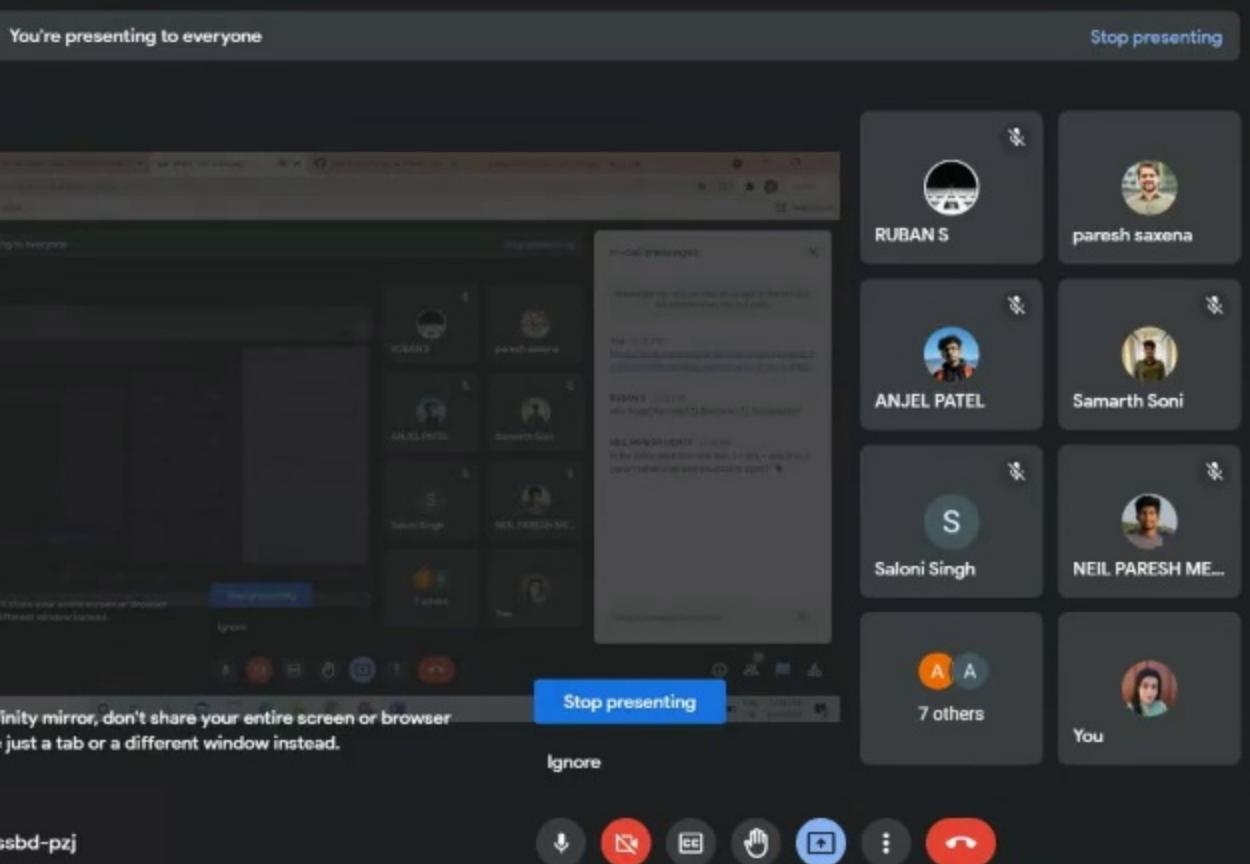
meet.google.com is sharing your screen. Stop sharing Hide

Blackjack using On Policy Monte Carlo

Saving failed since 12:19 PM

```
servation_space = Discrete(32), Discrete(11), Discrete(2))  
tion_space.n  
  
set()  
, False)  
ep(0)  
(8, False), -1.0, True, {})  
ep(1)  
(8, False), -1.0, True, {})  
aler  
]  
[  
ailed. This file was updated remotely or in another tab. Show diff meet.google.com is sharing your screen. Stop sharing Hide
```

[meet.google.com/brc-ssbd-pzj?authuser=0&pli=1](https://meet.google.com/brc-ssbd-pzj?authuser=0&pli=1)



## In-call messages

Messages can only be seen by people who are deleted when the contact is removed.

You 12:03 PM  
<https://colab.research.google.com>

RUBAN S 12:20 PM

NEIL PARESH MEHTA 12:38 PM  
In the policy algorithm why was it T  
cannot recall what was discussed in

Send a message to everyone

policy Monte Carlo - RL class - p20190504@hyderabad - Meet - brc-ssbd-pj - gym/blackjack.py at master · opencog/gym · Blackjack using On Policy Monte Carlo

YouTube Maps

## Blackjack using On Policy Monte Carlo

Saving failed since 12:19 PM

RAM Disk

```

 $\pi^*$  ← arbitrary
 $ns(s, a) \leftarrow$  empty list
 $\pi$  an arbitrary  $\varepsilon$ -soft policy

forever:
    generate an episode using  $\pi$ 
    for each pair  $s, a$  appearing in the episode:
         $R \leftarrow$  return following the first occurrence of  $s, a$ 
        Append  $R$  to  $Returns(s, a)$ 
         $Q(s, a) \leftarrow$  average( $Returns(s, a)$ )
    for each  $s$  in the episode:
         $a^* \leftarrow \arg \max_a Q(s, a)$ 
        For all  $a \in \mathcal{A}(s)$ :
             $\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$ 

```

policy and state value function for blackjack

Google Chrome

You're presenting to [redacted]  
Click here to return to [redacted]  
when you're ready to meet.google.com

This file was updated remotely or in another tab. Show diff

0s completed at 12:32 PM

29°C Rain

meet.google.com/grc-ssbd-pzj?authuser=0&pli=1

[YouTube](#) [Maps](#)

You're presenting to everyone

Stop presenting

infinity mirror, don't share your entire screen or browser; just a tab or a different window instead.

## Stop presenting

ignore



## In-call messages

Messages can only be seen by people who are deleted when the contact is removed.

You 12:03 PM  
<https://colab.research.google.com>

RUBAN S 12:20 PM

NEIL PARESH MEHTA 12:38 PM  
In the policy algorithm why was it T  
cannot recall what was discussed in

Send a message to everyone



YouTube Maps

## Blackjack using On Policy Monte Carlo

File Insert Runtime Tools Help Saving failed since 12:19 PM

Comment



```
ext

= defaultdict(lambda: np.zeros(env.action_space.n))

The policy we're following
l = create_epsilon_greedy_action_policy(env,Q,epsilon)

for i in range(l, episodes + 1):
    # Print out which episode we're on
    if i% 1000 == 0:
        print("\rEpisode {}/{}.".format(i, episodes), end="")
        clear_output(wait=True)

    # Generate an episode.
    # An episode is an array of (state, action, reward) tuples
    episode = []
    state = env.reset()
    for t in range(100):
        probs = pol(state)
        action = np.random.choice(np.arange(len(probs)), p=probs)
        next_state, reward, done, _ = env.step(action)
        episode.append((state, action, reward))
        if done:
            break
        state = next_state

    # Find all (state, action) pairs we've visited in this episode
    # We convert each state to a tuple so that we can use it as a dict key
    sa_in_episode = set([(tuple(x[0]), x[1]) for x in episode])
    for state, action in sa_in_episode:
        sa_pair = (state, action)
        if
```

I

Saving failed. This file was updated remotely or in another tab. Show diff

in the episode

✓ 0s completed at 12:32 PM

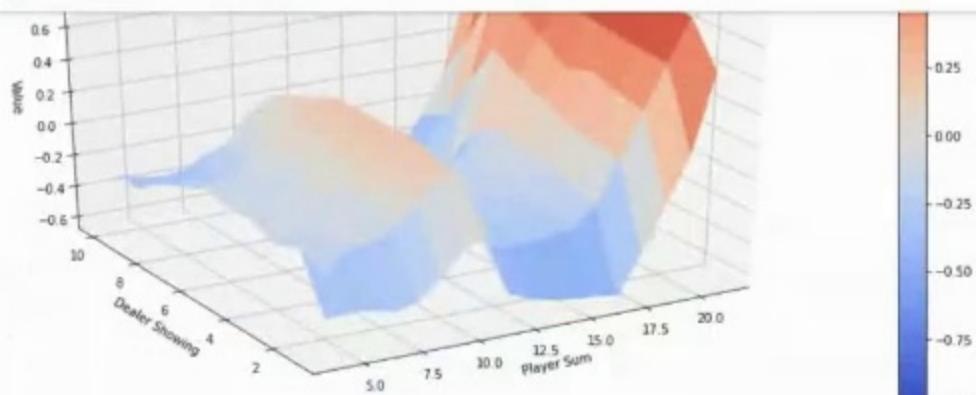
## Blackjack using On Policy Monte Carlo

View Insert Runtime Tools Help Saving failed since 12:19 PM

Comment



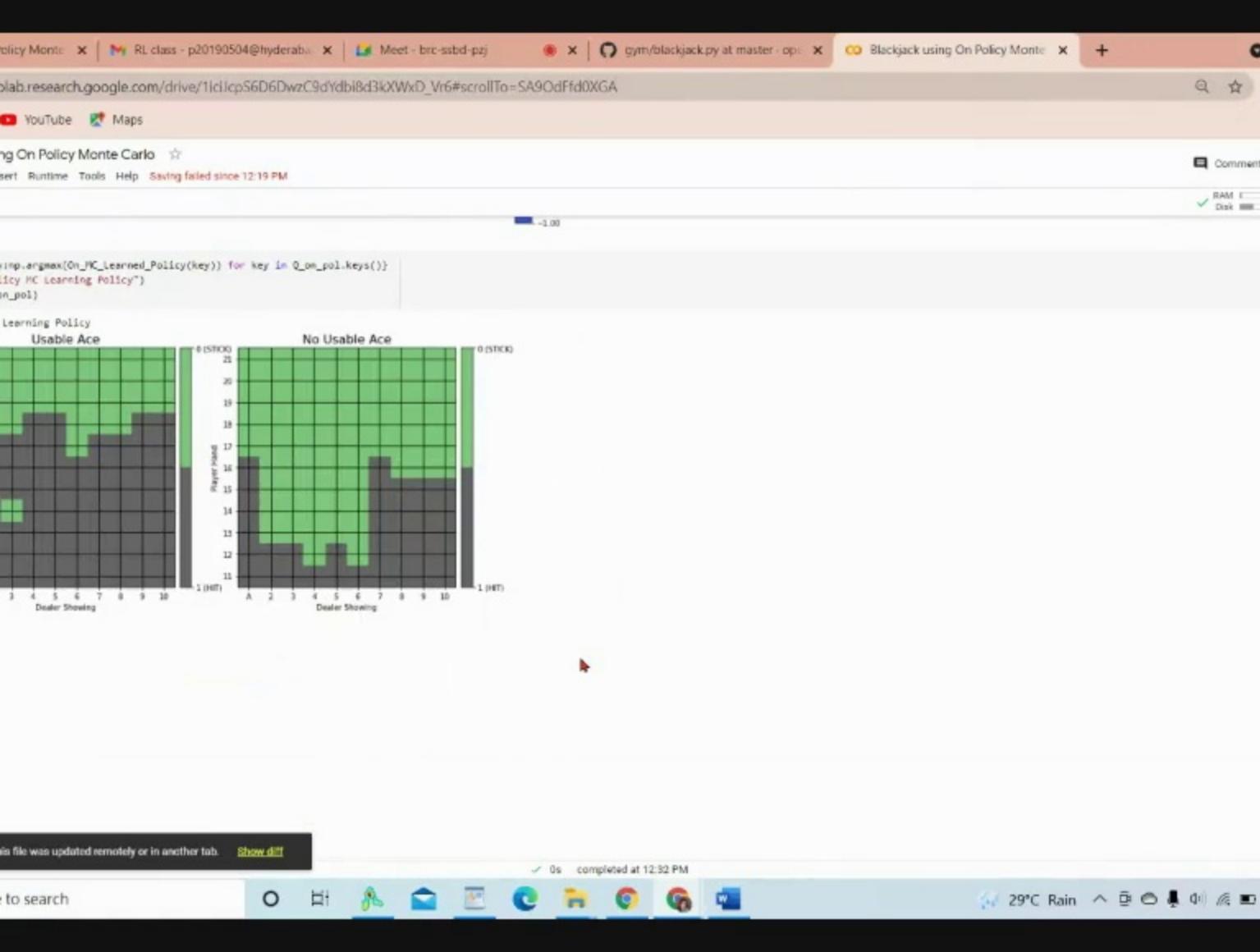
ext



Optimal Value Function for On Policy Learning (Usable Ace)



This file was updated remotely or in another tab. Show diff ✓ 0s completed at 12:32 PM



Policy Monte X RL class - p20190504@hyderabad X Meet - brc-ssbd-pzj X gym/blackjack.py at master · op: X Blackjack using On Policy Monte X +

meet.google.com/brc-ssbd-pzj?authuser=0&pli=1

YouTube Maps

You're presenting to everyone Stop presenting

In-call messages

You 12:03 PM [https://colab.research.google.com/6DwzC9dYdbiRd3kXWx0\\_Vr6#scrim7](https://colab.research.google.com/6DwzC9dYdbiRd3kXWx0_Vr6#scrim7)

RUBAN S 12:20 PM why Tuple(Discrete(32), Discrete(1))

NEIL PARESH MEHTA 12:38 PM In the policy algorithm why was it 1 cannot recall what was discussed in

NEIL PARESH MEHTA 12:40 PM yes sir I will check that out as well ok sir

ANJEL PATEL 12:44 PM Unrelated, but ma'am your notebook saved. We can't refer to the code

Send a message to everyone

has left the meeting

your entire screen or browser window instead.

Stop presenting

Ignore

ssbd-pzj

O D G E F C G W 29°C Rain

```
51     Aces can either count as 11 or 1, and it's called 'usable' at 11.  
52     This game is played with an infinite deck (or with replacement).  
53     The game starts with dealer having one face up and one face down card, while  
54     player having two face up cards. (Virtually for all Blackjack games today).  
55  
56     The player can request additional cards (hit=1) until they decide to stop  
57     (stick=0) or exceed 21 (bust).  
58  
59     After the player sticks, the dealer reveals their facedown card, and draws  
60     until their sum is 17 or greater. If the dealer goes bust the player wins.  
61  
62     If neither player nor dealer busts, the outcome (win, lose, draw) is  
63     decided by whose sum is closer to 21. The reward for winning is +1,  
64     drawing is 0, and losing is -1.  
65  
66     The observation of a 3-tuple of: the players current sum,  
67     the dealers one showing card (1-10 where 1 is ace),  
68     and whether or not the player holds a usable ace (0 or 1).  
69  
70     This environment corresponds to the version of the blackjack problem  
71     described in Example 5.1 in Reinforcement Learning: An Introduction  
72     by Sutton and Barto.  
73     http://incompleteideas.net/book/the-book-2nd.html  
74     ...  
75  
76     def __init__(self, natural=False, sab=False):  
77         self.action_space = spaces.Discrete(2)  
78         self.observation_space = spaces.Tuple(  
79             (spaces.Discrete(32), spaces.Discrete(11), spaces.Discrete(2)))  
80     )  
81     self.reset()  
82  
83     # Flag to payout 1.5 on a "natural" blackjack win, like casino rules  
84     # Ref: http://www.bicyclecards.com/how-to-play/blackjack/  
85     self.natural = natural  
86
```

YouTube Maps

## On Policy Monte Carlo

File Runtime Tools Help All changes saved

Comment

RAM 1 GB

```

    id all (state, action) pairs we've visited in this episode
    convert each state to a tuple so that we can use it as a dict key
    _episode = set([(tuple(x[0]), x[1]) for x in episode])
    state, action in ss_in_episode:
        ss_pair = (state, action)

    # First Visit MC:
    # Find the first occurrence of the (state, action) pair in the episode
    first_occurrence_idx = next(i for i,x in enumerate(episode)
                                 if x[0] == state and x[1] == action)
    # Sum up all rewards since the first occurrence
    S = sum([x[2]*(discount_factor**i) for i,x in enumerate(episode[first_occurrence_idx:])])
    # Calculate average return for this state over all sampled episodes
    returns_sum[ss_pair] += S
    returns_count[ss_pair] += 1.0
    Q[state][action] = returns_sum[ss_pair] / returns_count[ss_pair]

```

pol

mc('blackjack-v0')

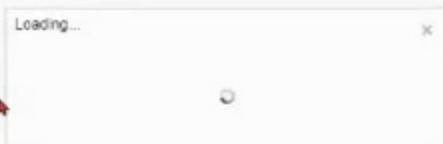
C\_Learned\_Policy = On\_policy\_mc\_control\_learn(env, 500000, 0.9, 0.05)

50/500000.

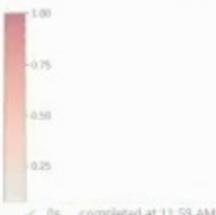
```

    def(float)
    actions in Q_on_policy.items():
        v_low = np.min(actions)
        v_high = np.max(actions)
        action_value = (v_low + v_high)/2
    return(V, title="Optimal Value Function for On-Policy Learning")

```



Optimal Value Function for On-Policy Learning (No Usable Ace)



rc-ssbd-pzi?authuser=0&coll=1

Maps

ng to everyone

Stop presenting

## In-call messages

NEIL PARESH MEHTA 12:38 PM  
In the policy algorithm why was it 1  
cannot recall what was discussed in

NEIL PARESH MEHTA 12:40 PM  
yes sir I will check that out as well  
ok sir

**ANJEL PATEL** 12:44 PM  
Unrelated, but ma'am your notebook  
saved. We can't refer to the code

RUBAN S 12:44 PM  
ma'am you have kept it open in two  
maybe that's why

Ankita Behera 12:45 PM  
it says saving failed on the top  
on your notebook

livity mirror, don't share your entire screen or browser; just a tab or a different window instead.

## Stop presenting

ignore



### Share with people and groups

No one has been added yet

 Get link

[https://colab.research.google.com/drive/1bgJHmH0aqFXYMIUf2n4RvPT\\_5cIX...](https://colab.research.google.com/drive/1bgJHmH0aqFXYMIUf2n4RvPT_5cIX...) Copy link

BITS Pilani University

1

RL class - p20190504@hyderabad | Meet - brc-ssbd-pzj | gym/blackjack.py at master · opencog/gym | Blackjack using On Policy Monte Carlo Control · GitHub

meet.google.com/brc-ssbd-pzj?authuser=0&pli=1

You're presenting to everyone Stop presenting

In-call messages

ANJEL PATEL 12:44 PM Unrelated, but ma'am your notebook saved. We can't refer to the code.

RUBAN S 12:44 PM ma'am you have kept it open in two maybe that's why

Ankita Behera 12:45 PM it says saving failed on the top on your notebook

Aditya Chopra 12:46 PM "The observation of a 3-tuple of: the sum, the dealer's one showing card (1 ace), and whether or not the player holds (0 or 1)." [https://colab.research.google.com/HmH0agFXYMUf2n4RvP7\\_tciXnxTo=H7pZWlg-0EJR](https://colab.research.google.com/HmH0agFXYMUf2n4RvP7_tciXnxTo=H7pZWlg-0EJR)

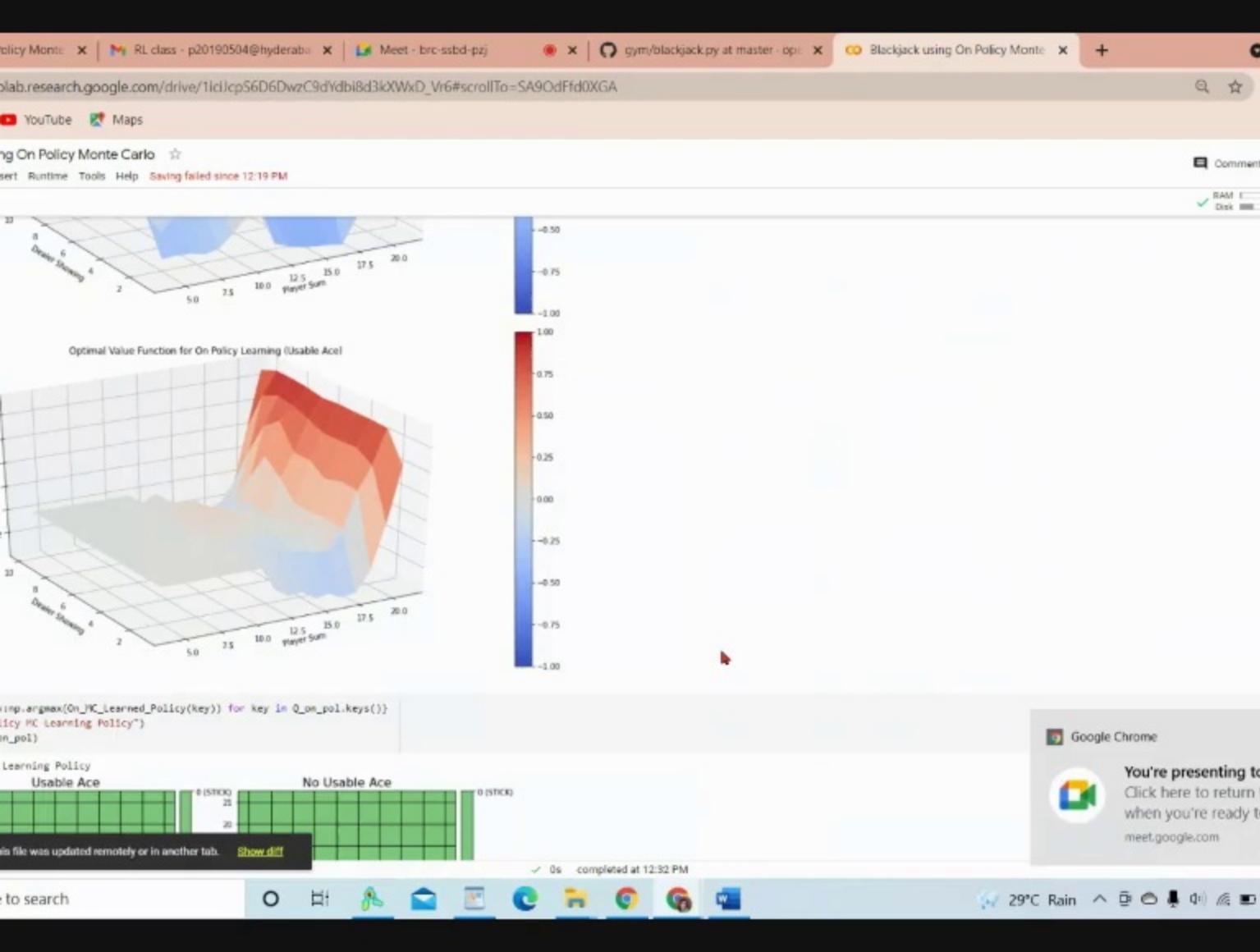
unity mirror, don't share your entire screen or browser just a tab or a different window instead.

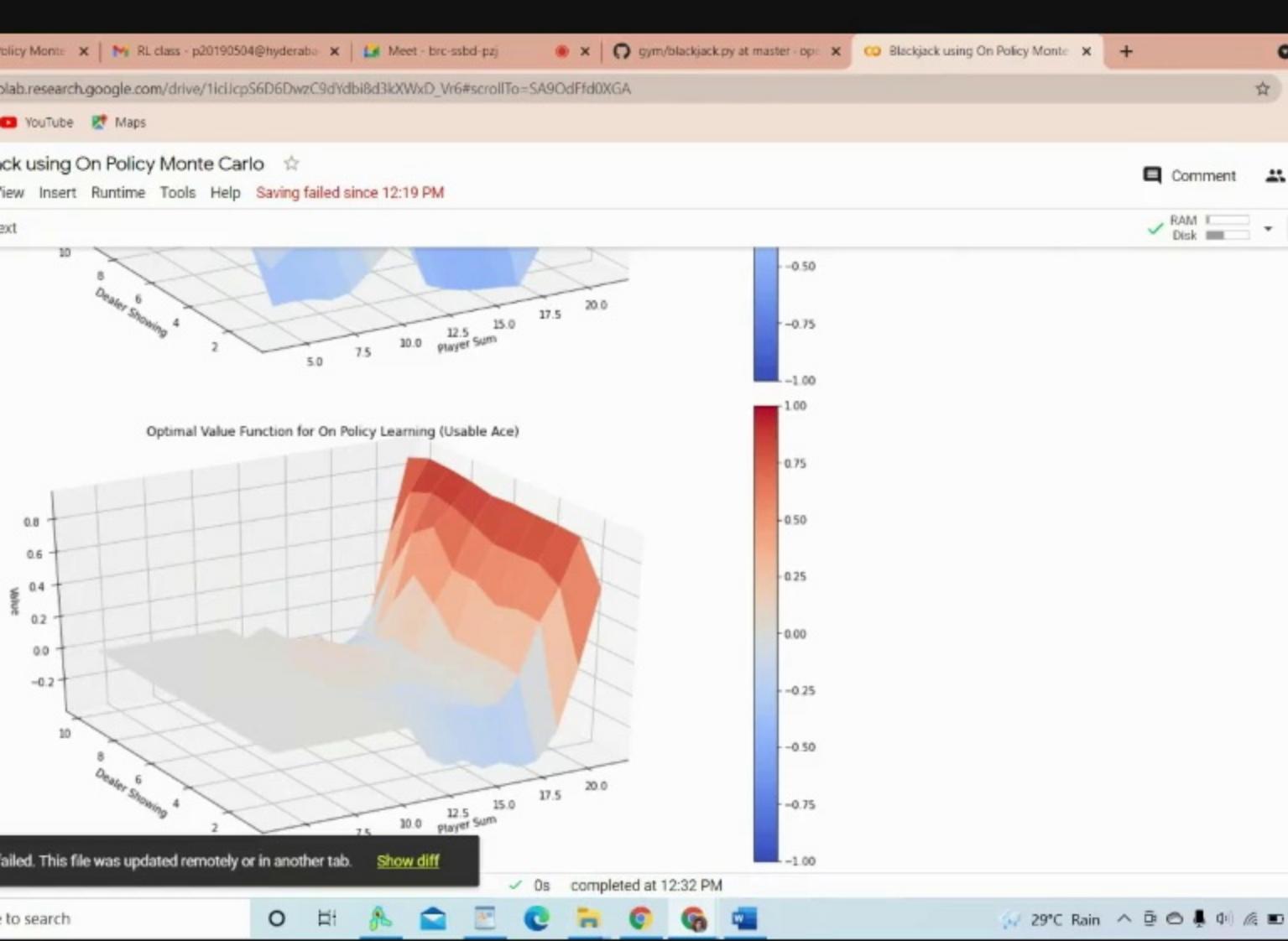
Stop presenting

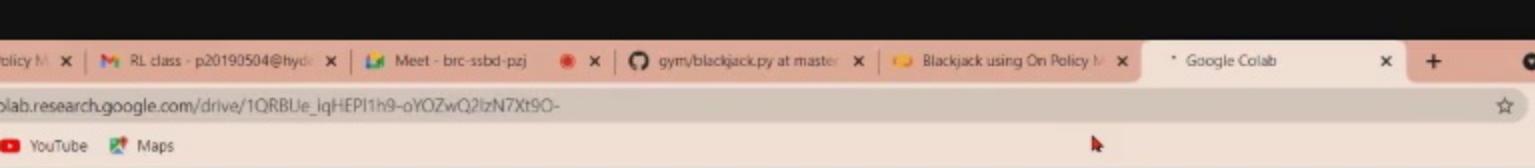
Ignore

ssbd-pzj

29°C Rain





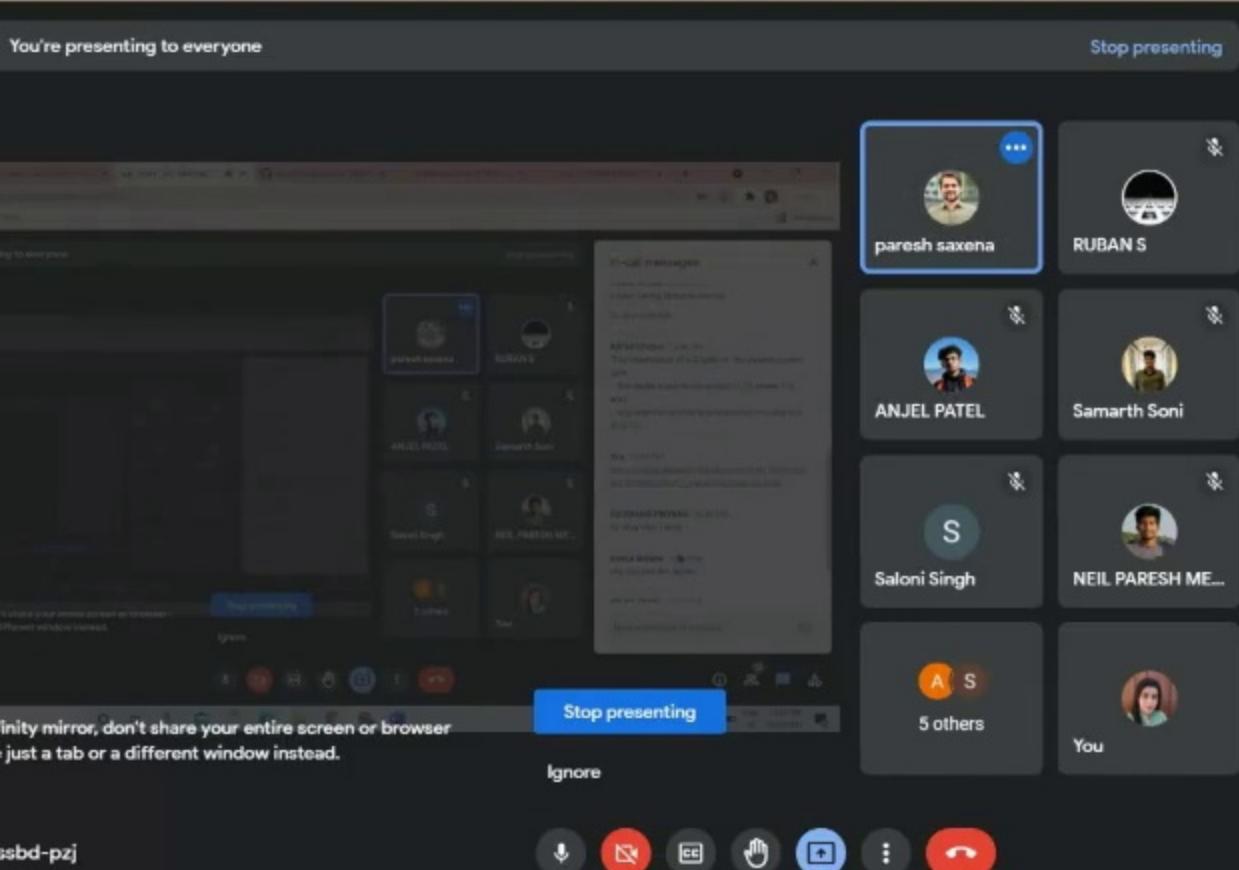


to search



29°C Rain

meet.google.com/brc-ssbd-pzj?authuser=0&pli=1



## In-call messages

it says saving failed on the top  
on your notebook

Aditya Chopra 12:46 PM  
"The observation of a 3-tuple of the sum,  
the dealer's one showing card (1 ace),  
and whether or not the player holds (0 or 1)."

You 12:46 PM  
<https://colab.research.google.com/>

SHUBHAM PRIYANK 12:46 PM  
its okay now i think.

Ankita Behera 12:47 PM  
yes, the last link works

AMICI D'ATTELLA

1996-1997

Send a message to everyone

Send a message to everyone

— 1 —

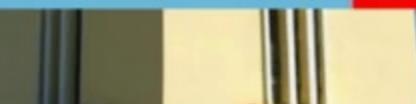




**BITS** Pilani  
Hyderabad Campus

## Reinforcement Learning (CS F317)

Dr. Paresh Saxena  
Dpt. of Computer Science & Information Systems  
Email: psaxena@hyderabad.bits-pilani.ac.in



## **Project:**



## **Next Steps – Implementation and Improvement**

---

4. Out of 8-10 papers, pick and implement one paper that is most relevant for you. One group will implement one paper.
  5. Propose idea to improve the work done in (4).
  6. Implement (5) and show/demonstrate the results.
- Submit a report (4-5) pages on (4), (5) and (6) to p20190504@hyderabad.bits-pilani.ac.in by Friday, 26<sup>th</sup> Nov 2021, 17:00. (Details on reports/presentation/viva will be discussed later.)

Dr. PARESH S. SAXENA  
(BITS Pilani)  
psaxena@hyderabad.bits-pilani.ac.in

# Temporal-Difference Learning

- Waiting only until the next time stamp:

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

TD(0) or one-step TD

## Tabular TD(0) for estimating $v_\pi$

Input: the policy  $\pi$  to be evaluated

Algorithm parameter: step size  $\alpha \in (0, 1]$

Initialize  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

$A \leftarrow$  action given by  $\pi$  for  $S$

        Take action  $A$ , observe  $R, S'$

$V(S) \leftarrow V(S) + \alpha [R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

    until  $S$  is terminal

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t).$$



$$\begin{aligned}
 v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t=s] && \xrightarrow{\hspace{1cm}} \text{MC} \\
 &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t=s] && \xrightarrow{\hspace{1cm}} \text{DP} \\
 &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t=s]. && \xrightarrow{\hspace{1cm}} \text{TD}
 \end{aligned}$$

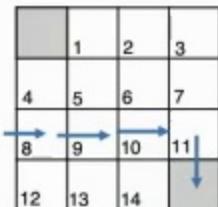
# TD Error

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t).$$

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma \delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2})) \\ &= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} + \cdots + \gamma^{T-t-1} \delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) \\ &= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} + \cdots + \gamma^{T-t-1} \delta_{T-1} + \gamma^{T-t}(0 - 0) \\ &\stackrel{\text{I}}{=} \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k. \end{aligned}$$

# Episode 2

$$\alpha = 0.9, \gamma = 1, R = -1$$



$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

$$\text{At } t=1, V(8) = V(8) + \alpha[R + \gamma V(9) - V(8)] = -0.9,$$

$$\text{At } t=2, V(9) = V(9) + \alpha[R + \gamma V(10) - V(9)] = -0.9,$$

$$\begin{aligned} \text{At } t=3, V(10) &= V(10) + \alpha[R + \gamma V(11) - V(10)] = \\ &= 0 + 0.9[-1 - 0.9] = 0.9[-1.9] = -1.71 \end{aligned}$$

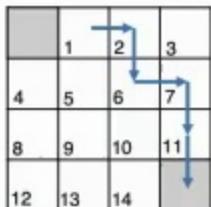
$$\begin{aligned} \text{At } t=4, V(11) &= V(11) + \alpha[R + \gamma V(12) - V(11)] = \\ &= 0 + 0.9[-1 + 0 - (-0.9)] = -0.09 \end{aligned}$$

Already, a higher value function for state 11

States (s)	Ep 1	Ep 2
1	-0.9	-0.9
2	-0.9	-0.9
3	0	0
4	0	0
5	0	0
6	-0.9	-0.9
7	-0.9	-0.9
8	0	-0.9
9	0	-0.9
10	0	0
11	-0.9	-0.09
12	0	0
13	0	0
14	0	0

# Episode 1

$$\alpha = 0.9, \gamma = 1, R = -1$$



$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

At t=2,  $V(2) = V(2) + \alpha[R + \gamma V(6) - V(2)] = -0.9$ ,

At t=3,  $V(6) = V(6) + \alpha[R + \gamma V(7) - V(6)] = -0.9$ ,

At t=4,  $V(7) = V(7) + \alpha[R + \gamma V(11) - V(7)] = -0.9$

At t=5,  $V(11) = V(11) + \alpha[R + \gamma V(T) - V(11)] = -0.9$



States (s)		
1	0	-0.9
2	0	-0.9
3	0	0
4	0	0
5	0	0
6	0	-0.9
7	0	-0.9
8	0	0
9	0	0
10	0	0
11	0	-0.9
12	0	0
13	0	0
14	0	0

# Assignment

---

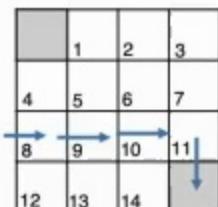


- Continue from Episode 2 from the previous slide.
- Generate a few more episodes.
- Check if TD converges.

# Episode 2



$$\alpha = 0.9, \gamma = 1, R = -1$$



$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

At t=1,  $V(8) = V(8) + \alpha[R + \gamma V(9) - V(8)] = -0.9$ ,

At t=2,  $V(9) = V(9) + \alpha[R + \gamma V(10) - V(9)] = -0.9$ ,

i) At t=3,  $V(10) = V(10) + \alpha[R + \gamma V(11) - V(10)] =$   
 $= 0 + 0.9[-1 - 0.9] = 0.9[-1.9] = -1.71$

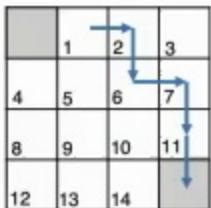
At t=4,  $V(11) = V(11) + \alpha[R + \gamma V(12) - V(11)] =$   
 $= 0 + 0.9[-1 + 0 - (-0.9)] = -0.09$

Already, a higher value function for state 11

States (s)	Ep 1	Ep 2
1	-0.9	-0.9
2	-0.9	-0.9
3	0	0
4	0	0
5	0	0
6	-0.9	-0.9
7	-0.9	-0.9
8	0	-0.9
9	0	-0.9
10	0	0
11	-0.9	-0.09
12	0	0
13	0	0
14	0	0

# Episode 1

$$\alpha = 0.9, \gamma = 1, R = -1$$



$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

At t=2,  $V(2) = V(2) + \alpha[R + \gamma V(6) - V(2)] = -0.9$ ,

At t=3,  $V(6) = V(6) + \alpha[R + \gamma V(7) - V(6)] = -0.9$ ,

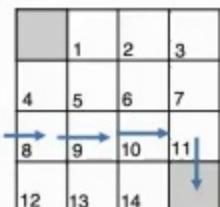
At t=4,  $V(7) = V(7) + \alpha[R + \gamma V(11) - V(7)] = -0.9$

At t=5,  $V(11) = V(11) + \alpha[R + \gamma V(14) - V(11)] = -0.9$

States (s)		
1	0	-0.9
2	0	-0.9
3	0	0
4	0	0
5	0	0
6	0	-0.9
7	0	-0.9
8	0	0
9	0	0
10	0	0
11	0	-0.9
12	0	0
13	0	0
14	0	0

# Episode 2

$$\alpha = 0.9, \gamma = 1, R = -1$$



$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

At t=1,  $V(8) = V(8) + \alpha[R + \gamma V(9) - V(8)] = -0.9$ ,

At t=2,  $V(9) = V(9) + \alpha[R + \gamma V(10) - V(9)] = -0.9$ ,

At t=3,  $V(10) = V(10) + \alpha[R + \gamma V(11) - V(10)] =$   
 $= 0 + 0.9[-1 - 0.9] = 0.9[-1.9] = -1.71$

At t=4,  $V(11) = V(11) + \alpha[R + \gamma V(12) - V(11)] =$   
 $= 0 + 0.9[-1 + 0 - (-0.9)] = -0.09$

Already, a higher value function for state 11



States (s)	Ep 1	Ep 2
1	-0.9	-0.9
2	-0.9	-0.9
3	0	0
4	0	0
5	0	0
6	-0.9	-0.9
7	-0.9	-0.9
8	0	-0.9
9	0	-0.9
10	0	0
11	-0.9	-0.09
12	0	0
13	0	0
14	0	0

# TD Error

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t).$$

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma \delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2})) \\ &= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} + \cdots + \gamma^{T-t-1} \delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) \\ &= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} + \cdots + \gamma^{T-t-1} \delta_{T-1} + \gamma^{T-t}(0 - 0) \\ &= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k. \end{aligned}$$

©



# Reinforcement Learning (CS F317)



**BITS** Pilani  
Hyderabad Campus

Dr. Paresh Saxena  
Dpt. of Computer Science & Information Systems  
Email: psaxena@hyderabad.bits-pilani.ac.in



# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S,A)$

```

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  $\pi$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ 
     $G \leftarrow 0$ 
    Loop for each step of episode,  $t = T-1, T-2, \dots, 0$ :
         $G \leftarrow \gamma G + R_{t+1}$ 
        Unless the pair  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ :
            Append  $G$  to  $Returns[S_t, A_t]$ 
             $Q(S_t, A_t) \leftarrow \text{average}(Returns[S_t, A_t])$ 
             $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$ 

```

# Monte Carlo: Problem Session

Innovate

achieve

lead

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

After 6 episodes:

$$Q(S|D) = [2.768 + (-7) + 0 + (-1.2) + 4.2 + 5.55] / 5 = 0.8636 \text{ (Divide by 5 since (S,D) has not appeared in the 3rd episode)}$$

$$Q(R|D) = 0$$



# Reinforcement Learning (CS F317)



**BITS** Pilani  
Hyderabad Campus

Dr. Paresh Saxena  
Dpt. of Computer Science & Information Systems  
Email: psaxena@hyderabad.bits-pilani.ac.in

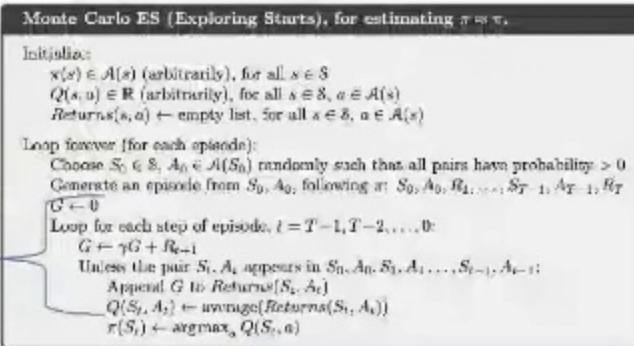


# Monte Carlo: Problem Session

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

Estimate  
 $Q(S,A)$



# Monte Carlo: Problem Session

Innovate

achieve

lead

- States = {Deciding State D, Terminal/Retire State T}
- Actions = {Steal S, Retire R}
- Rewards = {1, 2, 4, -10, 0}
- Six Episodes:
- Episode 1 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,1,D; D,S,2,D; D,S,-10,T
- Episode 2 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,-10,T
- Episode 3 ( $s_t, a_t, r_t, s_{t+1}$ ): D,R,0,T
- Episode 4 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,2,D; D,S,4,D; D,S,-10,T
- Episode 5 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,4,D; D,R,0,T
- Episode 6 ( $s_t, a_t, r_t, s_{t+1}$ ): D,S,1,D; D,S,1,D; D,S,1,D; D,S,4,D; D,S,1,D; D,S,2,D; D,R,0,T

Using first-visit MC and after 6 episodes, find  $Q(S|D)$  and  $Q(R|D)$ .

After 6 episodes:

$$Q(S|D) = [2.768 + (-7) + 0 + (-1.2) + 4.2 + 5.55] / 5 = 0.8636 \text{ (Divide by 5 since (S,D) has not appeared in the 3rd episode)}$$

$$Q(R|D) = 0$$



## Project:

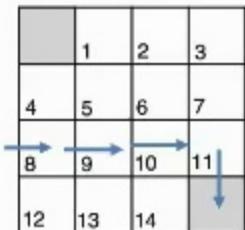
### Next Steps – Implementation and Improvement

---

4. Out of 8-10 papers, pick and Implement one paper that is most relevant for you. One group will implement one paper.
  5. Propose idea to improve the work done in (4).
  6. Implement (5) and show/demonstrate the results.
- Submit a report (4-5) pages on (4), (5) and (6) to p20190504@hyderabad.bits-pilani.ac.in by Friday, 26<sup>th</sup> Nov 2021, 17:00. (Details on reports/presentation/viva will be discussed later.)

# Episode 2

$$\alpha = 0.9, \gamma = 1, R = -1$$



$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

At t=1,  $V(8) = V(8) + \alpha[R + \gamma V(9) - V(8)] = -0.9$ ,

At t=2,  $V(9) = V(9) + \alpha[R + \gamma V(10) - V(9)] = -0.9$ ,

At t=3,  $V(10) = V(10) + \alpha[R + \gamma V(11) - V(10)] =$   
 $= 0 + 0.9[-1 - 0.9] = 0.9[-1.9] = -1.71$

At t=4,  $V(11) = V(11) + \alpha[R + \gamma V(12) - V(11)] =$   
 $= -0.9 + 0.9[-1 + 0 - (-0.9)] = -0.9 - 0.09$   
 $= -0.99$



States (s)	Ep 1	Ep 2
1	-0.9	-0.9
2	-0.9	-0.9
3	0	0
4	0	0
5	0	0
6	-0.9	-0.9
7	-0.9	-0.9
8	0	-0.9
9	0	-0.9
10	0	-1.71
11	-0.9	-0.99
12	0	0
13	0	0
14	0	0

# Markov Reward Process: Random Walk

innovate

achieve

lead

- MRP: MDP without actions



Random Walk: Start from C and go in any direction with  $\frac{1}{2}$  probability

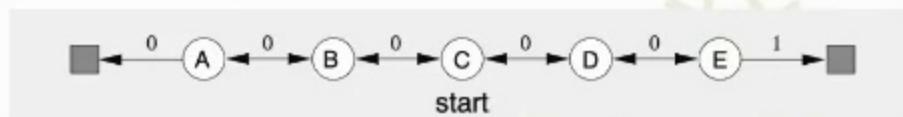
Reward: 1 at extreme right otherwise 0.

What is the true value of  $v(C)$  ?

What are the true values for  $v(A)$ ,  $v(B)$ ,  $v(C)$ ,  $v(D)$  and  $v(E)$  ?

# Random Walk

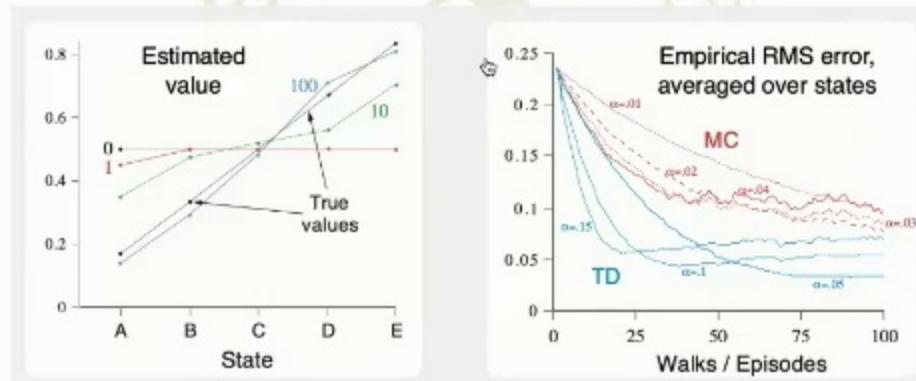
MRP: MDP without actions

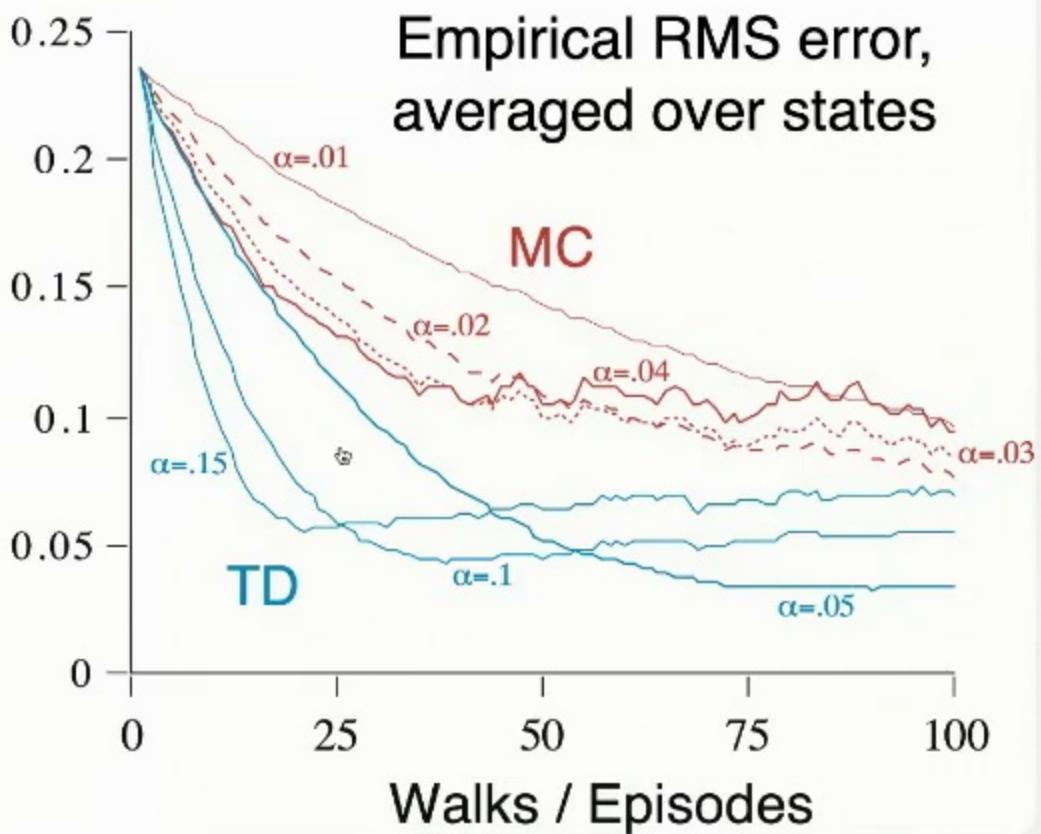


Random Walk: Start from C and in any direction with  $\frac{1}{2}$  probability

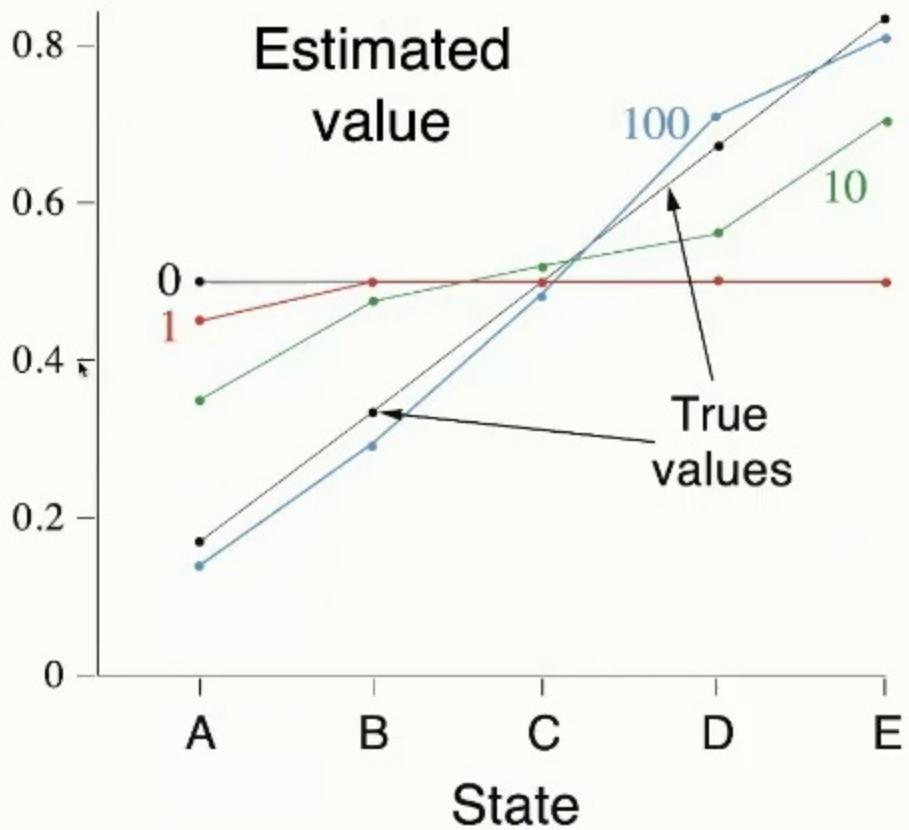
Reward: 1 at extreme right otherwise 0.

$$v(A) = \frac{1}{6}, v(B) = \frac{2}{6}, v(C) = \frac{1}{6}, v(D) = \frac{4}{6}, v(E) = \frac{5}{6}$$

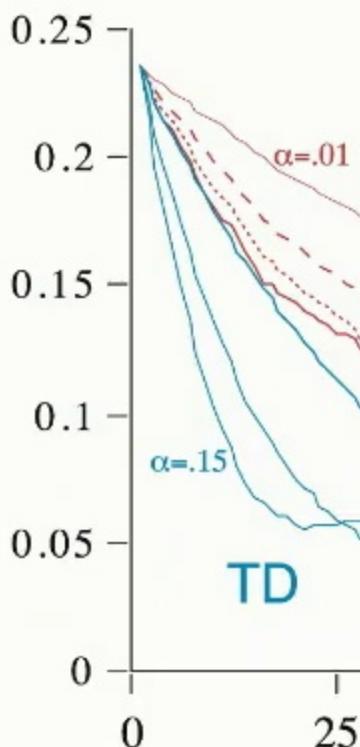




## Estimated value



True  
values



TD

# Markov Reward Process: Random Walk



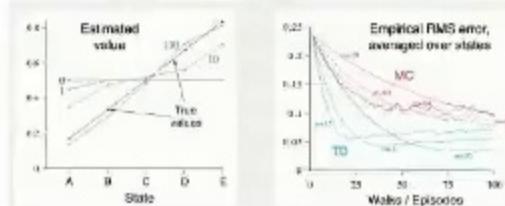
- MRP: MDP without actions



Random Walk: Start from C and go in any direction with ½ probability

Reward: 1 at extreme right otherwise 0.

$$v(A) = 1/6, v(B) = 2/6, v(C) = 3/6, v(D) = 4/6, v(E) = 5/6$$



# Markov Reward Process: Random Walk

innovate

achieve

lead

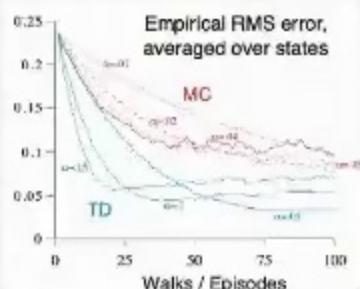
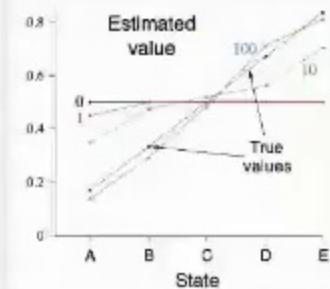
- MRP: MDP without actions



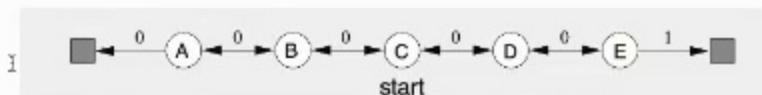
Random Walk: Start from C and go in any direction with  $\frac{1}{2}$  probability

Reward: 1 at extreme right otherwise 0.

$$v(A) = 1/6, v(B) = 2/6, v(C) = 1/6, v(D) = 4/6, v(E) = 5/6$$



# TD: Problem



Consider first episode as:

CB0BA0AT0 where T is the terminal state.

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- Assume that initially,  $V(A) = V(B) = V(C) = V(D) = V(E)$  and  $V(T) = 0$ . Also assume  $\gamma = 1$ . Now after the first episode if  $V(A) = 0.4$ , what is the value of  $\alpha$  ?

# TD: Batch Updates/Training

- Batch Updating: train completely on a finite amount of data, e.g., train repeatedly on 10 episodes until convergence.
- Compute updates according to TD or MC, but only update estimates after each complete pass through the data.
- For any finite Markov prediction task, under batch updating, TD converges for sufficiently small  $\alpha$ .
- Constant- $\alpha$  MC also converges under these conditions, but to a different answer!

# Problem: Unknown Markov Reward Process



## Given

A, 0	B, 0
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

- What is  $V(B)$  and  $V(A)$  ?

Dr. PARESH SAXENA  
(BITS Pilani)  
psaxena@hyderabad.bits-pilani.ac.in

# Problem: Unknown Markov Reward Process

innovate

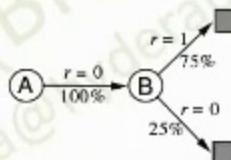
achieve

lead

## Given

A, 0	B, 0
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

- What is  $V(B)$  and  $V(A)$  ?
- $V(B)$  is straightforward 6 out of 8 times we get reward 1, so  $V(B) = 6/8 = \frac{3}{4}$
- What about  $V(A)$  ? (Two possibilities)
  - Since we visit A one time, and reward is 0, we can say  $V(A) = 0$ .
  - This will be the result by using MC batch method.
  - Another Possibility (to consider following):



- in this case  $V(A) = \frac{3}{4}$ . This will be the result by using TD(0) batch.
- This will give error wrt training data. However, it is still better since we expect that it will produce lower error on future data.



# Problem: Unknown Markov Reward Process

innovate

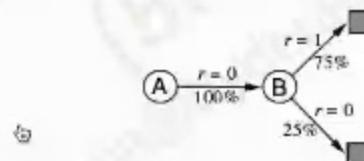
achieve

lead

## Given

A, 0	B, 0
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

- What is  $V(B)$  and  $V(A)$  ?
- $V(B)$  is straightforward 6 out of 8 times we get reward 1, so  $V(B) = 6/8 = \frac{3}{4}$
- What about  $V(A)$  ? (Two possibilities)
  - Since we visit A one time, and reward is 0, we can say  $V(A) = 0$ .
  - This will be the result by using MC batch method.
  - Another Possibility (to consider following):



- in this case  $V(A) = \frac{3}{4}$ . This will be the result by using TD(0) batch.
- This will give error wrt training data. However, it is still better since we expect that it will produce lower error on future data.



# Problem: Unknown Markov Reward Process

innovate

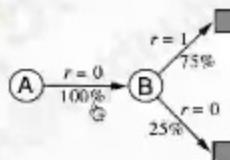
achieve

lead

## Given

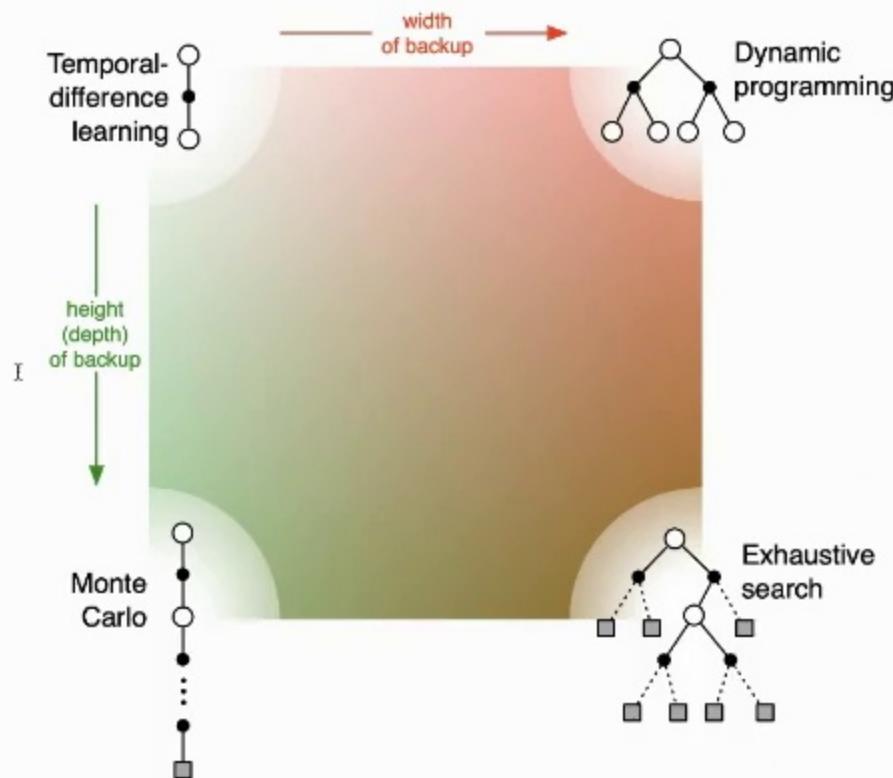
A, 0	B, 0
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

- What is  $V(B)$  and  $V(A)$  ?
- $V(B)$  is straightforward 6 out of 8 times we get reward 1, so  $V(B) = 6/8 = \frac{3}{4}$
- What about  $V(A)$  ? (Two possibilities)
  - Since we visit A one time, and reward is 0, we can say  $V(A) = 0$ .
  - This will be the result by using MC batch method.
  - Another Possibility (to consider following):



- in this case  $V(A) = \frac{3}{4}$ . This will be the result by using TD(0) batch.
- This will give error wrt training data. However, it is still better since we expect that it will produce lower error on future data.

# Unified View



# Problem: Unknown Markov Reward Process

innovate

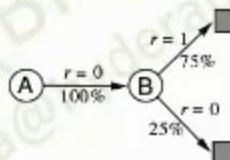
achieve

lead

## Given

A, 0	B, 0
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

- What is  $V(B)$  and  $V(A)$  ?
- $V(B)$  is straightforward 6 out of 8 times we get reward 1, so  $V(B) = 6/8 = \frac{3}{4}$
- What about  $V(A)$  ? (Two possibilities)
  - Since we visit A one time, and reward is 0, we can say  $V(A) = 0$ .
  - This will be the result by using MC batch method.
  - Another Possibility (to consider following):



- in this case  $V(A) = \frac{3}{4}$ . This will be the result by using TD(0) batch.
- This will give error wrt training data. However, it is still better since we expect that it will produce lower error on future data.