

Module 1

CSE3011 Reinforcement Learning

Credit Structure : 2-2-3



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Module 1 : Introduction to RL

Topics : Elements of RL - Agent, environment Interface, Goals and rewards, RL platforms, Applications of RL, Markov decision process (MDP), RL environment as a MDP, Maths essentials of RL, Policy and its types, episodic and continuous tasks, return and discount factor, fundamental functions of RL – value and Q functions, model-based and model-free learning, types of RL environments, Solving MDP using Bellman Equation, Algorithms for optimal policy using Dynamic Programming -Value iteration and policy iteration, Example : Frozen Lake problem, Limitations and Scope.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction to RL

- Reinforcement Learning(RL) is one of the areas of Machine Learning(ML).
- It is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions.
- For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
- It is one of the most active research areas in AI.
- It has evolved and capable of building a recommendation system to self-driving cars.
- Reason for this evolution is deep RL, a combination of DL and RL.
- <https://neptune.ai/blog/reinforcement-learning-applications>



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction to RL..

- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.
- Since there is no labeled data, the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as **game-playing, robotics**, etc.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

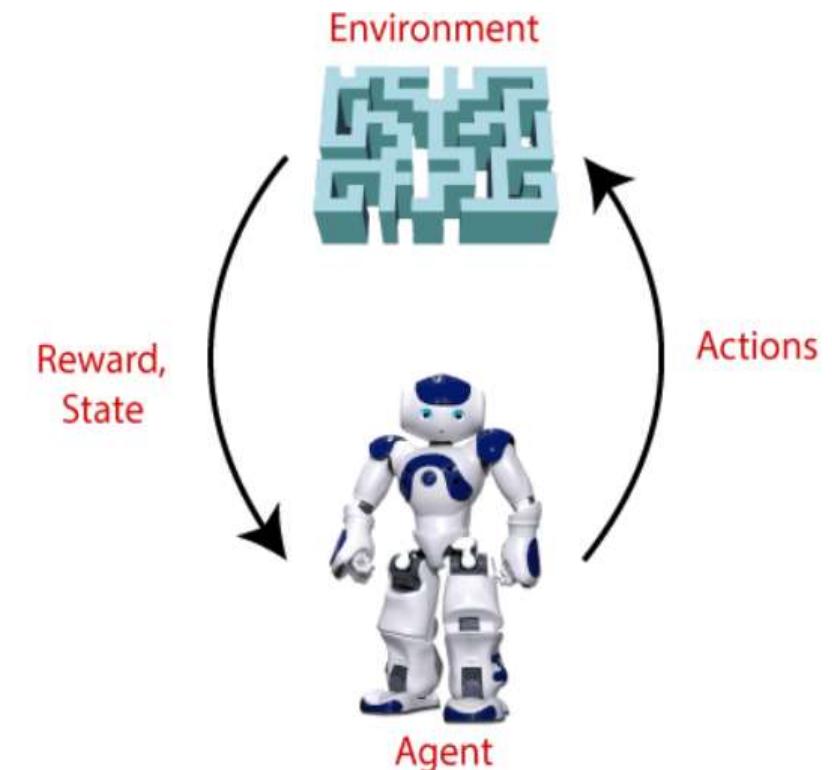


RL agents

- The goal of reinforcement learning is to train an agent to complete a task within an uncertain environment.
- At each time interval, the agent receives observations and a reward from the environment and sends an action to the environment.
- The reward is a measure of how successful the previous action (taken from the previous state) was with respect to completing the task goal.

Example

- **Goal of an AI agent has to find the diamond present within a maze environment**
- The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback for its actions.
- The agent continues doing these three things
(take action, change state/remain in the same state, and get feedback), and by doing these actions, he learns and explores the environment.
- The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty.
As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.



Elements of RL

- L01: Define the key elements of RL
- L02: Explain the steps involved in a typical RL algorithm
- L03: Differentiate the three machine learning paradigms namely, supervised, unsupervised and reinforcement learning.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Elements of RL

1. Agent:

- It is a software that learns to make intelligent decisions.
- In an RL setting, an agent is a learner.
- Ex1: a chess-player is an agent, the player learns to make the best moves (decisions) to win the game.
- Ex2 : Mario in a Super Mario Bros Video Game



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Elements of RL

2. Environment :

- It is the world of the agent, within which the agent stays, takes actions and interacts.
- Ex1: chess-board in a chess game.
- The chess player(agent) stays in the chess board to learn how to play the game.

Elements of RL

3. State and Action:

- In an RL setup, the environment has many positions where the agent can be in.
- Each such position is a state.
- A state is denoted by **s**
- Ex: in a chess-board environment, each position is a state.

Elements of RL

3. State and Action:

- The agent interacts with the environment and moves from one state to another state by performing an action.
- Ex: In a chess-game environment, the action is the move performed by the player(agent).
- An action is denoted by **a**

Elements of RL

4. Reward :

- The agent interacts with the environment by performing an action and moves from one state to another.
- Based on the action the agent receives a reward.
- Reward is a numerical value, ex: +1 for a good action, -1 for a bad action.
- Ex: in a chess-game, **good action** – the agent's move which takes one of the opponent's chess piece; **bad action** – the agent's move loosing one chess piece to the opponent



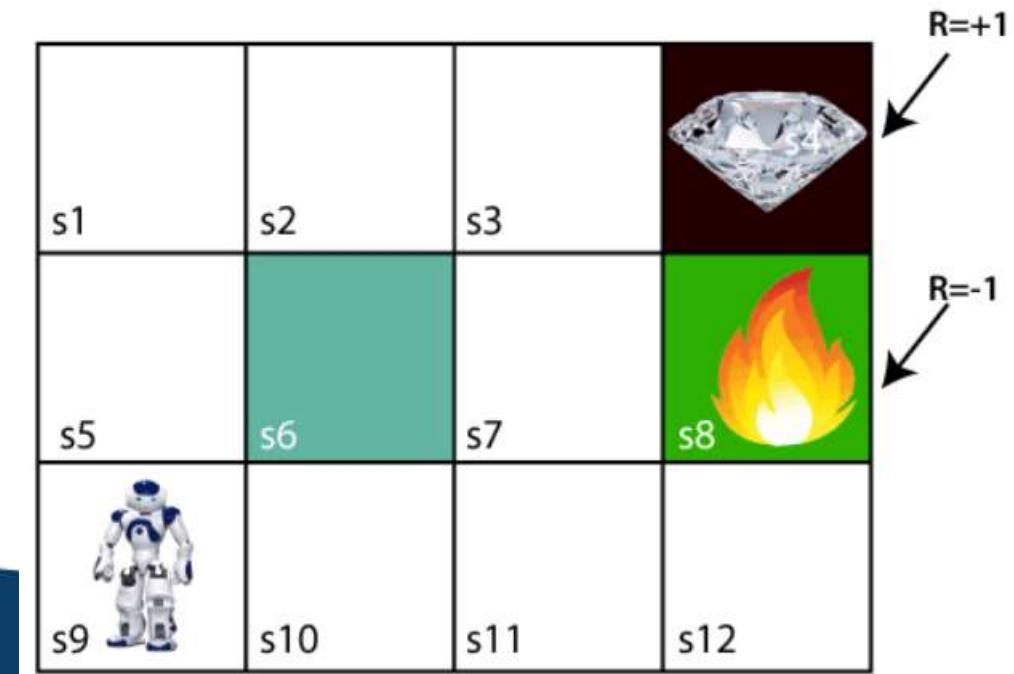
**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Basic idea of RL

- To understand the working process of the RL, we need to consider two main things:
- **Environment:** It can be anything such as a room, maze, football ground, etc.
- **Agent:** An intelligent agent such as AI robot.
- Let's take an example of a maze environment that the goal of the agent is to explore and find the path to the diamond in few steps.
- **States :** S1 to S12, where S6 is a wall, S8 is a fire pit and S4 has diamond.
- **Actions :** move left, right, up and down

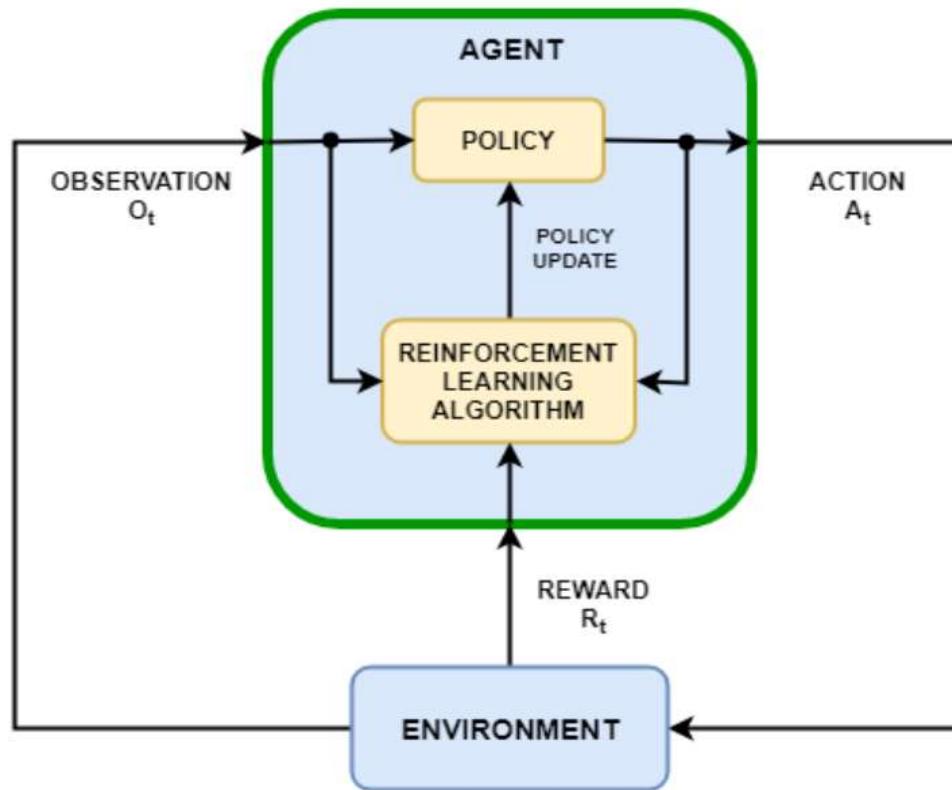


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



A typical RL setup



- The agent has two components : **policy** and the **RL algorithm**.
- **Policy :**
 - It is a mapping from the current environment's observation to a probability distribution of the actions to be taken.
 - Within an agent, the policy is implemented by a function approximator with tunable parameters and a specific approximation model, such as a deep neural network.
- **RL algorithm:**
 - The learning algorithm continuously updates the policy parameters based on the actions, observations, and rewards.
 - The goal of the learning algorithm is to find an optimal policy that maximizes the expected cumulative long-term reward received during the task.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



How RL differs from other ML paradigms?

- **Difference between RL and unsupervised learning?**
- **Task : Movie recommendation system**- recommend a new movie to the user
- Unsupervised learning: the model will recommend a new movie based on the similar movies the user has viewed before.
- In RL, each time the user watches a movie, the agent receives feedback from the user.
- Feedbacks are rewards(ratings given by the user to this movie, time spent watching the movie, etc).
- Based on these rewards, the RL agent will understand the movie preference of the user, then suggests new movies accordingly.

How RL differs from other ML paradigms?

Task : train a dog to catch a ball

- **Difference between RL and Supervised learning**
- Supervised learning, we train the dog explicitly with training data : turn left, go right, move forward seven steps, catch the ball and so on.
- In RL, we simply throw the ball, every time the dog catches the ball, we give it a cookie(reward).
- So the dog will learn to catch the ball while trying to maximize the cookies(rewards) it can get.



How RL differs from other ML paradigms?

- Hence, the supervised and unsupervised models learn from the training data set.
- In RL, the agent learns by continuously interacting with the environment.
- Hence entire RL is about the interaction between the agent and the environment.

A typical RL algorithm

- The steps involved in a typical RL algorithm are:
 1. First, the agent interacts with the environment by performing an action.
 2. By performing an action, the agent moves from one state to another.
 3. Then the agent will receive a reward based on the action it performed.
 4. Based on the reward, the agent will understand whether the action is good or bad.
 5. If the action was good, that is, if the agent received a positive reward, then the agent will prefer performing that action, else the agent will try performing other actions in search of a positive reward.

The goal of the agent is to maximize the reward it gets. If the agent receives a good reward, then it means it has performed a good action. If the agent performs a good action, then it implies that it can win the game. Thus, the agent learns to win the game by maximizing the reward



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



RL agent in the grid world environment

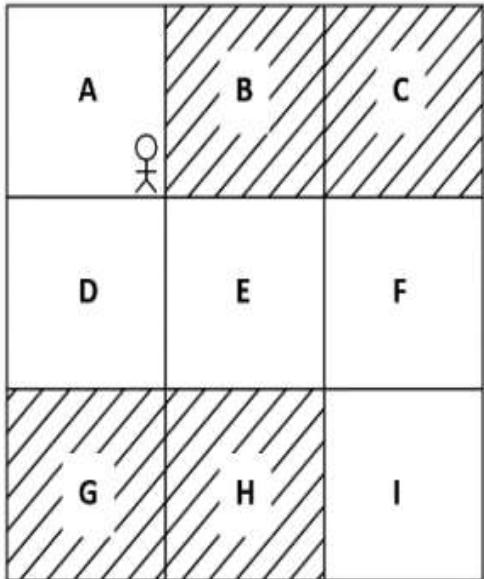


Figure 1.4: Grid world environment

- **Environment :** Grid World environment
- **States :** A,B,C,D,E,F,G,H and I. Shaded states are hole states.
- **Goal of the agent :** reach state I from state A.
- **Actions :** move up, down, left and right.
- Every time the agent reaches one of the shaded states it receives a **negative reward**(-1).
- Every time the agent reaches one of the unshaded states it receives a **positive reward**(+1).
- First time when the agent interacts with the envt (first iteration), it performs a random action in each state, and mostly ends up with negative rewards.
- But, over a series of iterations, it learns to perform the correct action in each state, based on the rewards it has obtained in that state in the previous iterations and hence reaches the goal.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



RL agent in the Grid World Environment

- Iteration 1:

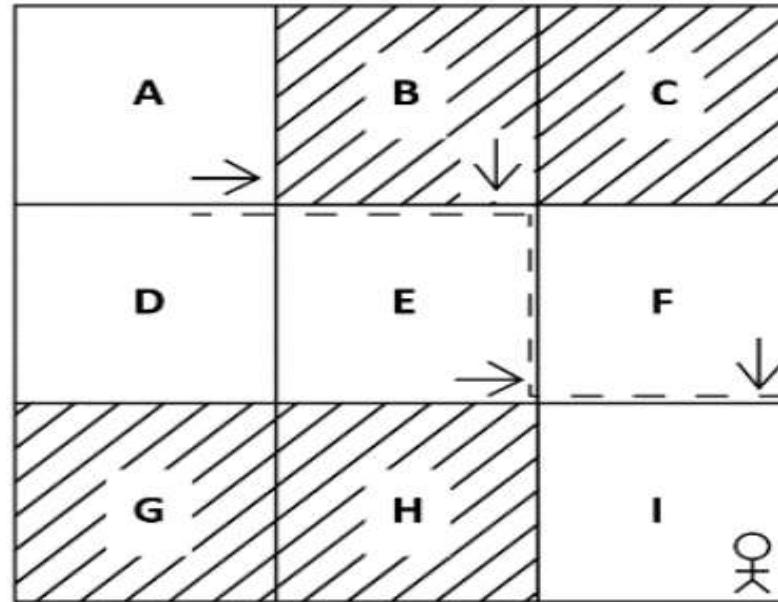


figure 1.5: Actions taken by the agent in iteration 1



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



RL agent in the Grid World Environment

- Iteration 2:

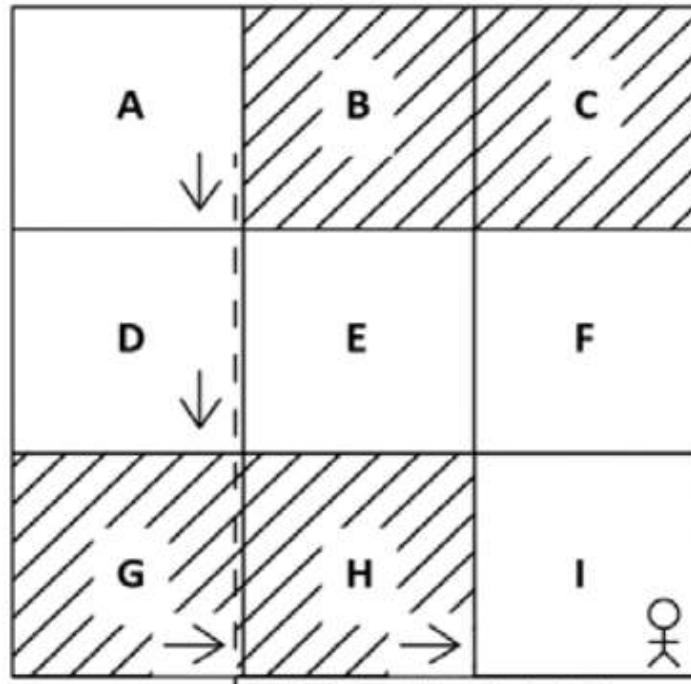


Figure 1.6: Actions taken by the agent in iteration 2

RL agent in the Grid World Environment

- Iteration 3:

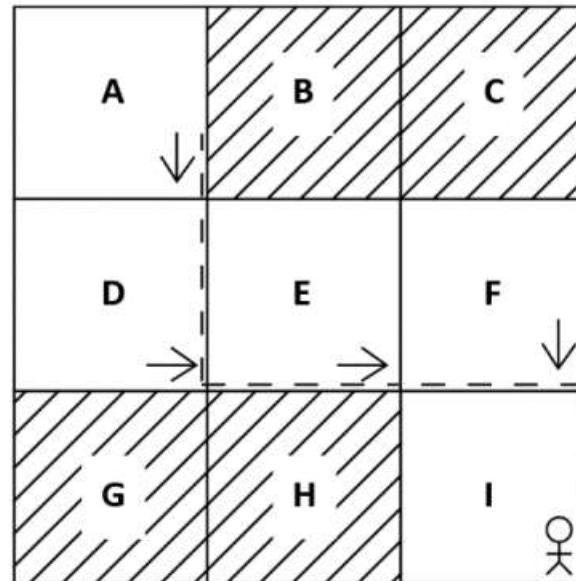


Figure 1.7: Actions taken by the agent in iteration 3



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



RL agent in the Grid World Environment

- Result of Iteration 3, the agent reaches the goal state without reaching the shaded states.
- The agent has successfully learnt to reach the goal state I from state A, without visiting the shaded states based on the rewards.
- The goal of the agent is to maximize the rewards and ultimately achieve the goal
- Each iteration known as an episode in RL terms.

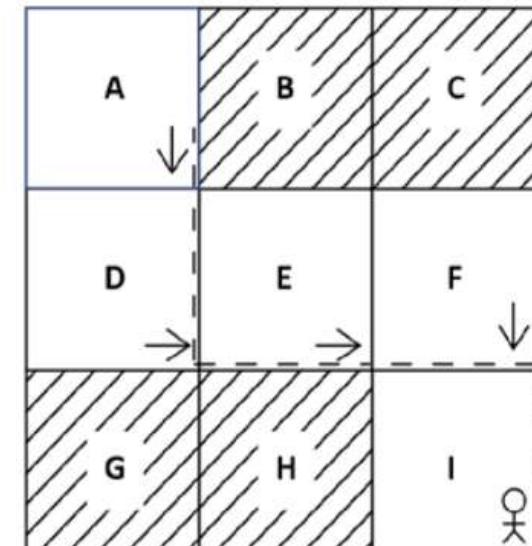


Figure 1.8: The agent reaches the goal state without visiting the shaded states

Types of RL environments

- LO1: List the types of RL environments
- LO2: Differentiate the different types of RL environments
- LO3: Explain the different types of RL environments with an example



Types of RL environments

- **Deterministic environment** : It is certain that, when an agent in state s, performs an action a, then it always reaches state s'.

Ex: **Chess** - there would be only a few possible moves for a coin at the current state and these moves can be determined.

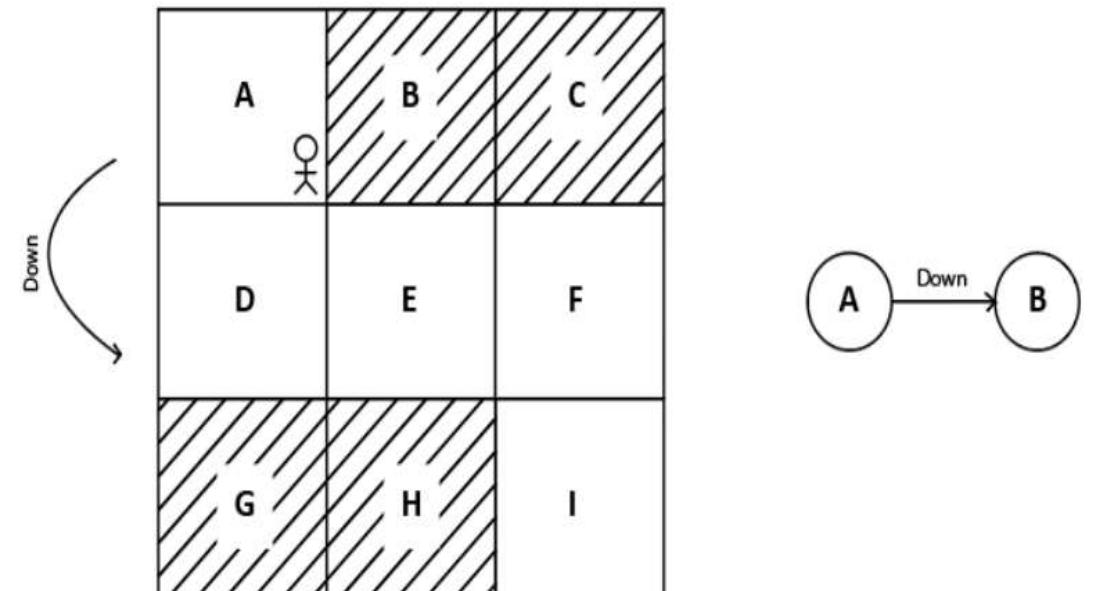


Figure 1.30: Deterministic environment

Types of RL environments

- **Stochastic environment** : When an agent in state s , performs an action a , then we cannot say that, it always reaches state s' .
- We cannot determine the outcome of the action in the current state
- This is due to the randomness in the stochastic environment.

Ex1: **Self-Driving Cars**- the actions of a self-driving car are not unique, it varies time to time.

Ex2: The radio station is a stochastic environment where the listener is not aware about the next song.

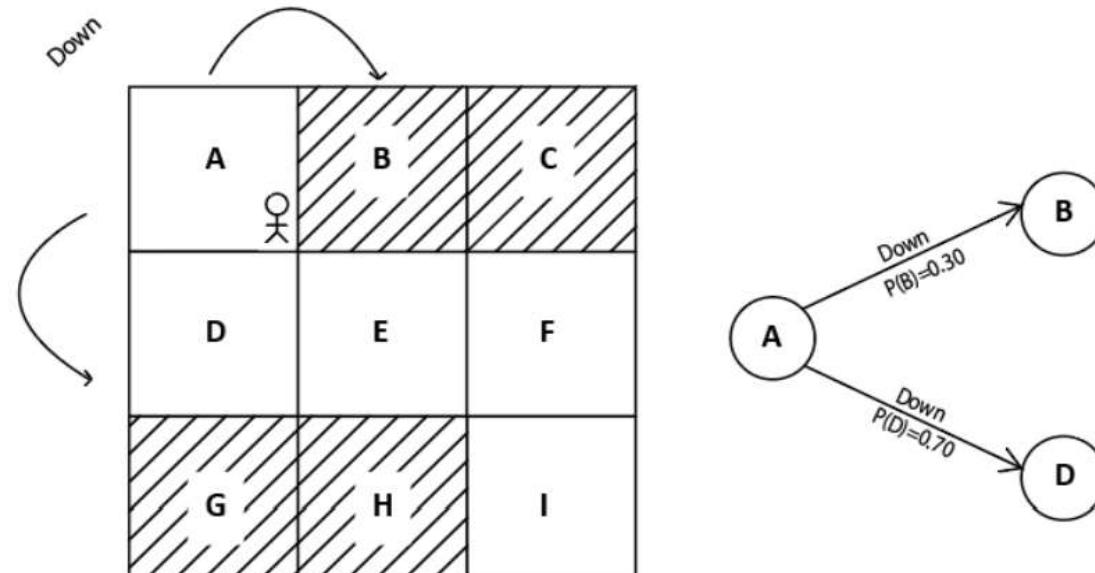


Figure 1.31: Stochastic environment

Types of RL environments

3. **Discrete Environment:** The action space of the environment is discrete.

Ex: action space of the grid world environment is [up, down, left, right].

4. **Continuous environment:** The action space of the environment is continuous

Ex1: To train an agent to drive a car, then the action space will involve multiple actions like [changing the car's speed, the no. of degrees to rotate the wheel, etc..]

Ex2: In a basketball game, the position of players (**Environment**) keeps changing continuously and hitting (**Action**) the ball towards the basket can have different angles and speed so infinite possibilities.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Types of RL environments

5. **Episodic/Non-Sequential Environment:** the agent's current action will not affect the future actions

Ex: A support bot (agent) answering to a question and then answering to another question and so on. So each question-answer is a single episode.

6. **Non-Episodic/Sequential Envt:** the agent's current action will affect its future actions.

Ex: a chess-board is a sequential environment since the agent's current action will affect its future actions in a chess match.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



7. Single and Multi-agent environments:

- **Single agent environment** where an environment is explored by a single agent. All actions are performed by a single agent in the environment.
- **Real-life Example:** Playing tennis against the ball is a single agent environment where there is only one player.
- If two or more agents are taking actions in the environment, it is known as a multi-agent environment.
- **Real-life Example:** Playing a soccer match is a multi-agent environment.

Markov decision process (MDP)

- LO1: State the Markov property, Markov chain, Markov reward process and the Markov decision process.
- LO2: Explain the Markov chain with a suitable example
- LO3: Recognize the mathematical essentials of RL
- LO4: Interpret the grid world RL environment as a MDP

Markov Decision Process(MDP)

- MDP is mainly used to study optimization problems via dynamic programming.
- **A Markov decision process (MDP) refers to a stochastic decision-making process that uses a mathematical framework to model the decision-making of a dynamic system.**
- **It is used in scenarios where the results are either random or controlled by a decision maker, which makes sequential decisions over time.**
- **MDPs evaluate which actions the decision maker should take considering the current state and environment of the system.**
- Almost all RL problems can be modelled as a MDP.
- In artificial intelligence, MDPs model sequential decision-making scenarios with probabilistic dynamics.
- They are used to design intelligent machines or agents that need to function longer in an environment where actions can yield uncertain results.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MDP

- MDP uses two entities namely Markov property and Markov chain.
- **Markov property** : **says the future depends only on the present and not on the past.**
- **Markov chain/Markov process** : has a sequence of states that strictly obey the Markov property.
- Markov chain is a probabilistic model that solely depends on the current state to predict the next state and not the previous states.
- **The future is conditionally independent of the past.**
- Ex : if the current state of weather is cloudy, we can predict the next state to be rainy.
- We made this prediction only based on the current state(cloudy) and not on the previous states which might be sunny, windy, etc.
- **Markov property is not valid for all processes.**
- Ex: while throwing a dice(the next state), doesn't depend on the previous number that showed up on the dice(the current state)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MDP

- MDP uses states and state transition probabilities.
- $P(s'|s)$ is the probability of moving from current state s to next state s'
- State transition probabilities can be represented using Markov table, state diagram or transition matrix.

Current State	Next State	Transition Probability
Cloudy	Rainy	0.7
Cloudy	Windy	0.3
Rainy	Rainy	0.8
Rainy	Cloudy	0.2
Windy	Rainy	1.0

Table 1.1: An example of a Markov table

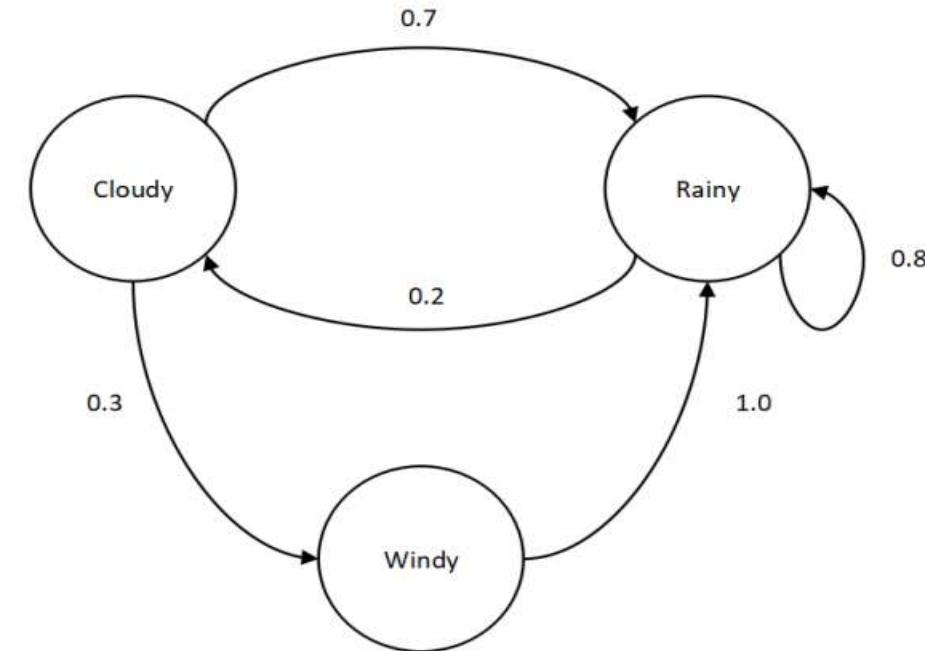


Figure 1.9: A state diagram of a Markov chain

	Cloudy	Rainy	Windy
Cloudy	0.0	0.7	0.3
Rainy	0.2	0.8	0.0
Windy	0.0	1.0	0.0

Figure 1.10: A transition matrix

- Hence, a Markov process consists of a set of states along with their transition probabilities

MDP...

- **Markov Reward process (MRP) :**

- An extension of the Markov chain with the reward function.
- A reward function says the reward we obtain in each state $R(s)$.
- The MRP consists of states s , transition probabilities $P(s'|s)$ and a reward function $R(s)$.

- **Markov Decision Process(MDP):**

- An extension of the MRP with states s , actions a , transition probabilities $P(s'|s)$ and a reward function $R(s)$.
- In a RL setup, the agent makes decisions only based on the current state and not based on the past states.
- Hence we can model a RL problem as a MDP



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Grid World as MDP

- Goal : the agent has to move from state A to state I, without visiting the shaded states.

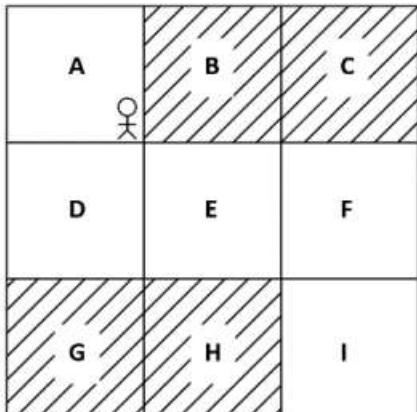


Figure 1.11: Grid world environment

- States : set of states, from A to I
- Actions : a set of actions that our agent can perform in each state as in up, down, left, right
- Transition probability: The probability of moving from the current state s to the next state s' , while performing an action a is denoted by $P(s'|s, a)$
- Ex: $P(B|A, \text{right}) = 1.0$

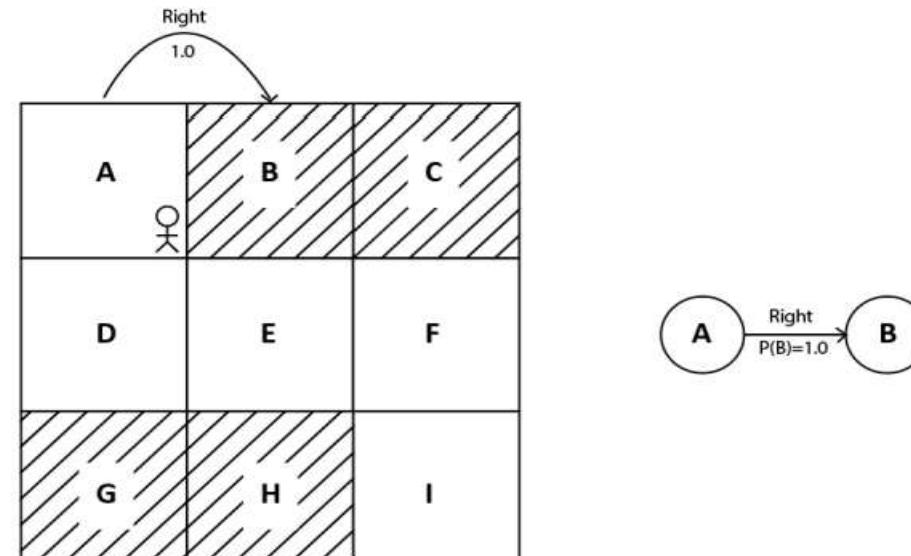


Figure 1.12: Transition probability of moving right from A to B



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Grid World as MDP

- **Reward function:** the reward the agent receives while moving from state s to state s' while performing action a . Denoted by $R(s, a, s')$.
- Ex: $R(A, right, B) = -1$. $R(C, down, F) = +1$

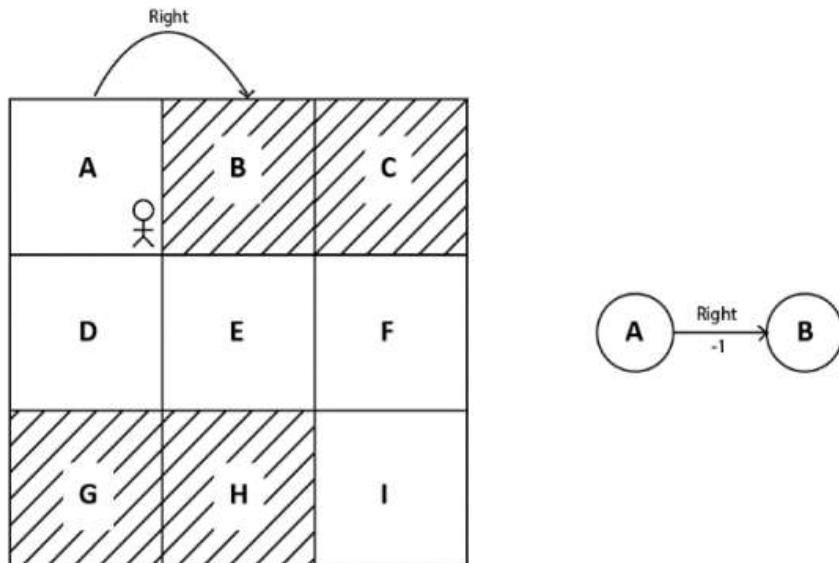


Figure 1.14: Reward of moving right from A to B

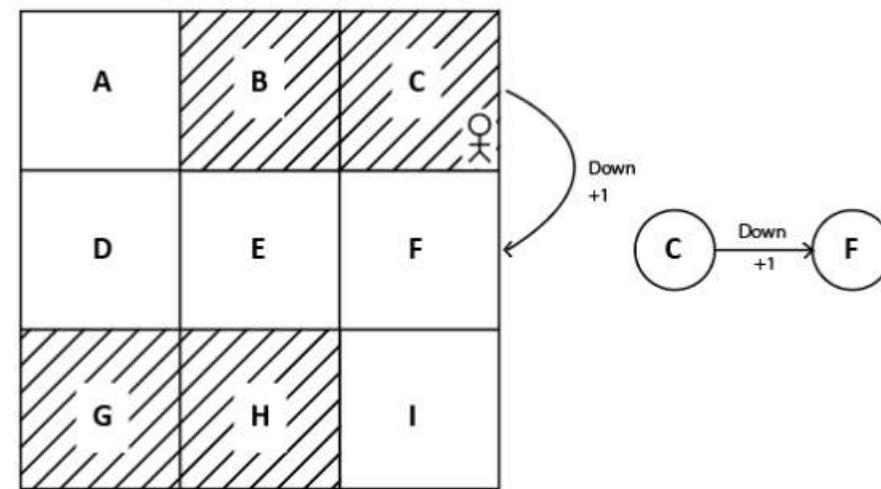


Figure 1.15: Reward of moving down from C to F

?

Fundamental concepts of RL

- **Maths essentials : Expectation of a random variable X**
- A random variable takes values from a random experiment such as throwing a dice, tossing a coin, etc.
- Ex : if we are throwing a fair dice, then the possible outcomes(X) are 1,2,3,4,5, and 6.
- The probability of occurrence of each of these outcomes are 1/6

X	1	2	3	4	5	6
P(x)	1/6	1/6	1/6	1/6	1/6	1/6

Table 1.2: Probabilities of throwing a dice

- Find the average value of the random variable X?

Ans : take the weighted average of X

$$E(X) = \sum_{i=1}^N x_i p(x_i)$$

Thus, the expectation of the random variable X is $E(X) = 1(1/6) + 2(1/6) + 3(1/6) + 4(1/6) + 5(1/6) + 6(1/6) = 3.5$.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



😊Fundamental concepts of RL

- **Expectation of a function of a random variable X**
- Ex: Let $f(X) = X^2$. Find the expected value of $f(X)$

x	1	2	3	4	5	6
$f(x)$	1	4	9	16	25	36
$P(x)$	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$

Table 1.3: Probabilities of throwing a dice

The expectation of a function of a random variable can be computed as:

$$\mathbb{E}_{x \sim p(x)}[f(X)] = \sum_{i=1}^N f(x_i)p(x_i)$$

Thus, the expected value of $f(X)$ is given as $E(f(X)) = 1(1/6) + 4(1/6) + 9(1/6) + 16(1/6) + 25(1/6) + 36(1/6) = 15.1$.

Fundamental concepts of RL

- **Action Space** : the set of all possible actions in the environment
 - Ex: for the grid world environment, the action space is [up,down,left,right]
- **Types of Action Space**: discrete and continuous
- **A Discrete action space** has actions that are discrete.
 - Ex: the action space of the grid world environment
- **A continuous action space** has action that are continuous.
 - Ex: training an agent to drive a car, actions are continuous in nature such as speed, degrees to rotate the wheel, etc..



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Fundamental concepts of RL

- A **policy** defines the agent's behaviour in an environment.
- It tells the agent what action to perform in each state.
- Ex: in the grid world envt with states from A to I, and 4 actions, a policy may tell the agent to move **down** in state A, move **right** in state D, so on.
- In the first iteration, the agent starts with a random policy, taking a random action in each state.
- Learns whether the actions taken in each state are good or bad based on the reward it gets.
- Over a series of iterations, the agent learns a good policy that gets a positive reward.
- This **good(optimal) policy** is the policy that gets the agent a good reward and helps the agent to reach the goal state.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Fundamental concepts of RL..

- Ex of an optimal policy

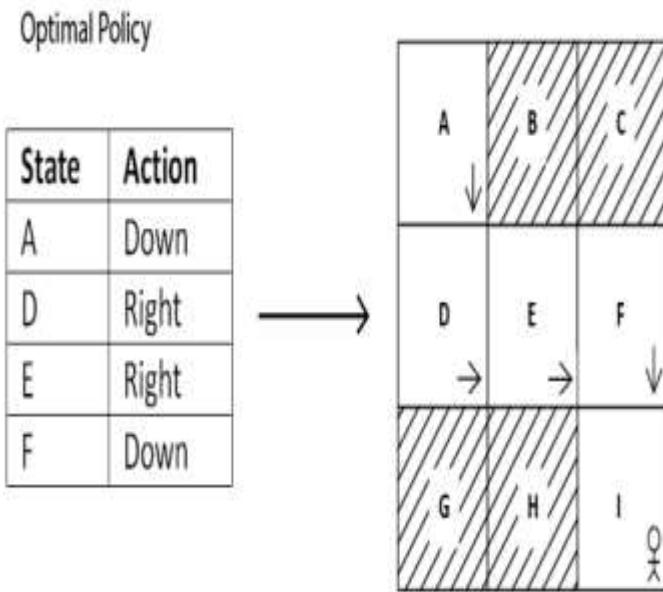


Figure 1.17: The optimal policy in the grid world environment

- Types of policy : deterministic and stochastic
- **Deterministic Policy:**
 - this policy tells the agent to perform one particular action in a state.
 - Denoted by μ
 - If the agent is in state 's' at time 't', the deterministic policy tells the agent to perform action 'a', expressed by $a_t = \mu(s_t)$
Ex : $\mu(A) = \text{down}$
- **Stochastic policy :**
 - This policy doesn't map a state to one particular action.
 - It maps a state to a probability distribution over an action space
 - Denoted by π , $a_t \sim \pi(s_t)$ or $\pi(a_t|s_t)$
 - Ex: if the stochastic policy for state A over the 4 action space [up,down,left, right] is [0.10,0.70,0.10,0.10] respectively, when the agent in state A, it chooses action 'up' 10% of the time, 'down' 70% of the time, left 10% of the time and right 10% of the time.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Fundamental concepts of RL..

Deterministic policy

Maps states → Action

Example :

A → Down

Stochastic policy

Maps states → Probability distribution over action space

Example :

A → [0.10, 0.70, 0.10, 0.10]
up down left right

Figure 1.18: The difference between deterministic and stochastic policies



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- **Types of Stochastic policy : Categorical and Gaussian**
- **Categorical policy:**
- If the action space of a stochastic policy is discrete, then it is a categorical policy
- Prob distributions are taken over a discrete action space

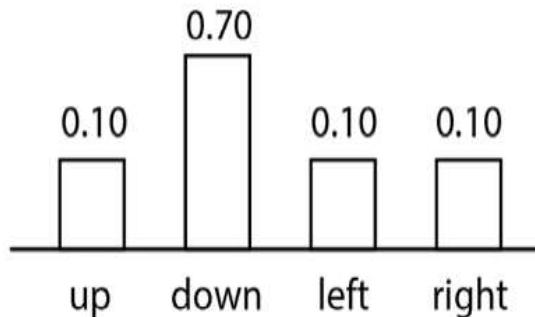


Figure 1.19: Probability of next move from state A for a discrete action space

- **Gaussian policy**
- A stochastic policy whose action space is continuous
- It uses a Gaussian prob distribution over an action space
- Ex: if we are training an agent to drive a car, there is a continuous action in our action space – speed of the car whose value ranges from 0 to 150kmph.
- The stochastic policy uses the Gaussian distribution over the action space to select an action

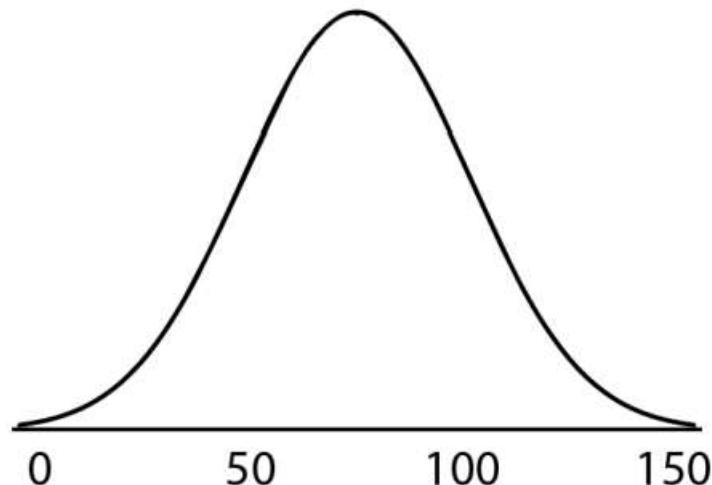


Figure 1.20: Gaussian distribution

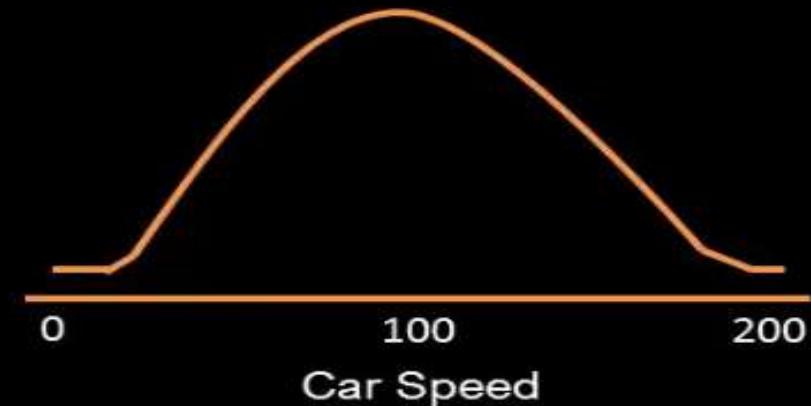


**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Categorical Policy vs Gaussian Policy in Reinforcement Learning



Categorical Policy is used when action space is discrete. It selects the actions from the categorical distribution of discrete action space.

Gaussian Policy is used when action space is continuous. It selects the action from the Gaussian distribution of continuous action space.



**PRESIDENCY
UNIVERSITY**



Private University Estd. in Karnataka State by Act No. 41 of 2013

- **Episode**: the agent-environment interaction starting from the initial state until the final state is called an episode
- Often known as a trajectory(the path taken by the agent)
- Denoted by τ
- An agent can play a game any no. of episodes, each episode is independent of the other.
- What is the use of playing the same game for multiple episodes?
 - To learn the optimal policy, that is, the policy that tells the agent to perform the correct action in each state
- The episode information is of the form state, action, reward starting from the initial state to the final state, i.e $(s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T)$

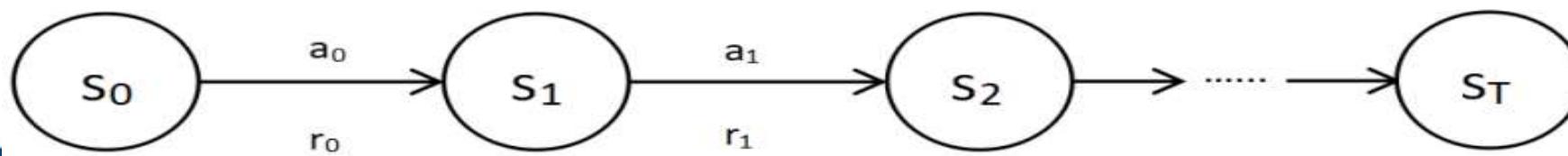


Figure 1.21: An example of an episode

Episode and optimal policy in Grid world Envt

- The agent generates the first episode using a random policy
- Explores the envt over several episodes to learn an optimal policy
- **Episode 1**

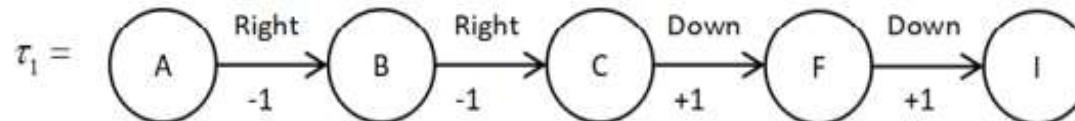
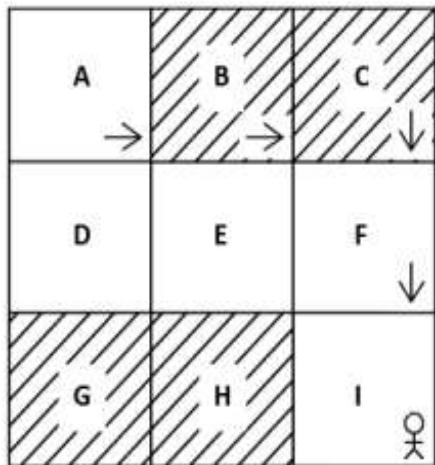


Figure 1.22: Episode 1



**PRESIDENCY
UNIVERSITY**



Private University Estd. in Karnataka State by Act No. 41 of 2013

- **Episode 2** : the agent tries a different policy to avoid the negative rewards it got in the previous episode

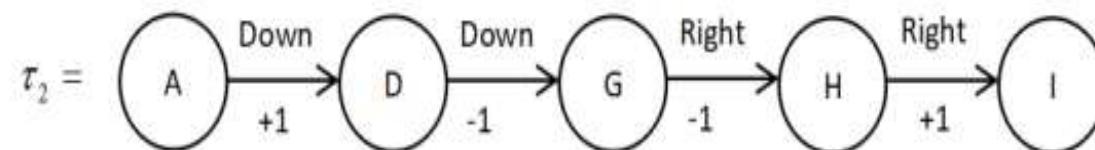
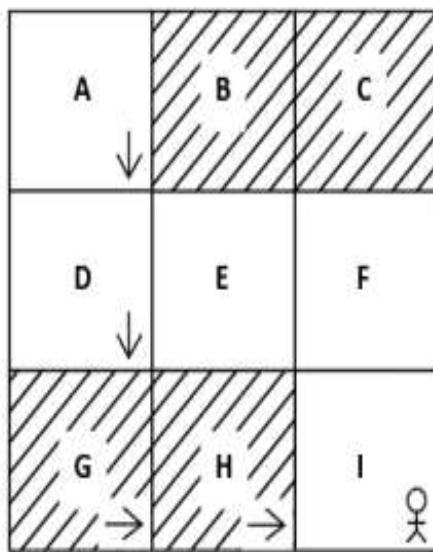


Figure 1.23: Episode 2

- **Episode n:** Over a series of episodes, the agent learns the optimal policy, the policy that takes the agent from state A to state I, without visiting the shaded states and also maximising the rewards.

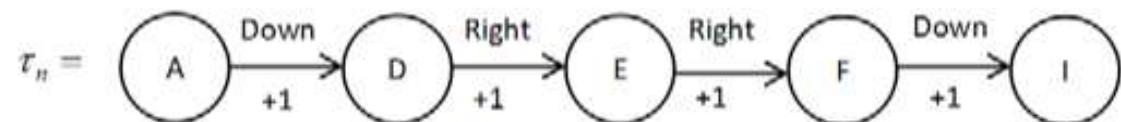
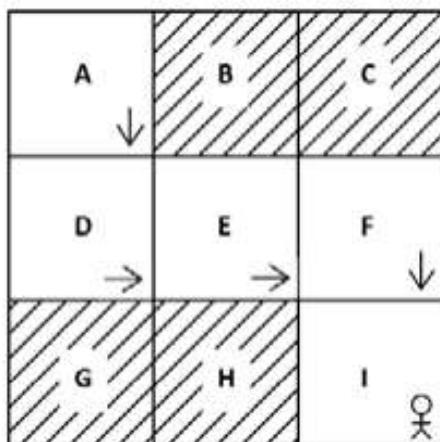


Figure 1.24: Episode n



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Episodic and continuous tasks

- **Episodic tasks:** a task made up of episodes and thus they have a terminal state
 - Ex: car racing game
- **Continuous tasks:** do not have any episodes and so don't have any terminal state.
 - Ex: a personal assistance robot does not have a terminal state
- **Horizon :** the time step until which the agent interacts with the envt.
- **Types : finite and infinite horizon**
- **Finite horizon :** the agent-envt interaction stops at a particular time step.
 - Ex: in an episodic task, the agent-envt interaction stops after the agent reaches the final state T.
- **Infinite horizon:** the agent-envt interaction never stops
 - Ex: a continuous task without final state has an infinite horizon.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Return and discount factor

- **Return** : sum of the rewards obtained by an agent in an episode.
- Denoted by R or G.
- Ex: if the agent starts at initial state at time step t=0 and reaches the final state at time step T, then the return by the agent is

$$R(\tau) = r_0 + r_1 + r_2 + \dots + r_T$$

$$R(\tau) = \sum_{t=0}^T r_t$$

- Ex: for the trajectory below:

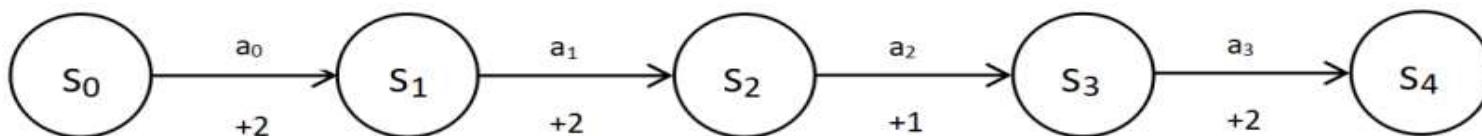


Figure 1.25: Trajectory/episode τ

The return of the trajectory is the sum of the rewards, that is,
 $R(\tau) = 2 + 2 + 1 + 2 = 7$.

Return and discount factor..

- So the goal of the agent is to maximise the return, i.e, maximise the sum of the rewards obtained over an episode.
- How can we maximise this return? How can we perform the correct action in each state?
- By using the **optimal policy – the policy that gets our agent the maximum return (sum of the rewards) by performing the correct action in each state.**
- How to define return for continuous tasks, where there is no terminal state?
- Return for continuous tasks – sum of the rewards upto infinity.

$$R(\tau) = r_0 + r_1 + r_2 + \dots + r_{\infty}$$

Return and discount factor..

- **How to maximise the return that sums to infinity?**
 - Using discount factor

$$R(\tau) = \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \dots + \gamma^n r_\infty$$

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

- Discount factor helps us by preventing the return in reaching infinity, by deciding how much importance we should give to immediate rewards and future rewards.
- Its value ranges from 0 to 1



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Return and discount factor..

- If the discount factor is too small(close to 0), it means we give more importance to immediate rewards than future rewards.
- If the discount factor is set to a large value(close to 1), it means we give more importance to future rewards than immediate rewards.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Return and discount factor..

- **What happens when the discount factor is small? $\gamma = 0.2$?**

$$\begin{aligned} R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \dots \\ &= (0.2)^0 r_0 + (0.2)^1 r_1 + (0.2)^2 r_2 + \dots \\ &= (1)r_0 + (0.2)r_1 + (0.04)r_2 + \dots \end{aligned}$$

- At time step 0, the reward r_0 is weighted by a discount factor of 1.
- At time step 1, the reward r_1 is weighted by a heavily decreased discount factor of 0.2.
- At time step 2, the reward r_2 is weighted by a heavily decreased discount factor of 0.04.
- **So, when we set discount factor to a small value, we give more importance to immediate rewards than future rewards.**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Return and discount factor..

- What happens when the discount factor is large? $\gamma = 0.9$?

$$\begin{aligned} R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \dots \\ &= (0.9)^0 r_0 + (0.9)^1 r_1 + (0.9)^2 r_2 + \dots \\ &= (1)r_0 + (0.9)r_1 + (0.81)r_2 + \dots \end{aligned}$$

- At time step 0, the reward r_0 is weighted by a discount factor of 1.
- At time step 1, the reward r_1 is weighted by a slightly decreased discount factor of 0.9.
- At time step 2, the reward r_2 is weighted by a slightly decreased discount factor of 0.81.

As we can observe, the discount factor is decreased for subsequent time steps but unlike the previous case, the discount factor is not decreased heavily. Thus, when we set the discount factor to a high value, we give more importance to future rewards than the immediate reward.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Immediate or future rewards?

- It depends on the task that we want to train an agent for.
- Suppose, in a chess game, the goal is to defeat the opponent's king.
- If we give importance to the immediate rewards like a reward on pawn defeat any opponent player then the agent will learn to perform these sub-goals no matter if his players are also defeated.
- So, in this task future rewards are more important



Return and discount factor..

- **What happens when the discount factor is set to 0? $\gamma = 0$?**

$$\begin{aligned} R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \dots \\ &= (0)^0 r_0 + (0)^1 r_1 + (0)^2 r_2 + \dots \\ &= (1)r_0 + (0)r_1 + (0)r_2 + \dots \\ &= r_0 \end{aligned}$$

- **Our return is just the immediate reward**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Return and discount factor..

- **What happens when the discount factor is set to 1? $\gamma = 1$?**

$$\begin{aligned} R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \dots \\ &= (1)^0 r_0 + (1)^1 r_1 + (1)^2 r_2 + \dots \\ &= r_0 + r_1 + r_2 + \dots \end{aligned}$$

- **Our return is just the sum of the rewards upto infinity**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Return and discount factor..

- If the discount factor is set to 0, the agent never learns, as it considers only the immediate reward.
- If the discount factor is set to 1, the agent will learn forever, looking for the future rewards that lead to infinity.
- So, the optimal value of discount factor is between 0.2 to 0.8
- For certain tasks future rewards are more important than immediate rewards and vice versa.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Value function

- **Value function** also called the state value function gives the value of a state.
- The value of a state ‘s’ is the return of the trajectory τ starting from that state to the final state following a policy π .

$$V^\pi(s) = [R(\tau)|s_0 = s]$$

- The policy could be deterministic or stochastic
- A deterministic policy maps each state to a one particular action
- A stochastic policy selects action for a state based on a probability distribution of the action space.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Value function for a deterministic policy

- If the trajectory τ , for the grid world environment, using some policy deterministic policy π is :

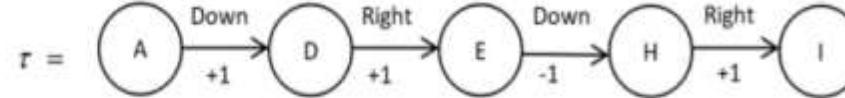
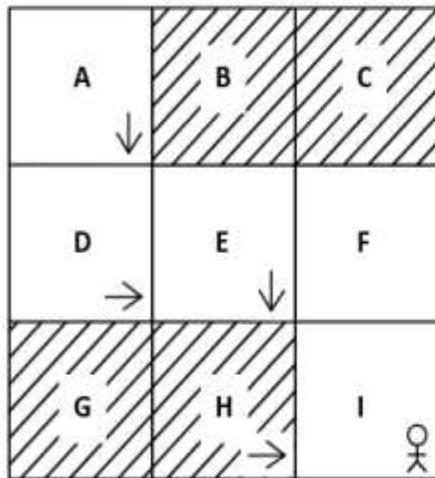


Figure 1.26: A value function example



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Value function for a deterministic policy...

- The value function can be calculated for each state as the return(sum of the rewards) of the trajectory starting from that state :
 - The value of state **A** is the return of the trajectory starting from state A. Thus,
 $V(A) = 1+1+ -1+1 = 2.$
 - The value of state **D** is the return of the trajectory starting from state D. Thus,
 $V(D) = 1-1+1= 1.$
 - The value of state **E** is the return of the trajectory starting from state E. Thus,
 $V(E) = -1+1 = 0.$
 - The value of state **H** is the return of the trajectory starting from state H. Thus,
 $V(H) = 1.$
- The value function of the finals state is zero, since a reward is associated only with a state that has a transition.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Value function for stochastic policy

- (Expected) value function of a state with a stochastic policy is the expected return that the agent would get starting from that state s and following a stochastic policy π .
- Return of a state in a trajectory τ , following a stochastic policy π , is a random variable.
- It takes different values with some probability in each trajectory.
- It is expressed as :

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s]$$

Value function for stochastic policy...

- Ex: In state A, the stochastic policy gives a prob distribution over the action space [up,down,left,right] as [0.0, 0.8, 0.0, 0.2], i.e perform the action down 80% of the time, that is, $\pi(\text{down}|A) = 0.8$, and the action right 20% of the time, that is $\pi(\text{right}|A) = 0.20$
- This gives two trajectories from state A.
- Assume the stochastic policy selects “right” in states D and E and “down” in B and F 100% of the time.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- $\pi(\text{down}|A) = 0.8 \quad \pi(\text{right}|A) = 0.20$
- $\pi(\text{right}|D) = 1 \quad \pi(\text{right}|E) = 1$
- $\pi(\text{down}|B) = 1 \quad \pi(\text{down}|F) = 1$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Value function for stochastic policy...

- The first trajectory τ_1 with $\pi(\text{down}|A) = 0.8$ is :

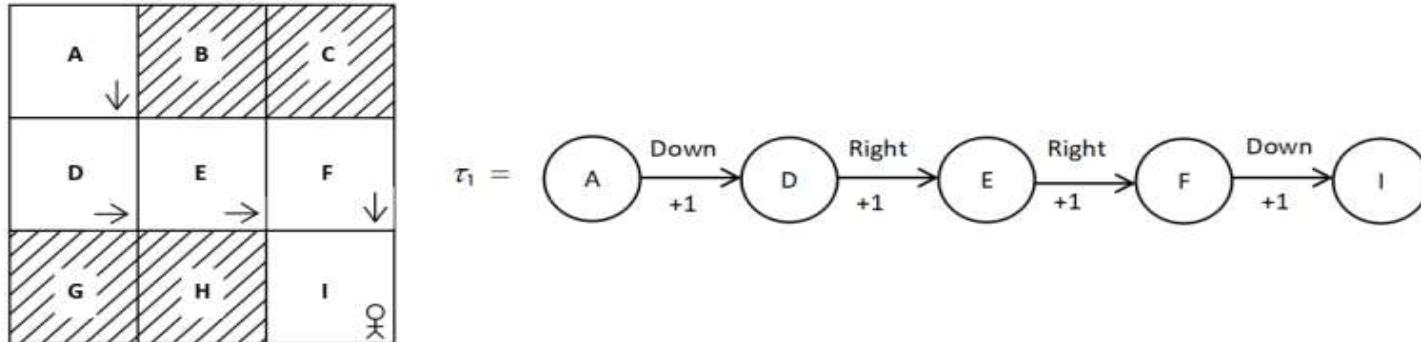


Figure 1.27: Episode τ_1

- Value of state A is, the return(sum of the rewards) of the trajectory starting from state A.
- Thus, $V(A) = R(\tau_1) = 1+1+1+1=4$.

Value function for stochastic policy...

- The second trajectory τ_2 with $\pi(\text{right}|A) = 0.20$ is :

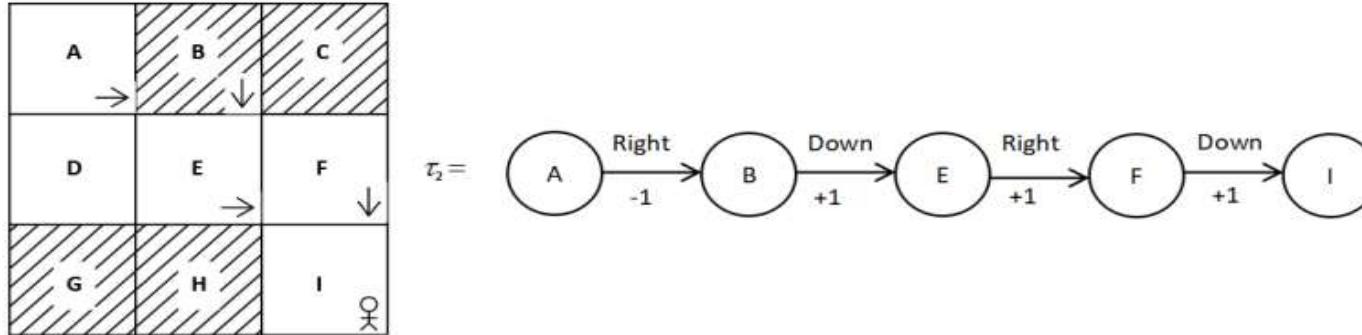


Figure 1.28: Episode τ_2

- Value of state A is, the return(sum of the rewards) of the trajectory τ_2 starting from state A.
- Thus, $V(A) = R(\tau_2) = -1 + 1 + 1 + 1 = 2$.
- Also, it's the same policy, $V(A)$ differs with the trajectories.
- For this policy, return is 4, 80% of the time and 2, for 20% of the time.

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s]$$

Value function for stochastic policy...

- Value of a state for a stochastic policy is the expected return of the trajectory starting from that state.

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s]$$

- Expected return is the weighted average, sum of the returns, multiplied by their probabilities.
- So, $V(A)$ is :

$$\begin{aligned} V^\pi(A) &= \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = A] \\ &= \sum_i R(\tau_i) \pi(a_i | A) \\ &= R(\tau_1) \pi(\text{down} | A) + R(\tau_2) \pi(\text{right} | A) \\ &= 4(0.8) + 2(0.2) \\ &= 3.6 \end{aligned}$$

Value function for stochastic policy...

- Thus, the value of a state is the expected return of the trajectory starting from that state.
- Value function depends on the policy.
- There can be many value functions for a state, according to different policies.
- The optimal value function $V^*(s)$ is the maximum value of the state, among all its value functions

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Ex: Text book Pg.33 . We can find the optimal state from a Value table.

State	Value
s_0	7
s_1	11

Table 1.4: Value table

Q function

- Q function denotes the value of a state-action pair for a particular state, s .
- It is the return that the agent will obtain starting from a state s , and performing an action a , following a policy π
- It is also known as state-action value function

$$Q^\pi(s, a) = [R(\tau) | s_0 = s, a_0 = a]$$

- Ex: Given trajectory τ :

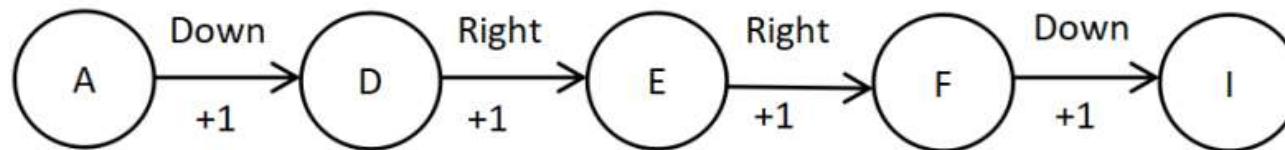


Figure 1.29: A trajectory/episode example



PRESIDENCY
UNIVERSITY



Private University Estd. in Karnataka State by Act No. 41 of 2013

Q function...

- Find the q-function of **A-down, D-right :**

- **Ans 1:**

$$Q^\pi(A, \text{down}) = [R(\tau) | s_0 = A, a_0 = \text{down}]$$

$$Q(A, \text{down}) = 1 + 1 + 1 + 1 = 4$$

- **Ans 2: ? 3**

- **Since return is a random variable taking a different value with some probability, instead of taking the return directly, we take the expected return.**

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s, a_0 = a]$$

It implies that the Q value is the expected return the agent would obtain starting from state s and performing action a following policy π .



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Q function....

- Q function depends on the policy.
- There will be Q values for a (s,a) pair depending on the policy.
- The optimal policy for a (s,a) pair is :

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

- The optimal policy π^* is the policy that gives the maximum Q value for a (s,a).
- Given a Q table, we can find the optimal policy π^*

Q Table			Optimal policy	
State	Action	Value	State	Action
s ₀	0	9		
s ₀	1	11		
s ₁	0	17		
s ₁	1	13		

→

State	Action
s ₀	1
s ₁	0

Table 1.6: Optimal policy extracted from the Q table



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Model-based and model-free learning

- **Model-based learning** : the agent learns the optimal policy by using the model dynamics of the environment.
- Model dynamics of the environment is defined using
 - 1. state transition probabilities and 2. reward function
- **Model-free learning** : the agent tries to learn the optimal policy without using the model dynamics of the environment



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Bellman Equation and Dynamic Programming

- LO1: State the Bellman equation of the value function and Q function
- LO2: Solve MDP using Bellman equation
- LO3: Apply Bellman equation to find the value function of a state for a given deterministic environment

Bellman Equation and Dynamic Programming

- In RL, the agent has to learn an optimal policy to perform a particular task
- An optimal policy selects the correct action for an agent in each state, so that the agent can get the maximum return and achieve its goal.
- Two classical RL algorithms – **value and policy iteration**, helps the agent to learn an optimal policy.
- These algos are model based and use Dynamic Programming
- Bellman equation is used in RL to find the optimal value function and optimal Q functions recursively.
- These are then used to find the optimal policy

Bellman Equation?

- Bellman equation?
- Bellman optimality equation?
- Relationship between value and Q function?
- Dynamic Programming – value and policy iteration methods?
- Solving the Frozen Lake problem using value and policy iteration methods.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Bellman equation for the value function

- As per Bellman Equation, the value of a state is the sum of the immediate reward $R(s, a, s')$ and the discounted value of the next state $\gamma V(s')$.

$$V(s) = R(s, a, s') + \gamma V(s')$$

- $R(s, a, s')$ implies the immediate reward obtained while performing an action a in state s and moving to the next state s'
- γ is the discount factor
- $V(s')$ implies the value of the next state

Bellman equation for the value function

- **In a deterministic environment:**
- Ex: Given a trajectory τ using some policy π as :

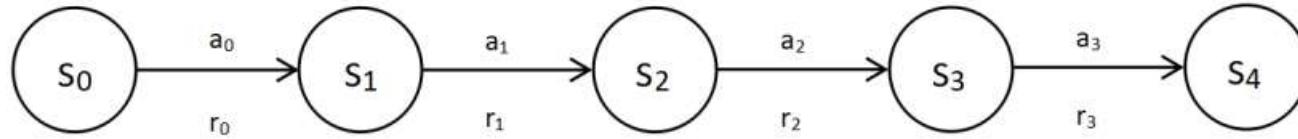


Figure 3.1: Trajectory

- Using Bellman equation find $V(s_2)$?
- Ans:
$$\begin{aligned} V(s_2) &= R(s_2, a_2, s_3) + \gamma V(s_3) \\ &= r_2 + \gamma V(s_3) \end{aligned}$$
- Hence the Bellman equation of the value function for a deterministic environment associated with a policy π is :
$$V^\pi(s) = R(s, a, s') + \gamma V^\pi(s')$$
- Where the rhs term is known as **Bellman backup**

Bellman equation for the value function

- In a stochastic environment:

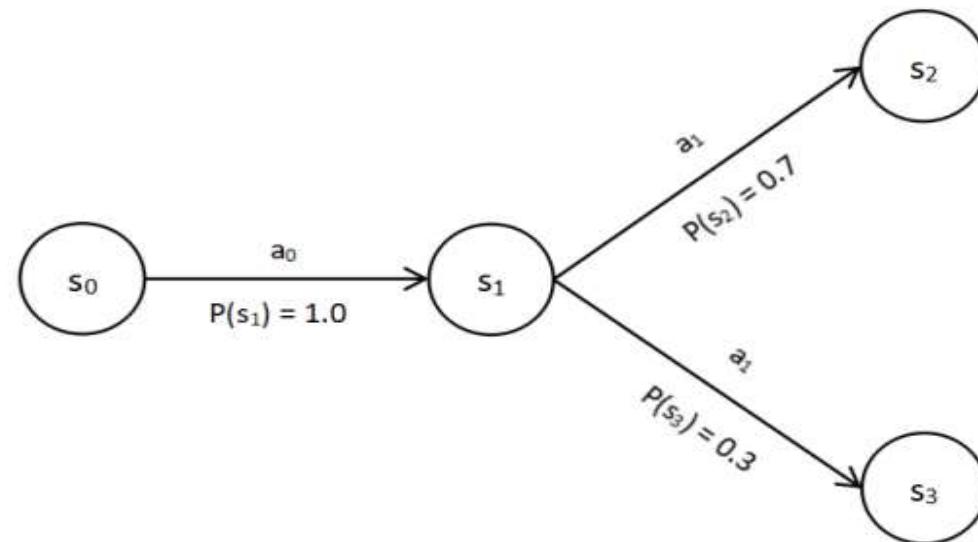


Figure 3.2: Transition probability of performing action a_1 in state s_1

Bellman equation for the value function

- Modify the Bellman equation of the value function with expectations (weighted average)
- Bellman backup multiplied with the transition probability of the next state.

$$V^\pi(s) = \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^\pi(s')]$$

In the preceding equation, the following applies:

- $P(s'|s, a)$ denotes the transition probability of reaching s' by performing an action a in state s
- $[R(s, a, s') + \gamma V^\pi(s')]$ denotes the Bellman backup



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Bellman equation for the value function

- Ex: same trajectory, find $V(s_1)$?
- Ans:

$$V(s_1) = P(s_2|s_1, a_1)[R(s_1, a_1, s_2) + V(s_2)] + P(s_3|s_1, a_1)[R(s_1, a_1, s_3) + V(s_3)]$$

$$V(s_1) = 0.70[R(s_1, a_1, s_2) + V(s_2)] + 0.30[R(s_1, a_1, s_3) + V(s_3)]$$

- **Hence the Bellman equation for the value function for a stochastic environment for a policy π is :**

$$V^\pi(s) = \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^\pi(s')]$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Bellman equation for the value function

- **What if the policy itself is stochastic?** Instead of performing the same action in a state, we select an action based on the prob distbn over the action space.
- Ex:

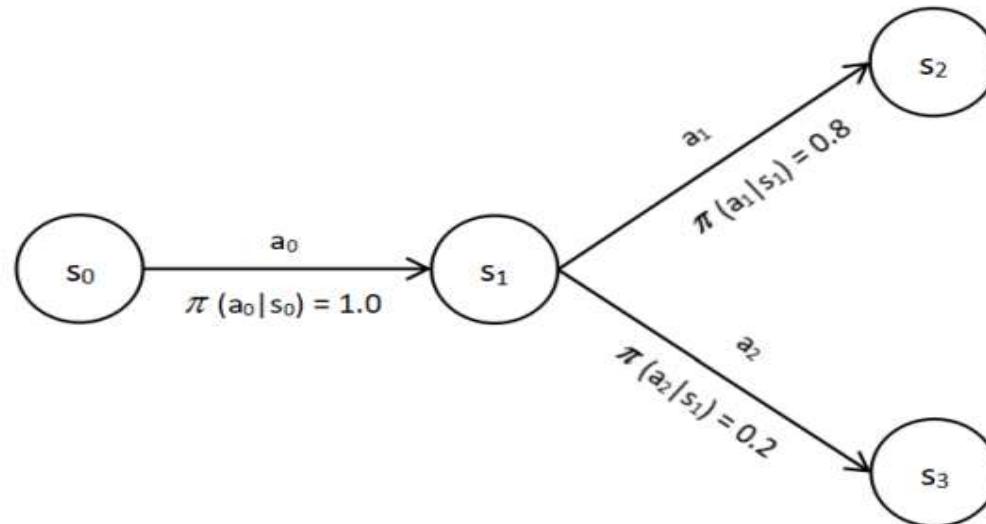


Figure 3.3: Trajectory using a stochastic policy



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Bellman eqn for value function...

- to include the **stochasticity present in the environment** in the Bellman equation, we took the expectation (the weighted average), that is, a sum of the Bellman backup multiplied by the corresponding transition probability of the next state. •
- Similarly, to include the **stochastic nature of the policy** in the Bellman equation, we can use the expectation (the weighted average), that is, a sum of the Bellman backup multiplied by the corresponding probability of action.

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^\pi(s')]$$

- Using expectations :

$$V^\pi(s) = \mathbb{E}_{\substack{a \sim \pi \\ s' \sim P}} [R(s, a, s') + \gamma V^\pi(s')]$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Bellman Eqn of the Q function

- For **deterministic envt**: Bellman eqn of the Q function says that, the Q value of a state-action pair is a sum of the immediate reward and the discounted Q value of the next state-action pair :

$$Q(s, a) = R(s, a, s') + \gamma Q(s', a')$$

In the preceding equation, the following applies:

- $R(s, a, s')$ implies the immediate reward obtained while performing an action a in state s and moving to the next state s'
- γ is the discount factor
- $Q(s', a')$ is the Q value of the next state-action pair



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Ex: Given a trajectory τ using some policy π , find the Q value of (s_2, a_2) .

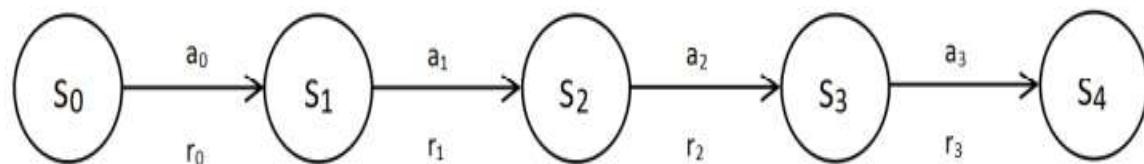


Figure 3.4: Trajectory

- *Ans:*

$$Q(s_2, a_2) = R(s_2, a_2, s_3) + \gamma Q(s_3, a_3)$$

$$Q(s_2, a_2) = r_2 + \gamma Q(s_3, a_3)$$

Thus, the Bellman equation for the Q function can be expressed as:

$$Q^\pi(s, a) = R(s, a, s') + \gamma Q^\pi(s', a')$$

Where the superscript π implies that we are using the policy π and the right-hand side term $R(s, a, s') + \gamma Q^\pi(s', a')$ is the **Bellman backup**.

Bellman Eqn of the Q function...

- **For stochastic environment:** An agent in state ‘s’, performs an action ‘a’, then the next state is not always the same.
- Bellman eqn of the Q function for a stochastic envt, uses the expectation (weighted average), that is, a sum of the Bellman backup multiplied by their corresponding transition probability of the next state.
- The Bellman equation of the Q function is:

$$Q^\pi(s, a) = \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma Q^\pi(s', a')]$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Bellman Eqn of the Q function...

- With a stochastic policy, we cannot guarantee a' in s'

$$Q^\pi(s, a) = \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a') \right]$$

- Final Bellman Equation for $Q(s, a)$ using expectation is :

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim P} [R(s, a, s') + \gamma \mathbb{E}_{a' \sim \pi} Q^\pi(s', a')]$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Bellman Optimality Theorem

- Gives the optimal Bellman value and Q function
- $V(s)$ depends on the policy .
- The optimal Bellman $V(s)$ for a particular state s , is denoted by $V^*(s)$
- It is the one that gives the maximum value among all the values of that state.
- To find $V^*(s)$, find $V(s)$ using all possible actions, without using any policy. Choose the max among all $V(s)$.

$$V^*(s) = \max_a \mathbb{E}_{s' \sim P} [R(s, a, s') + \gamma V^*(s')]$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- If there are two possible actions, namely 0 and 1 in state s, then

$$V^*(s) = \max \left(\begin{array}{l} \mathbb{E}_{s' \sim P}[R(s, 0, s') + \gamma V^*(s')] \\ \mathbb{E}_{s' \sim P}[R(s, 1, s') + \gamma V^*(s')] \end{array} \right)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Optimal Bellman Q function

- Without using any policy, find the $Q(s', a')$ for all possible a' , find the max of them.

$$Q^*(s, a) = \mathbb{E}_{s' \sim P} [R(s, a, s') + \gamma \max_{a'} Q^*(s', a')]$$

- If there are two actions 0 and 1 in s' then,

$$Q^*(s, a) = \mathbb{E}_{s' \sim P} [R(s, a, s') + \gamma \max(Q^*(s', 0), Q^*(s', 1))]$$

To summarize....

- The Bellman optimality equations of the value and the Q functions are :

$$V^*(s) = \max_a \mathbb{E}_{s' \sim P} [R(s, a, s') + \gamma V^*(s')]$$

$$Q^*(s, a) = \mathbb{E}_{s' \sim P} \left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right]$$

- They can be written by removing the expectation as

$$V^*(s) = \max_a \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V^*(s')]$$

$$Q^*(s, a) = \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma \max_{a'} Q^*(s', a')]$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Relationship between $V^*(s)$ and $Q^*(s, a)$ is :
- The optimal value function is the maximum expected return when we start from a state s
$$V^*(s) = \max_{\pi} V^{\pi}(s)$$
- The optimal Q function is the maximum expected return when we start from a state s and perform an action a
$$Q^{\pi}(s, a) = \max_{\pi} Q^{\pi}(s, a)$$
- So, we can say that the optimal value function is the maximum of optimal Q value over all possible actions

$$V^*(s) = \max_a Q^*(s, a)$$

- We can derive V from Q :
$$V^*(s) = \max_a Q^*(s, a)$$
- We can derive Q from V :
$$Q^*(s, a) = \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')]$$
- Substituting we get :
$$V^*(s) = \max_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')]$$
- These Bellman equations are used to find an optimal policy.

Dynamic Programming

- Dynamic programming (DP) is a technique for solving complex problems.
- In DP, instead of solving a complex problem as a whole, we break the problem into simple sub-problems,
- Solve each sub-problem and store the solution.
- If the same sub-problem occurs, we don't recompute; instead, we use the already computed solution.
- Thus, DP helps in drastically minimizing the computation time. It has its applications in a wide variety of fields including computer science, mathematics, bioinformatics, etc.
- Two methods that use DP to find an optimal policy:
 - **Value iteration and policy iteration**



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- DP is a model-based method – it needs the model dynamics to find an optimal policy
- Model dynamics – state transition probabilities and reward function
- **An optimal policy** – tells the agent to perform the correct action in each state.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- To find this optimal policy, first we find the optimal value function of each state, $V^*(s)$.
- Later, use this optimal value function $V^*(s)$, to find the optimal policy [by finding the Q function]



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- The Bellman's optimal value function is :

$$V^*(s) = \max_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')] \quad (9)$$

- In the relationship between the value and Q functions, we know that given the value function we can derive the Q function :

$$Q^*(s, a) = \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')] \quad (10)$$

- Substituting (10) in (9), we get

$$V^*(s) = \max_a Q^*(s, a)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- If we know the Q values of all (s, a) pairs of a particular state s , using this we can find the optimal value function of that state.
- Ex: if the Q-values of all (s, a) pairs are :

State	Action	Value
s_0	0	2.7
s_0	1	3
s_1	0	4
s_1	1	2

- Using this we can find the optimal state values of each state as :
- This is the outline of the value-iteration algorithm

State	Value
s_0	3
s_1	4

The value iteration algorithm

- LO1: Name the two methods to find the optimal policy using dynamic programming
- LO2: Describe the value iteration algorithm to find an optimal policy
- LO3: Given the model dynamics of a particular state in a RL environment, apply the value iteration algorithm, to find an optimal policy



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



The value iteration algorithm

- **Step 1:** Compute the optimal value function of each state iteratively, by taking the max of the Q functions (of all possible actions in a state), i,e

$$V^*(s) = \max_a Q^*(s, a)$$

- **Step 2:** Extract the optimal policy from the computed optimal value function



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Step 1 : Value iteration Algorithm..

1. Initialize the value table of all states to zero.
2. For each state s , do :
 - 2.1 for each possible action a in state s do:
 - 2.1.1 find the Q-value of this (s,a) pair as :

$$Q(s, a) = \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V(s')] \quad [\text{OR}] \quad Q(s, a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$

- 2.2 find the max among the Q values and update that as the value of that state

$$V^*(s) = \max_a Q^*(s, a)$$

Value iteration Algorithm..

3. If the value table of 2 consecutive iterations doesn't change then, go to next step to find the optimal policy; else repeat step 2, using the updated value table of this iteration.
4. Find the Q value of each (s,a) pair using the optimal value table obtained in step 3 using :
$$Q(s, a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$
5. Extract the optimal policy from the Q-value table got in step 4.

- Given the model dynamics of state A:

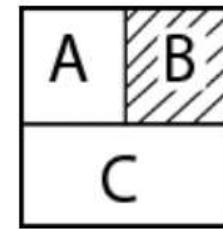
Table 3.3 shows the model dynamics of state A:

State (s)	Action (a)	Next State (s')	Transition Probability $P(s' s,a)$ or $P_{ss'}^a$	Reward Function $R(s,a,s')$ or $R_{ss'}^a$
A	0	A	0.1	0
A	0	B	0.8	-1
A	0	C	0.1	1
A	1	A	0.1	0
A	1	B	0.0	-1
A	1	C	0.9	0

Table 3.3: Model dynamics of state A

Value iteration Algorithm Example...

- Given a smaller grid world environment :
- Actions : 0 – left/right and 1 – up/down
- Goal : from state A reach state C, without visiting the shaded state B**
- What is the optimal policy?**
 - Ans : perform action 1 in state A.**



Solution:

- **Step 1: Compute the optimal value function**

1. Initialize the value table of all states to zero.
2. For each state s and all possible actions a find $Q(s,a)$ as

$$Q(s, a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$

3. Using $V^*(s) = \max_a Q^*(s, a)$ find $\max_a Q^*(s, a)$ and update this as the value table of state s
4. Repeat steps 2 and 3 until the value table converges.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Example for Step 1:

Observation:

Initialize value table to zero

State	value
A	0
B	0
C	0

Step 1: On state A: Possible actions are (0, 1)

∴ find $R(A, 0) \leftarrow \frac{P^0_{AA} [R^0_{AA} + \gamma V(A)] +$

$$R(A, 1) =$$

To vsize $R(s, a) = \cdot \in P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$

$$\begin{aligned}
 \therefore Q(A, D) &= P_{AA}^{\circ} \left[R_{AA}^{\circ} + \gamma V(A) \right] + P_{AB}^{\circ} \left[R_{AB}^{\circ} + \gamma V(B) \right] \\
 &\quad + P_{AC}^{\circ} \left[R_{AC}^{\circ} + \gamma V(C) \right] \\
 &= 0.1 [0 + 0] + 0.8 [-1 + 0] + 0.1 [1 + 0] \\
 &= \cancel{0.8} - 0.8 + 0.1 = -0.7
 \end{aligned}$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



$$\begin{aligned}
 \underline{Q(A, i)} &= P_{AA}^i [R_{AA}^i + \gamma V(A)] + P_{AB}^i [R_{AB}^i + \gamma V(B)] \\
 &\quad + P_{AC}^i [R_{AC}^i + \gamma V(C)] \\
 &= 0.1[0 + 0] + 0[-1 + 0] + 0.9[\underline{0} + 0] \\
 &= 0 + 0 + 0.9 = \underline{0.9}
 \end{aligned}$$

∴ updated value table:

state	value
A	0.9
B	0
C	0



JSS
UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

40
YEARS
OF ACADEMIC
WISDOM

$$Q(B, 0) = P.$$

$$Q(B, 1) =$$

the agent of it no
is:-

(ii) lastly, if we know the model dynamics of state B, & C we can find $Q(B, 0)$, $Q(B, 1)$, $Q(C, 0)$ and $Q(C, 1)$. Finally updated state value

State	Value
A	0.9
B	-0.2
C	0.5



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Q12: Using the value table of itnt, find the values of all states with all possible actions in each state:

Ex: On state A:-

$$R(A, D) = P_{AA}^D [R_{AA}^D + \gamma V(A)] + \\ P_{AB}^D [R_{AB}^D + \gamma V(B)] + \\ P_{AC}^D [R_{AC}^D + \gamma V(C)]$$

$$= 0.1(0 + 0.9) + 0.8(-1 - 0.2) + 0.1(1 + 0.5)$$

$$= -0.72$$

$$R(A, I) = P_{AA}^I [R_{AA}^I + \gamma V(A)] + P_{AB}^I [R_{AB}^I + \gamma V(B)] +$$

$$P_{AC}^I [R_{AC}^I + \gamma V(C)]$$

$$= 0.1(0 + 0.9) + 0.0(-1 - 0.2) + 0.9(1 + 0.5)$$

$$= 1.44$$

Pg (20)

Similarly we find the Q-value of all states & update the state table as [Assumption]

State	Value
A	1.44
B	-0.50
C	1.0

Value table from itnz:

Itn 3: Repeat the same steps while updating values, use the update value table got from the previous itn.

Assume value table from itn 2

state	value
A	1.94
B	-0.70
C	1.3

Convergence?

Keep repeating, until the value table betw consecutive itns does' change or changes by a very small fraction based on a threshold)

Assume value table from itn 3 is

~~wt~~ \because the diff is small, we take this as the optimal value table.

state	value
A	1.95
B	-0.72
C	1.3

Next, step 2: to extract the optimal Policy from the
optimal value table.

Step 2 - Extract the optimal policy from the optimal value table (function) from step 1.

Assume Optimal value table is

State	Value
A	1.95
B	-0.72
C	1.3

Use the R_{t+1} to compute the policy for a state s .

2.1) find the Q -value of all (s, a) pairs are:

$$Q(s, a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$

[Use the optimal value function of step 1 to find $V(s')$]



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



2.d) Extract the policy, by selecting the action
that has the $\underset{a}{\max} Q$ value in a state.

$$\pi^* = \underset{a}{\operatorname{argmax}} Q(s, a)$$

for state A,



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



for state A :-

$$\begin{aligned} R(A, 0) &= P_{AA}^0 [R_{AA}^0 + \gamma V(+)] + P_{AB}^0 [R_{AB}^0 + \gamma V(B)] + \\ &\quad P_{AC}^0 [R_{AC}^0 + \gamma V(C)] \\ &= 0.1 [0 + \underline{0.95}] + 0.4 [-1 - \underline{0.72}] + \\ &\quad 0.1 [1 + \underline{1.3}] \\ &= -0.95 \end{aligned}$$

$$R(A, 1) = 2.26.$$

PoC



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



∴ Q-table is

State	Action	Value
A	0	-0.95
	1 ✓	2.28 ✓
B	0	-0.5
	1 ✓	0.5 ✓
C	0	-1.1
	1 ✓	1.4 ✓

from this Q-table, pick the action in each state that has the max. value as an optimal policy.

∴ In state A, the optimal policy is action 1, i.e., moving down.

Solving the frozen lake problem with value it.

Start	S	F	F	F
	F	H	F	H
	F	F	F	H
	H	F	F	G
				Goal

S - starting state
F - frozen state
H - hole state
G - goal state

States are encoded from 0 to 15.

Actions:

left - 0
down 1
right 2
up - 3



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Step 1: Compute the optimal value fn.

def value_fn(env):

num_iters = 1000

threshold = 1e-20

gamma = 1.0

value_table = np.zeros(env.observation_space.n)

for i in range(num_iters):

updated_value_table = np.copy(value_table)

for s in range(env.observation_space.n):

Q-values = [

sum([prob * ($r + \gamma \cdot \text{updated-value-table}[s]$
for prob, s, r, - in env. $P[s][a]$)
for a in range(env. action-space.n)])

value-table[s] = max(Q-values)

if (np.sum(np.fabs(updated-value-table -
value-table)) <= threshold):

break

return value-table.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



step 1: finding optimal policy:-

def extract-policy (value-table):

$$\gamma = 1.0$$

policy = np.zeros(env.observation_space.n)

for s in range (env.observation_space.n):

\hat{Q} -values = [sum([prob * (γ^t * gamma * value-table[s][a])
for prob, s, r, a in env.P[s][a]])]

for prob, s, r, a in env.P[s][a]]])

for a in range (env.action_space.n)]

return policy

$\text{policy}[s] = \text{np.argmax}(\text{np.array}(\hat{Q}\text{-values}))$

Pg: (25)

flow diagram..

value-itr(enr)

main()

enr = - - -

optimal = value-itr(enr)

optimal-value = Value-iteration(enr)
for

optimal-policy = extract-policy(optimal-value-tr)
print(optimal-policy)

def value-iteration(enr) :
 return value-table
 def extract-policy(value-table) :
 return policy

main()

// initialize the enr

Policy iteration algorithm

- LO1: Describe the policy iteration algorithm to find an optimal policy
- LO2: Given the model dynamics of a particular state in a RL environment, apply the policy iteration algorithm, to find an optimal policy
- LO3: Differentiate the RL environments where DP can/cannot be applied.



Policy Iteration Method:-

→ compute the optimal value f_n , using

in the value its method:

1. Compute the optimal value f_n by taking the max over the α for (α values)
2. Extract the optimal policy from the optimal value f_n , got in step 1.

in the policy its method:

1. Compute the optimal value f_n iteratively, by using the policy.
2. Extract the optimal policy from the optimal value f_n , got in step 1. [this is the same policy that generated the optimal value f_n]

How to find the value fn of a state for a given policy π ?

If π is stochastic:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a) [R(s,a,s') + \gamma V^\pi(s')]$$

If π is deterministic:

$$V^\pi(s) = \sum_{s'} P(s'|s,\pi) [R(s,\pi,s') + \gamma V^\pi(s')]$$

$$(P1) \quad V^\pi(s) = \sum_{s'} P_{ss'}^{\pi} [R_{ss'}^{\pi} + \gamma V^\pi(s')]$$

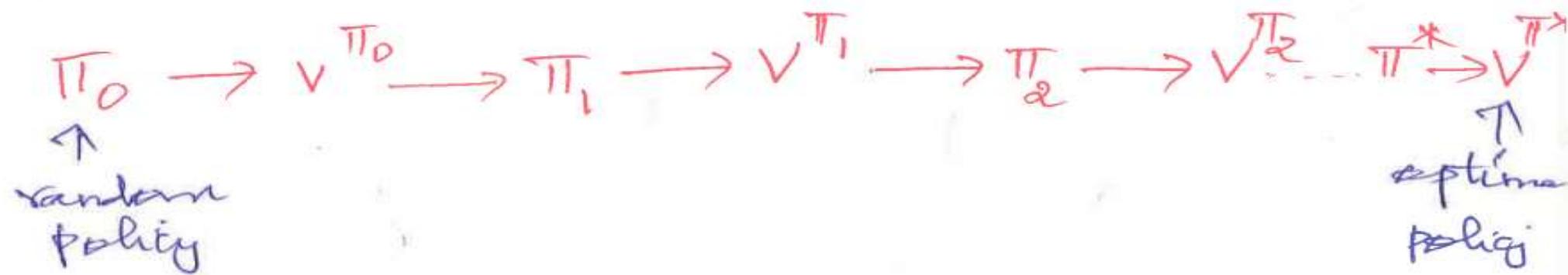
P₁ (P1)

Assume the policy is deterministic:

1. Start with a random policy π_0 .
2. Initialize the value v_{π_0} (table) of all states to zeros.
3. Use π_0 , to find an optimal value table iteratively v^{π_0} . [This will not be optimal ~~policy~~^{value} table, since it is generated from a random policy]
4. Use v^{π_0} to generate a policy π_1 .

5. compute V^{π_1} using π_1 . check if V^{π_1} is optimal. If so, stop. Else generate π_2 from V^{π_1} .

6. Use π_2 to generate a V^{π_2} . If V^{π_2} is optimal stop. Else generate a new policy π_3 and so on.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



How to decide the value function is optimal?

How to decide that the value f_π is optimal?

If it doesn't change over iterations?

Since the value f_π is generated from a policy, over a series of iterations, if the policy doesn't change b/w 2 consecutive iterations, then it is an optimal policy π^* .

The value f_π generated from π^* is the optimal value f_{π^*} . V^{π^*}



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Bellman - code :-

policy = random-policy

for i in range (num-itrns):

 value-fn = compute-value-fn(policy)

 new-policy = extract-policy (value-fn)

 if policy == new-policy :

 break

 else

 policy = new-policy.



UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

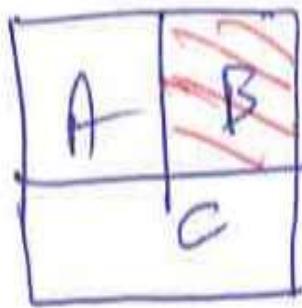


Algorithm for Policy Iteration..

1. Initialize a random policy
2. Compute a value function iteratively, from this policy
3. Extract a new policy using the value function got from step 2.
4. If the extracted policy is same as the policy used in step 2, then stop; else send the extracted new policy to step 2 and repeat steps 2 to 4.

An example for Policy iteration

Ex: [Pg: 118]



states:	A	-	0
	B	-	1
	C	-	2

Actions: 0-left/right; 1-up/down.

Goal: from A, reach C, without visiting B.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Given model dynamics of State A.

State s	Action a	Next state s'	$P_{ss'}^a$	$R_{ss'}^a$
A	0	A - -	0.1 -	0
A	0	B - -	0.8 -	-1
A	0	C - -	0.1	1
A	1	A - -	0.1	0
A	1	B - -	0.0	-1
A	1	C - -	0.9	1

Step1:- Initialize a random policy:-

$$A \rightarrow 1; B \rightarrow 0; C \rightarrow 1.$$

Step2: Compute the value for using this random policy

$$V^{\pi}(s) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')]$$

Step3: Initialize the value table of all states to zero.

state	value
A	0
B	0
C	0

Initial value table

$$\gamma = 1$$

step 9_{th}: find the $V(s)$ for a state, only for those actions mentioned in the policy; π .

$\therefore V(A)$ - only for action 1.

$$\begin{aligned}\therefore V(A) &= P_{AA}^1 [R_{AA}^1 + \gamma V(A)] + P_{AB}^1 [R_{AB}^1 + \gamma V(B)] \\ &\quad + P_{AC}^1 [R_{AC}^1 + \gamma V(C)] \\ &= 0.1[0+0] + 0[-1+0] + 0.9[1+0] \\ &= 0.9\end{aligned}$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



(ii) Early from the model dynamics of states B and C, find $V(B)$ for action 0 and $V(C)$ for action 1.

Assume the value table from itn1 is
This will not be optimal.

State	Value
A	0.9
B	-0.2
C	0.1

Ques: find $V(S)$ using the value table of from

Ques: ~~the P~~

$$\begin{aligned}
 V(A) &= P_{AA}^{-1} \left[R_{AA}^{-1} + \gamma V(A) \right] + P_{AB}^{-1} \left[R_{AB}^{-1} + \gamma V(B) \right] \\
 &\quad + P_{AC}^{-1} \left[R_{AC}^{-1} + \gamma V(C) \right] \\
 &= 0.1 [0 + 0.9] + 0 [-1 - 0.2] + 0.9 [1 + 0.1] \\
 &= 1.08
 \end{aligned}$$

Clearly keep & find $V(B)$ and $V(C)$. Assume
value table from itn₂ is

state	value
A	1.08
B	-0.5
C	0.5

Value table from itn₃ is

A -	1.45
B	-0.9
C -	0.6

Value table from itn₄ is

A	1.45
B	-0.9
C -	0.6

No change the optimal

final value table \leftarrow ie
from the



UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013



Step 3: Extract a new policy using this value θ_j from step 2:

∴ Policy ~~is generated~~ tells us which is the correct action in each state. This is given by

$$\pi(s, a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$

for all (s, a) pairs of a state s .

for state A:-

$$\begin{aligned} \text{Find } \pi(A, 0) : & P_{AA}^0 [R_{AA}^0 + \gamma V(A)] + P_{AB}^0 [R_{AB}^0 + \\ & + P_{AC}^0 [R_{AC}^0 + \gamma V(C)]] \end{aligned}$$

$$= 0.1[0 + 1 - 4b] + 0.8(-1 - 0.9) +$$

$$\pi(A, 1) : 0.1[1 + 0.61]$$

$$= -1.21.$$

$$\begin{aligned}
 V(A, 1) &= P_{AA}^{-1} [R_{AA}^{-1} + \gamma v(A)] + P_{AB}^{-1} [R_{AB}^{-1} + \gamma v(B)] \\
 &\quad + P_{AC}^{-1} [R_{AC}^{-1} + \gamma v(C)] \\
 &= 0.1[0 + 1.46] + 0.0[-1 - 0.9] + 0.9[1 + \\
 &\quad 0.6] \\
 &= 1.59
 \end{aligned}$$

(ii) Similarly do for states B and C for all actions.

∴ A table is

State	Action	Value
A	0	-1.21
	1 ✓	1.59 ✓
B	0 ✓	0.1 ✓
	1	0.0
C	0 ✓	0.5 ✓
	1	-0.0

Pg (33)

From this R-table, pick the action in each state, that gives the max. value. This gives a new policy.

$$A \rightarrow l; B \rightarrow D; C \rightarrow O.$$

Step 5: Check the new policy

If the extracted new policy from step(3) is same as the policy used in step(2), then

Step : Else goto step 2 with this new policy & repeat steps 2 to 4.

Solving the Frozen Lake Problem with policy iteration

main()

import gym

import numpy as np

env = gym.make('FrozenLake-v0')

optimal_policy = policy_itn(env)

print(optimal_policy)



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



```
def policy-itr(env):
    num-itrns = 1000
    // initialize policy to zero in all states
    Policy = np.zeros([env.observation_space.n])
    for i in range(num-itrns):
        val-fn = compute-val-fn(Policy)
        new-policy = extract-policy(val-fn)
        if (np.all(Policy == new-policy)):
            break
        Policy = new-policy
    return(Policy)
```



PRESIDENT
UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

40
YEARS
OF ACADEMIC
WISDOM

```
def compute-val-fn(policy):
    num-itrns = 1000
    threshold = 1e-20
    gamma = 1.0
    // init. val-table of all states to zero
    Val-tab = np.zeros(env.observation_space)
    for i in range(num-itrns):
        updated-val-tab = np.copy(Val-tab)
        for s s in range(env.observation-
// find value of a state only for action a[i] is
a = policy[s] the policy Prize
            space):
                pass
```



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



$$\text{II} \quad V^{\pi}(s) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')]$$

val-tab[s] = sum(

[prob * (r + gamma * updated-val-tab[s_])

for prob, s_, r, - in env.P[s][a]])

if (np.sum(np.fabs(updated-val-tab - val-tab)) <= threshold:

break

return(val-tab)

def extract-policy(val-tab):

// policy is generated using $\alpha(s,a)$, for all

// actions 'a' is a state 's'; not with resp to any π .

$$\text{// } \alpha(s,a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V[s']]$$

gamma = 1.0

// init. policy of all states to zero.

policy = np.zeros(env.observation_space.n)

for i in ~~range(n)~~:

for s in range(env.observation_space.n):

Q-values = [sum([prob * (-r + gamma *

val-tab[s-])

for prob, s-, r, - in env.P[s][a])])

for a in range(env.action_space.n)]

policy[s] = np.argmax(np.array(Q-values))

// this retains the index of max Q-value

return policy.



**THE PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

40
YEARS OF ACADEMIC
WISDOM

Limitations of DP:-

DP is a model-based method to find the optimal policy. In both value & policy it's methods, we need to know the model dynamics [trans. probabilities] and reward functions.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Summary of Value-iteration algorithm

In value iteration:-

$$V^*(s) = \max_a Q^*(s, a)$$

1. find the optimal val. fn. of a state 's', by finding the max over all Q functions (s, a) of all actions in state 's'.

$$Q(s, a) = \sum_{s'} P_{s, a} [R_{s, a} + \gamma V(s')]$$

2. Extract the optimal policy from this optimal val. fn.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Summary of Policy-iteration algorithm

In Policy iteration -

1. find the optimal val. fn of a state 's', by finding the using the policy iteratively

$$V^{\pi}(s) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')]$$

start with a random policy and find the value fn.

Pg.: (37)

2. find the policy that generated this optimal value fn, is the optimal policy.

3. in both methods, we should know $P_{ss'}^a$.

Module 2

CSE3011 Reinforcement Learning

Credit Structure : 2-2-3



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Module 2 : Monte-Carlo (MC) methods

Topics : Monte Carlo methods, prediction and control tasks, Monte Carlo prediction: algorithm, types of MC prediction, examples, incremental mean updates, Monte Carlo Control: algorithm, on-policy MC control, MC with epsilon-greedy policy, off-policy MC control. Limitations of MC method.

Introduction

- Model-free methods do not require the model dynamics of the environment to compute the value and Q functions in order to find the optimal policy.
- One such popular model-free method is the Monte Carlo (MC) method.
- The Monte Carlo method is a statistical technique used to find an approximate solution through sampling.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- For instance, the Monte Carlo method approximates the expectation of a random variable by sampling, and when the sample size is greater, the approximation will be better.
- Let's suppose we have a random variable X and say we need to compute the expected value of X; that is $E(X)$, then we can compute it by taking the sum of the values of X multiplied by their respective probabilities as follows:

$$E(X) = \sum_{i=1}^N x_i p(x_i)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- We can estimate the expected value of X by just sampling the values of X for some N times and compute the average value of X as the expected value of X as follows:

$$\mathbb{E}_{x \sim p(x)}[X] \approx \frac{1}{N} \sum_i x_i$$

- When N is larger our approximation will be better. Thus, with the Monte Carlo method, we can approximate the solution through sampling and our approximation will be better when the sample size is large.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

In reinforcement learning, we perform two important tasks, and they are:

- The prediction task
- The control task



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Prediction Task

- In the prediction task, a policy π is given as an input and we try to predict the value function or Q function using the given policy.
- Our goal is to evaluate the given policy. That is, we need to determine whether the given policy is good or bad.
- If the agent obtains a good return using the given policy then we can say that our policy is good. Thus, to evaluate the given policy, we need to understand what the return the agent would obtain if it uses the given policy. To obtain the return, we predict the value function or Q function using the given policy.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Control Task

- Unlike the prediction task, in the control task, we will not be given any policy as an input. In the control task, our goal is to find the optimal policy.
- So, we will start off by initializing a random policy and we try to find the optimal policy iteratively. That is, we try to find an optimal policy that gives the maximum return.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Monte Carlo Prediction

- First, let's recap the definition of the value function. The value function or the value of the state s can be defined as the expected return the agent would obtain starting from the state s and following the policy π . It can be expressed as:

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s]$$

Monte Carlo Prediction

In a nutshell, in the Monte Carlo prediction method, we approximate the value of a state by taking the average return of a state across N episodes instead of taking the expected return.

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



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Grid world environment example

Our goal is to reach the state I from the state A without visiting the shaded states, and the agent receives +1 reward when it visits the unshaded states and -1 reward when it visits the shaded states:

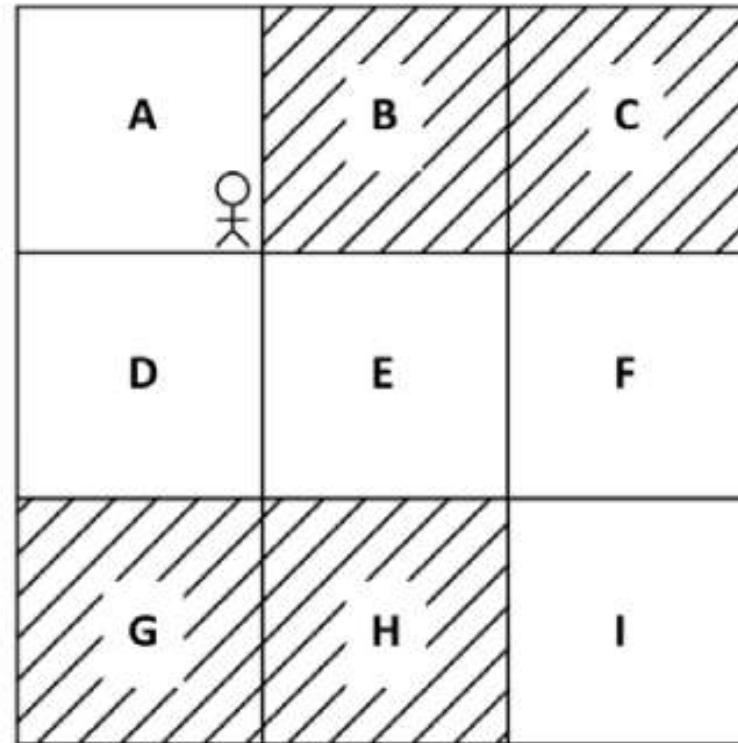


Figure 4.1: Grid world environment

Grid world environment example

- Let's say we have a stochastic policy π . Let's suppose, in state A, our stochastic policy π selects action down 80% of time and action right 20% of the time, and it selects action right in states D and E and action down in states B and F 100% of the time.
- First, we generate an episode τ_1 using our given stochastic policy π as Figure shows:

Grid world environment example

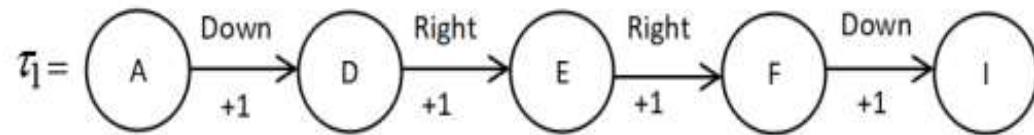
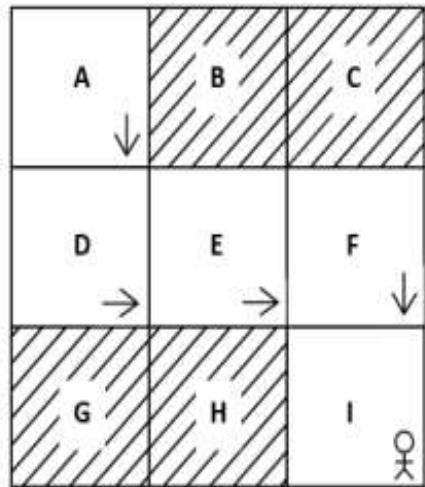


Figure 4.2: Episode τ_1

Grid world environment example

For a better understanding, let's focus only on state **A**. Let's now compute the return of state **A**. The return of a state is the sum of the rewards of the trajectory starting from that state. Thus, the return of state **A** is computed as $R_1(A) = 1+1+1+1 = 4$ where the subscript 1 in R_1 indicates the return from episode 1.

Say we generate another episode τ_2 using the same given stochastic policy π as Figure 4.3 shows:

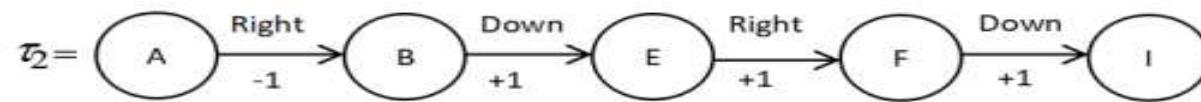
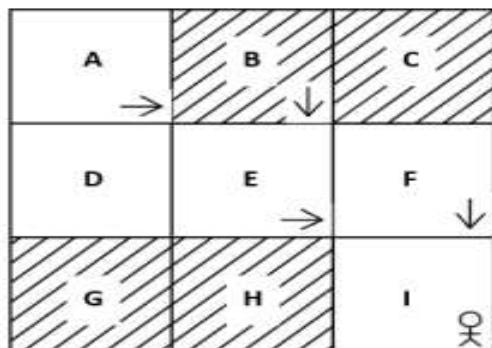


Figure 4.3: Episode τ_2

Let's now compute the return of state **A**. The return of state **A** is $R_2(A) = -1+1+1+1 = 2$.

Grid world environment example

Say we generate another episode τ_3 using the same given stochastic policy π as Figure 4.4 shows:

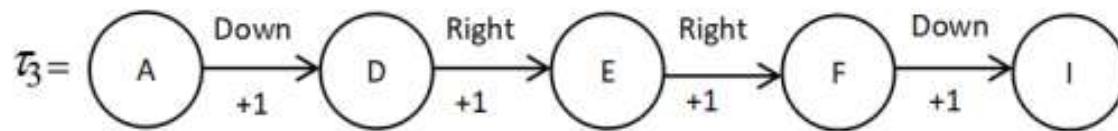
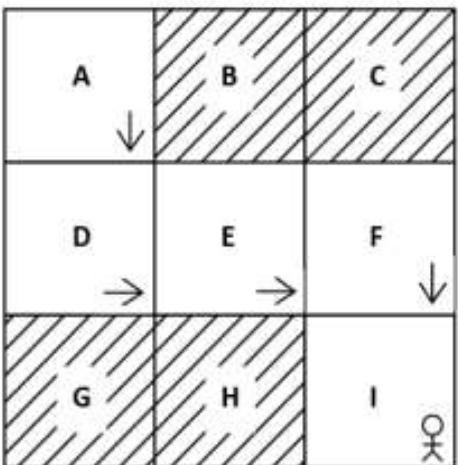


Figure 4.4: Episode τ_3

Let's now compute the return of state **A**. The return of state **A** is $R_3(A) = 1+1+1+1 = 4$.

Grid world environment example

The value of a state can be approximated by computing the average return of the state across some N episodes (trajectories):

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

We need to compute the value of state **A**, so we can compute it by just taking the average return of the state **A** across the N episodes as:

$$V(A) \approx \frac{1}{N} \sum_{i=1}^N R_i(A)$$

We generated three episodes, thus:

$$V(A) \approx \frac{1}{3} \sum_{i=1}^3 R_i(A)$$

$$V(A) = \frac{1}{3} (R_1(A) + R_2(A) + R_3(A))$$

$$V(A) = \frac{4 + 2 + 4}{3} = 3.3$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Monte Carlo Prediction

Thus, in the Monte Carlo prediction method, to predict the value of a state (value function) using the given input policy π , we generate some N episodes using the given policy and then we compute the value of a state as the average return of the state across these N episodes.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC prediction algorithm

The Monte Carlo prediction algorithm is given as follows:

1. Let $\text{total_return}(s)$ be the sum of return of a state across several episodes and $N(s)$ be the counter, that is, the number of times a state is visited across several episodes. Initialize $\text{total_return}(s)$ and $N(s)$ as zero for all the states. The policy π is given as input.
2. For M number of iterations:
 1. Generate an episode using the policy π
 2. Store all the rewards obtained in the episode in the list called rewards
 3. For each step t in the episode:
 1. Compute the return of state s_t as $R(s_t) = \text{sum}(\text{rewards}[t:])$
 2. Update the total return of state s_t as
$$\text{total_returns}(s_t) = \text{total_return}(s_t) + R(s_t)$$
 3. Update the counter as $N(s_t) = N(s_t) + 1$



MC prediction algorithm

3. Compute the value of a state by just taking the average, that is:

$$V(s) = \frac{\text{total_return}(s)}{N(s)}$$

The preceding algorithm implies that the value of the state is just the average return of the state across several episodes.

To get a better understanding of how exactly the preceding algorithm works, let's take a simple example and compute the value of each state manually. Say we need to compute the value of three states s_0 , s_1 , and s_2 . We know that we obtain a reward when we transition from one state to another. Thus, the reward for the final state will be 0 as we don't make any transitions from the final state. Hence, the value of the final state s_2 will be zero. Now, we need to find the value of two states s_0 and s_1 .



MC prediction algorithm

Step 1:

Initialize the `total_returns(s)` and $N(s)$ for all the states to zero as *Table 4.1* shows:

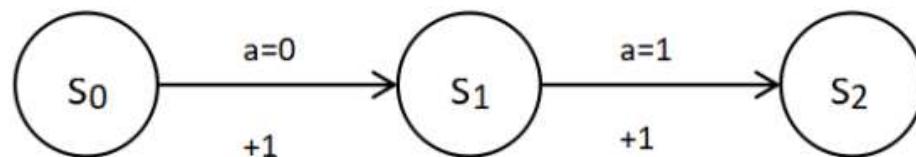
State	total_return(s)	N(s)
s_0	0	0
s_1	0	0

Table 4.1: Initial values

Say we are given a stochastic policy π ; in state s_0 our stochastic policy selects the action 0 for 50% of the time and action 1 for 50% of the time, and it selects action 1 in state s_1 for 100% of the time.

Step 2: Iteration 1:

Generate an episode using the given input policy π , as *Figure 4.5* shows:



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC prediction algorithm

Store all rewards obtained in the episode in the list called rewards. Thus,
rewards = [1, 1].

First, we compute the return of the state s_0 (sum of rewards from s_0):

$$\begin{aligned} R(s_0) &= \text{sum(rewards[0:])} \\ &= \text{sum}([1, 1]) \\ &= 2 \end{aligned}$$

Update the total return of the state s_0 in our table as:

$$\begin{aligned} \text{total_returns}(s_0) &= \text{total_returns}(s_0) + R(s_0) \\ &= 0 + 2 = 2 \end{aligned}$$

Update the number of times the state s_0 is visited in our table as:

$$\begin{aligned} N(s_0) &= N(s_0) + 1 \\ &= 0 + 1 = 1 \end{aligned}$$

Now, let's compute the return of the state s_1 (sum of rewards from s_1):

$$\begin{aligned} R(s_1) &= \text{sum(rewards[1:])} \\ &= \text{sum}([1]) \\ &= 1 \end{aligned}$$

MC prediction algorithm

Update the total return of the state s_1 in our table as:

$$\begin{aligned}\text{total_returns}(s_1) &= \text{total_returns}(s_1) + R(s_1) \\ &= 0 + 1 = 1\end{aligned}$$

Update the number of times the state s_1 is visited in our table as:

$$\begin{aligned}N(s_1) &= N(s_1) + 1 \\ &= 0 + 1 = 1\end{aligned}$$

Our updated table, after iteration 1, is as follows:

State	total_return(s)	N(s)
s_0	2	1
s_1	1	1

Table 4.2: Updated table after the first iteration

MC prediction algorithm

Iteration 2:

Say we generate another episode using the same given policy π as *Figure 4.6* shows:

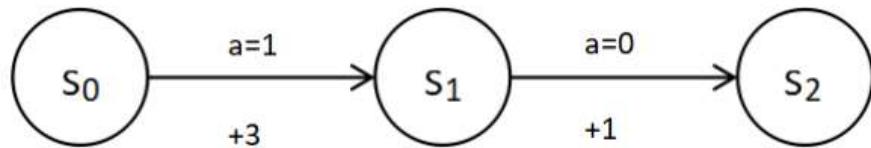


Figure 4.6: Generating an episode using the given policy π

Store all rewards obtained in the episode in the list called rewards. Thus,
rewards = [3, 1].

First, we compute the return of the state s_0 (sum of rewards from s_0):

$$\begin{aligned} R(s_0) &= \text{sum(rewards[0:])} \\ &= \text{sum}([3, 1]) \\ &= 4 \end{aligned}$$

Update the total return of the state s_0 in our table as:

$$\begin{aligned} \text{total_returns}(s_0) &= \text{total_returns}(s_0) + R(s_0) \\ &= 2 + 4 = 6 \end{aligned}$$

MC prediction algorithm

Update the number of times the state s_0 is visited in our table as:

$$\begin{aligned}N(s_0) &= N(s_0) + 1 \\&= 1 + 1 = 2\end{aligned}$$

Now, let's compute the return of the state s_1 (sum of rewards from s_1):

$$\begin{aligned}R(s_1) &= \text{sum(rewards[1:])} \\&= \text{sum([1])} \\&= 1\end{aligned}$$

Update the return of the state s_1 in our table as:

$$\begin{aligned}\text{total_returns}(s_1) &= \text{total_returns}(s_1) + R(s_1) \\&= 1 + 1 = 2\end{aligned}$$

MC prediction algorithm

Update the number of times the state is visited:

$$\begin{aligned}N(s_1) &= N(s_1) + 1 \\&= 1 + 1 = 2\end{aligned}$$

Our updated table after the second iteration is as follows:

State	total_return(s)	N(s)
s ₀	6	2
s ₁	2	2

Table 4.3: Updated table after the second iteration

Since we are computing manually, for simplicity, let's stop at two iterations; that is, we just generate only two episodes.



MC prediction algorithm

Step 3:

Now, we can compute the value of the state as:

$$V(s) = \frac{\text{total_returns}(s)}{N(s)}$$

Thus:

$$V(s_0) = \frac{\text{total_return}(s_0)}{N(s_0)} = \frac{6}{2} = 3$$

$$V(s_1) = \frac{\text{total_return}(s_1)}{N(s_1)} = \frac{2}{2} = 1$$

Thus, we computed the value of the state by just taking the average return across multiple episodes. Note that in the preceding example, for our manual calculation, we just generated two episodes, but for a better estimation of the value of the state, we generate several episodes and then we compute the average return across those episodes (not just 2).



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Types of MC prediction

- First-visit Monte Carlo
- Every-visit Monte Carlo



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



First-visit Monte Carlo

- In the first-visit Monte Carlo method, if the same state is visited again in the same episode, we don't compute the return for that state again.
- For example, consider a case where an agent is playing snakes and ladders. If the agent lands on a snake, then there is a good chance that the agent will return to a state that it had visited earlier. So, when the agent revisits the same state, we don't compute the return for that state for the second time.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



First-visit Monte Carlo Algorithm

The following shows the algorithm of first-visit MC; as the point in bold says, we compute the return for the state s_t , only if it is occurring for the first time in the episode:

1. Let $\text{total_return}(s)$ be the sum of return of a state across several episodes and $N(s)$ be the counter, that is, the number of times a state is visited across several episodes. Initialize $\text{total_return}(s)$ and $N(s)$ as zero for all the states. The policy π is given as input
2. For M number of iterations:
 1. Generate an episode using the policy π
 2. Store all the rewards obtained in the episode in the list called rewards
 3. For each step t in the episode:
 - If s_t has been visited before, then skip this step.
 - Otherwise, compute the return for the state s_t as $\text{return}_t = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{T-t} r_T$.
 - Update $\text{total_return}(s_t) \leftarrow \text{total_return}(s_t) + \text{return}_t$ and $N(s_t) \leftarrow N(s_t) + 1$.



First-visit Monte Carlo Algorithm

If the state s_t is occurring for the first time in the episode:

1. Compute the return of the state s_t as $R(s_t) = \text{sum}(\text{rewards}[t:])$
 2. Update the total return of the state s_t as $\text{total_return}(s_t) = \text{total_return}(s_t) + R(s_t)$
 3. Update the counter as $N(s_t) = N(s_t) + 1$
3. Compute the value of a state by just taking the average, that is:

$$V(s) = \frac{\text{total_return}(s)}{N(s)}$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Every-visit Monte Carlo Algorithm

Here, we compute the return every time a state is visited in the episode. The algorithm of every-visit Monte Carlo is the same as the one we saw earlier at the beginning of this section and it is as follows:

1. Let $\text{total_return}(s)$ be the sum of the return of a state across several episodes and $N(s)$ be the counter, that is, the number of times a state is visited across several episodes. Initialize $\text{total_return}(s)$ and $N(s)$ as zero for all the states. The policy π is given as input

Every-visit Monte Carlo Algorithm

2. For M number of iterations:
 1. Generate an episode using the policy π
 2. Store all the rewards obtained in the episode in the list called rewards
 3. For each step t in the episode:
 1. Compute the return of the state s_t as $R(s_t) = \text{sum}(\text{rewards}[t:])$
 2. Update the total return of the state s_t as $\text{total_return}(s_t) = \text{total_return}(s_t) + R(s_t)$
 3. Update the counter as $N(s_t) = N(s_t) + 1$
 3. Compute the value of a state by just taking the average, that is:

$$V(s) = \frac{\text{total_return}(s)}{N(s)}$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Remember that the only difference between the first-visit MC and every-visit MC methods is that in the first-visit MC method, we compute the return for a state only for its first time of occurrence in the episode but in the every-visit MC method, the return of the state is computed every time the state is visited in an episode.



Incremental mean updates

- In both first-visit MC and every-visit MC, we estimate the value of a state as an average (arithmetic mean) return of the state across several episodes as shown as follows:

$$V(s) = \frac{\text{total_return}(s)}{N(s)}$$

Incremental mean updates

- Instead of using the arithmetic mean to approximate the value of the state, we can also use the incremental mean, and it is expressed as:

$$N(s_t) = N(s_t) + 1$$

$$V(s_t) = V(s_t) + \frac{1}{N(s_t)} (R_t - V(s_t))$$

Incremental mean updates

- Consider our environment as non-stationary. In that case, we don't have to take the return of the state from all the episodes and compute the average.
- As the environment is non-stationary we can ignore returns from earlier episodes and use only the returns from the latest episodes for computing the average. Thus, we can compute the value of the state using the incremental mean as shown as follows:

Incremental mean updates

$$V(s_t) = V(s_t) + \alpha(R_t - V(s_t))$$

Where $\alpha = 1/N(s_t)$ and R_t is the return of the state s_t .



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC prediction (Q function)

- We generate several episodes using the given policy π , then, we calculate the `total_return(s, a)`, the sum of the return of the state-action pair across several episodes.
- We calculate $N(s, a)$, the number of times the state-action pair is visited across several episodes. Then we compute the Q function or Q value as the average return of the state-action pair as shown as follows:

$$Q(s, a) = \frac{\text{total_return}(s, a)}{N(s, a)}$$

MC prediction (Q function)

For instance, let consider a small example. Say we have two states s_0 and s_1 and we have two possible actions 0 and 1. Now, we compute $\text{total_return}(s, a)$ and $N(s, a)$. Let's say our table after computation looks like Table 4.4:

State	Action	$\text{total_return}(s,a)$	$N(s,a)$
s_0	0	4	2
s_0	1	2	2
s_1	0	2	2
s_1	1	2	1

Table 4.4: The result of two actions in two states



MC prediction (Q function)

- Thus, we can compute the Q value for all state-action pairs as:

$$Q(s_0, 0) = \text{total_return}(s_0, 0)/N(s_0, 0) = 4/2 = 2$$

$$Q(s_0, 1) = \text{total_return}(s_0, 1)/N(s_0, 1) = 2/2 = 1$$

$$Q(s_1, 0) = \text{total_return}(s_1, 0)/N(s_1, 0) = 2/2 = 1$$

$$Q(s_1, 1) = \text{total_return}(s_1, 1)/N(s_1, 1) = 2/1 = 2$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC prediction (Q function) algorithm

The algorithm for predicting the Q function using the Monte Carlo method is as follows.

1. Let $\text{total_return}(s, a)$ be the sum of the return of a state-action pair across several episodes and $N(s, a)$ be the number of times a state-action pair is visited across several episodes. Initialize $\text{total_return}(s, a)$ and $N(s, a)$ for all state-action pairs to zero. The policy π is given as input

MC prediction (Q function) algorithm

2. For M number of iterations:
 1. Generate an episode using policy π
 2. Store all rewards obtained in the episode in the list called rewards
 3. For each step t in the episode:
 1. Compute return for the state-action pair, $R(s_t, a_t) = \text{sum}(\text{rewards}[t:])$
 2. Update the total return of the state-action pair,
 $\text{total_return}(s_t, a_t) = \text{total_return}(s_t, a_t) + R(s_t, a_t)$
 3. Update the counter as $N(s_t, a_t) = N(s_t, a_t) + 1$
 3. Compute the Q function (Q value) by just taking the average, that is:

$$Q(s, a) = \frac{\text{total_return}(s, a)}{N(s, a)}$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC prediction of the Q function

- We have two types of MC— first-visit MC and every-visit MC.
- In first-visit MC, we compute the return of the state-action pair only for the first time the state-action pair is visited in the episode and in every-visit MC we compute the return of the state-action pair every time the state-action pair is visited in the episode.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Incremental mean

We can compute the Q value using the incremental mean as shown as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t - Q(s_t, a_t))$$

Monte Carlo control

- In the control task, our goal is to find the optimal policy. Unlike the prediction task, here, we will not be given any policy as an input. So, we will begin by initializing a random policy, and then we try to find the optimal policy iteratively. That is, we try to find an optimal policy that gives the maximum return.
- That is, if we have a Q function, then we can extract policy by selecting an action in each state that has the maximum Q value as the following shows:

$$\pi = \arg \max_a Q(s, a)$$

Monte Carlo control

From the new Q function, we extract a new policy. We repeat these steps iteratively until we find the optimal policy.

Iteration 1—Let π_0 be the random policy. We use this random policy to generate an episode, and then we compute the Q function Q^{π_0} by taking the average return of the state-action pair. Then, from this Q function Q^{π_0} , we extract a new policy π_1 . This new policy π_1 will not be an optimal policy since it is extracted from the Q function, which is computed using the random policy.

Iteration 2—So, we use the new policy π_1 derived from the previous iteration to generate an episode and compute the new Q function Q^{π_1} as average return of a state-action pair. Then, from this Q function Q^{π_1} , we extract a new policy π_2 . If the policy π_2 is optimal we stop, else we go to iteration 3.



**PRESIDENCY
UNIVERSITY**



Private University Estd. in Karnataka State by Act No. 41 of 2013

Monte Carlo control

Iteration 3—Now, we use the new policy π_2 derived from the previous iteration to generate an episode and compute the new Q function Q^{π_2} . Then, from this Q function Q^{π_2} , we extract a new policy π_3 . If π_3 is optimal we stop, else we go to the next iteration.

We repeat this process for several iterations until we find the optimal policy π^* as shown in *Figure 4.21*:

$$\pi_0 \rightarrow Q^{\pi_0} \rightarrow \pi_1 \rightarrow Q^{\pi_1} \rightarrow \pi_2 \rightarrow Q^{\pi_2} \rightarrow \pi_3 \rightarrow Q^{\pi_3} \rightarrow \dots \rightarrow \pi^* \rightarrow Q^{\pi^*}$$

Figure 4.21: The path to finding the optimal policy



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC control algorithm

- Once we have the Q function, we extract a new policy by selecting an action in each state that has the maximum Q value. In the next iteration, we use the extracted new policy to generate an episode and compute the new Q function (Q value) as the average return of the state-action pair.
- We repeat these steps for many iterations to find the optimal policy.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC control algorithm

- One more thing, we need to observe that just as we learned in the first-visit MC prediction method, here, we compute the return of the state-action pair only for the first time a state-action pair is visited in the episode



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC control algorithmic steps

1. Let $\text{total_return}(s, a)$ be the sum of the return of a state-action pair across several episodes and $N(s, a)$ be the number of times a state-action pair is visited across several episodes. Initialize $\text{total_return}(s, a)$ and $N(s, a)$ for all state-action pairs to zero and initialize a random policy π
2. For M number of iterations:
 1. Generate an episode using policy π
 2. Store all rewards obtained in the episode in the list called rewards



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC control algorithmic steps

3. For each step t in the episode:

If (s_t, a_t) is occurring for the first time in the episode:

1. Compute the return of a state-action pair, $R(s_t, a_t) = \text{sum}(\text{rewards}[t:])$
2. Update the total return of the state-action pair as,
 $\text{total_return}(s_t, a_t) = \text{total_return}(s_t, a_t) + R(s_t, a_t)$
3. Update the counter as $N(s_t, a_t) = N(s_t, a_t) + 1$
4. Compute the Q value by just taking the average, that is,

$$Q(s_t, a_t) = \frac{\text{total_return}(s_t, a_t)}{N(s_t, a_t)}$$

4. Compute the new updated policy π using the Q function:

$$\pi = \arg \max_a Q(s, a)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC Control methods

- On-policy control—In the on-policy control method, the agent behaves using one policy and also tries to improve the same policy. That is, in the on-policy method, we generate episodes using one policy and also improve the same policy iteratively to find the optimal policy. For instance, the MC control method, which we just learned above, can be called on-policy MC control as we are generating episodes using a policy π , and we also try to improve the same policy π on every iteration to compute the optimal policy.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC Control methods

- Off-policy control—In the off-policy control method, the agent behaves using one policy b and tries to improve a different policy π . That is, in the off-policy method, we generate episodes using one policy and we try to improve the different policy iteratively to find the optimal policy.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



on-policy Monte Carlo control

There are two types of on-policy Monte Carlo control methods:

- Monte Carlo exploring starts
- Monte Carlo with the epsilon-greedy policy



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Monte Carlo exploring starts algorithm

1. Let $\text{total_return}(s, a)$ be the sum of the return of a state-action pair across several episodes and $N(s, a)$ be the number of times a state-action pair is visited across several episodes. Initialize $\text{total_return}(s, a)$ and $N(s, a)$ for all state-action pairs to zero and initialize a random policy π
2. For M number of iterations:
 1. **Select the initial state s_0 and initial action a_0 randomly such that all state-action pairs have a probability greater than 0**
 2. Generate an episode from the selected initial state s_0 and action a_0 using policy π
 3. Store all the rewards obtained in the episode in the list called rewards



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Monte Carlo exploring starts algorithm

4. For each step t in the episode:

If (s_t, a_t) is occurring for the first time in the episode:

1. Compute the return of a state-action pair, $R(s_t, a_t) = \text{sum}(\text{rewards}[t:])$
2. Update the total return of the state-action pair as $\text{total_return}(s_t, a_t) = \text{total_return}(s_t, a_t) + R(s_t, a_t)$
3. Update the counter as $N(s_t, a_t) = N(s_t, a_t) + 1$
4. Compute the Q value by just taking the average, that is,

$$Q(s_t, a_t) = \frac{\text{total_return}(s_t, a_t)}{N(s_t, a_t)}$$

5. Compute the updated policy π using the Q function:

$$\pi = \arg \max_a Q(s, a)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Monte Carlo exploring starts algorithm

- One of the major drawbacks of the exploring starts method is that it is not applicable to every environment. That is, we can't just randomly choose any state-action pair as an initial state-action pair because in some environments there can be only one state-action pair that can act as an initial state-action pair. So we can't randomly select the state-action pair as the initial state-action pair.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Monte Carlo exploring starts algorithm

- For example, suppose we are training an agent to play a car racing game; we can't start the episode in a random position as the initial state and a random action as the initial action because we have a fixed single starting state and action as the initial state and action.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Monte Carlo with the epsilon-greedy policy

- So, now the question is whether the agent should explore all the other actions in the state and select the best action as the one that has the maximum Q value or exploit the best action out of already-explored actions. This is called an exploration-exploitation dilemma.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Monte Carlo with the epsilon-greedy policy

- To avoid this dilemma, we introduce a new policy called the epsilon-greedy policy. Here, all actions are tried with a non-zero probability (ϵ). With a probability ϵ , we explore different actions randomly and with a probability $1-\epsilon$, we choose an action that has the maximum Q value.
- That is, with a probability ϵ , we select a random action (exploration) and with a probability $1-\epsilon$ we select the best action (exploitation).

Monte Carlo with the epsilon-greedy policy

- In the epsilon-greedy policy, if we set the value of epsilon to 0, then it becomes a greedy policy (only exploitation).
- when we set the value of epsilon to 1, then we will always end up doing only the exploration.
- So, the value of epsilon has to be chosen optimally between 0 and 1.

Monte Carlo with the epsilon-greedy policy

- Say we set epsilon = 0.5; then we will generate a random number from the uniform distribution and if the random number is less than epsilon (0.5), then we select a random action (exploration).
- if the random number is greater than or equal to epsilon then we select the best action, that is, the action that has the maximum Q value (exploitation).

Monte Carlo with the epsilon-greedy policy

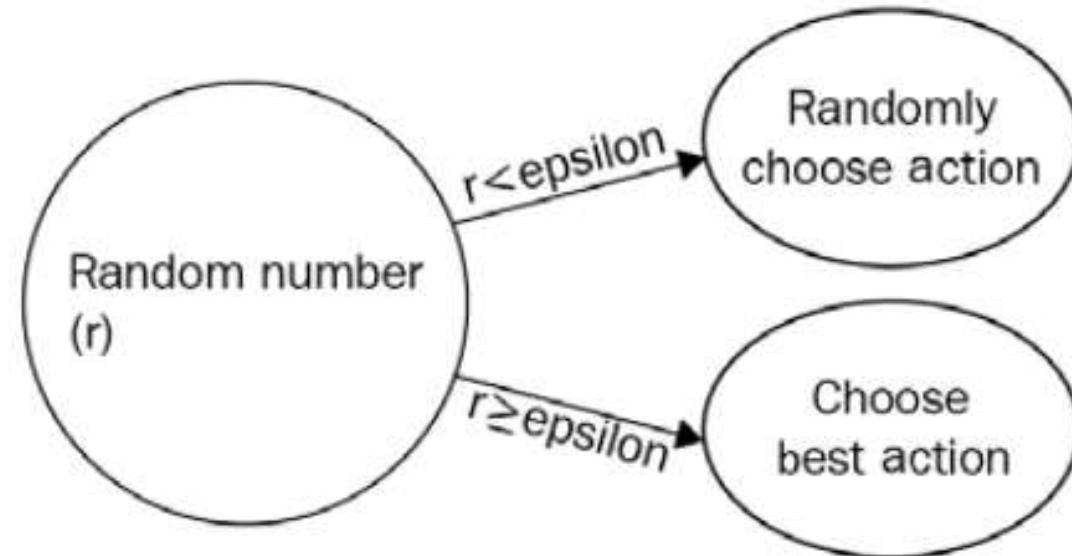


Figure 4.22: Epsilon-greedy policy



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC control algorithm with the epsilon-greedy policy

The MC control algorithm with the epsilon-greedy policy

The algorithm of Monte Carlo control with the epsilon-greedy policy is essentially the same as the MC control algorithm we learned earlier except that here we select actions based on the epsilon-greedy policy to avoid the exploration-exploitation dilemma. The following steps show the algorithm of Monte Carlo with the epsilon-greedy policy:

1. Let $\text{total_return}(s, a)$ be the sum of the return of a state-action pair across several episodes and $N(s, a)$ be the number of times a state-action pair is visited across several episodes. Initialize $\text{total_return}(s, a)$ and $N(s, a)$ for all state-action pairs to zero and initialize a random policy π



**PRESIDENCY
UNIVERSITY**



Private University Estd. in Karnataka State by Act No. 41 of 2013

MC control algorithm with the epsilon-greedy policy

2. For M number of iterations:

1. Generate an episode using policy π
2. Store all rewards obtained in the episode in the list called rewards
3. For each step t in the episode:
If (s_t, a_t) is occurring for the first time in the episode:
 1. Compute the return of a state-action pair, $R(s_t, a_t) = \text{sum}(\text{rewards}[t:])$
 2. Update the total return of the state-action pair as $\text{total_return}(s_t, a_t) = \text{total_return}(s_t, a_t) + R(s_t, a_t)$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC control algorithm with the epsilon-greedy policy

3. Update the counter as $N(s_t, a_t) = N(s_t, a_t) + 1$
4. Compute the Q value by just taking the average, that is,

$$Q(s_t, a_t) = \frac{\text{total_return}(s_t, a_t)}{N(s_t, a_t)}$$

4. Compute the updated policy π using the Q function. Let $a^* = \arg \max_a Q(s, a)$. The policy π selects the best action a^* with probability $1 - \epsilon$ and random action with probability ϵ

As we can observe, in every iteration, we generate the episode using the policy π and also we try to improve the same policy π in every iteration to compute the optimal policy.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Off-policy Monte Carlo control

- Off-policy Monte Carlo is another interesting Monte Carlo control method. In the off-policy method, we use two policies called the behavior policy and the target policy. As the name suggests, we behave (generate episodes) using the behavior policy and we try to improve the other policy called the target policy.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Off-policy Monte Carlo control

In the on-policy method, we generate an episode using the policy π and we improve the same policy π iteratively to find the optimal policy. But in the off-policy method, we generate an episode using a policy called the behavior policy b and we try to iteratively improve a different policy called the target policy π .



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Off-policy Monte Carlo control algorithm

The algorithm is given as follows:

1. Initialize the Q function $Q(s, a)$ with random values, set the behavior policy b to be epsilon-greedy, and also set the target policy π to be greedy policy.
2. For M number of episodes:
 1. Generate an episode using the behavior policy b
 2. Initialize return R to 0



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Off-policy Monte Carlo control algorithm

3. For each step t in the episode, $t = T-1, T-2, \dots, 0$:
 1. Compute the return as $R = R + r_{t+1}$
 2. Compute the Q value as $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t - Q(s_t, a_t))$
 3. Compute the target policy $\pi(s_t) = \arg \max_a Q(s_t, a)$
3. Return the target policy π



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Need of importance sampling

- We are finding the target policy π from the Q function, which is computed based on the episodes generated by a different policy called the behavior policy, our target policy will be inaccurate. This is because the distribution of the behavior policy and the target policy will be different.
- So, to correct this, we introduce a new technique called importance sampling. This is a technique for estimating the values of one distribution when given samples from another.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Importance sampling types:

- Ordinary importance sampling

Here, the importance sampling ratio will be the ratio of the target policy to the behavior policy $\frac{\pi(a|s)}{b(a|s)}$

- Weighted importance sampling

Here, the importance sampling ratio will be the weighted ratio of the target policy to the behavior policy $w \frac{\pi(a|s)}{b(a|s)}$.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Off-policy Monte Carlo control algorithm

The complete algorithm of the off-policy Monte Carlo method is explained in the following steps:

1. Initialize the Q function $Q(s, a)$ with random values, set the behavior policy b to be epsilon-greedy, and target policy π to be greedy policy and initialize the cumulative weights as $C(s, a) = 0$
2. For M number of episodes:
 1. Generate an episode using the behavior policy b
 2. Initialize return R to 0 and weight W to 1
 3. For each step t in the episode, $t = T-1, T-2, \dots, 0$:
 1. Compute the return as $R = R + r_{t+1}$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Off-policy Monte Carlo control algorithm

2. Update the cumulative weights $C(s_t, a_t) = C(s_t, a_t) + W$

3. Update the Q value as

$$Q(s_t, a_t) = Q(s_t, a_t) + \frac{W}{C(s_t, a_t)}(R_t - Q(s_t, a_t))$$

4. Compute the target policy $\pi(s_t) = \arg \max_a Q(s_t, a)$

5. If $a_t \neq \pi(s_t)$ then break

6. Update the weight as $W = W \frac{1}{b(a_t | s_t)}$

3. Return the target policy π



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



MC limitation

- Issue with the Monte Carlo method is that it is applicable only to episodic tasks. We learned that in the Monte Carlo method, we compute the value of the state by taking the average return of the state and the return is the sum of rewards of the episode.
- But when there is no episode, that is, if our task is a continuous task (non-episodic task), then we cannot apply the Monte Carlo method.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Module 3

Temporal Difference (TD)

Learning

Slides prepared by Dr J Alamelu Mangai

- LO1 : Understand MC method
- LO2 : Understand 2 types of RL tasks : prediction and control
- LO3 : Understand the advantages of MC over DP methods
- LO4: Understand the different types of MC prediction
- LO5: Apply MC prediction to train an agent to play the blackjack game
- LO6: Understand the different types of MC control
- LO7: Apply MC control to train an agent to play the blackjack game

Slides prepared by Dr J Alamelu Mangai

Introduction

- TD is a popular model-free method
- It combines the advantages of DP and MC methods
- A recap of pros and cons of DP and MC methods

Dynamic Programming (DP)

Advantages :

- Uses Bellman equation to find the value of a state as

$$V(s) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$

- We don't have to wait till the end of an episode to find the value of a state.
- We can estimate the value of a state, just based on the value of the next state - **boot strapping**

Disadvantages:

- DP can be used only when the model dynamics is known

Slides prepared by Dr J Alamelu Mangai

Monte-Carlo (MC)

Advantages :

- It is a model free method.
- We don't require the model dynamics to estimate the value and Q functions.

Disadvantages:

- To find $V(s)$ or $Q(s,a)$, we need to wait till the end of the episode.
- If the episode is too long, it will cost us a lot of time.
- MC methods cannot be applied to continuous tasks/non-episodic tasks, without a terminal state.

Slides prepared by Dr J Alamelu Mangai

Temporal Difference (TD) learning

- We use boot-strapping, and don't have to wait till the end of an episode to find $V(s)$ or $Q(s,a)$
- Like MC it is a model-free method.
- Two categories of TD learning : **TD-prediction and TD-control**
- **TD- prediction :**
 - A policy is given as input and we try to predict the $V(s)$ and $Q(s,a)$ using this policy
 - Helps the agent to understand how good it is for the agent to be in each state, if it uses the given policy
 - The agent can estimate the expected return of each state, if it uses the given policy in that state.

Slides prepared by Dr J Alamelu Mangai

- TD-control:
- We are not given any policy as input, but the goal is to find an optimal policy
- We initialize a random policy and we try to find the optimal policy iteratively.
- This optimal policy will give the maximum return

Slides prepared by Dr J Alamelu Mangai

TD-Prediction

- A policy is given as input and we try to estimate the value function of each state $V(s)$, using the given policy
- TD uses bootstrapping like DP, hence we don't have to wait till the end of an episode to find $V(s)$
- Like MC, it doesn't require the model dynamics to find $V(s)$ and $Q(s,a)$.
- The update rule of TD takes these advantages into account.
- In MC method,
$$V(s) \approx R(s)$$
- Since a single return cannot approximate $V(s)$ perfectly, we take the mean of the return over N episodes,

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

Slides prepared by Dr J Alamelu Mangai

- TD learning uses bootstrapping and doesn't wait till the end of an episode, to find the value of a state as

$$V(s) \approx r + \gamma V(s')$$

- It doesn't use the model dynamics.
- In MC, since a single value $V(s)$ cannot approximate the value of a state perfectly, we took the incremental mean

$$V(s) = V(s) + \alpha(R - V(s))$$

- In TD, we use the TD-learning update rule

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

Slides prepared by Dr J Alamelu Mangai

Difference between MC and TD

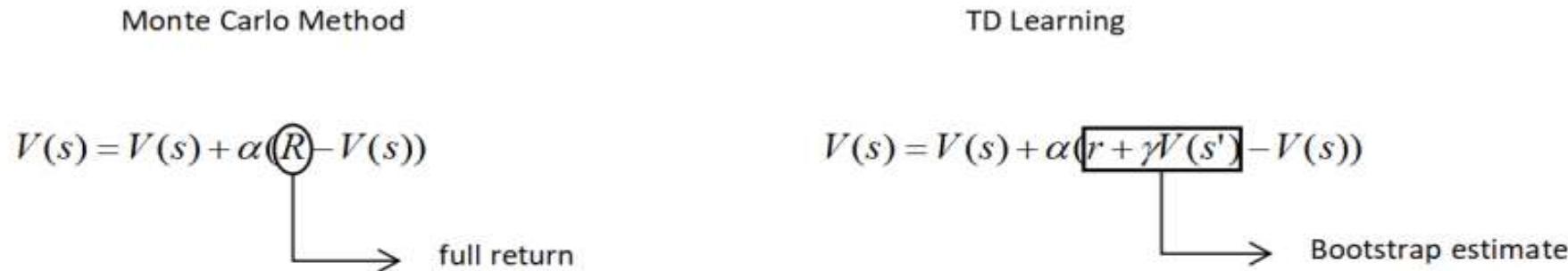


Figure 5.1: A comparison between MC and TD learning

Slides prepared by Dr J Alamelu Mangai

- $r + \gamma V(s')$ is an estimate of the value of a state – called – TD target.
- Hence $(r + \gamma V(s')) - V(s)$ is (target – predicted) – called the TD error.

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

\downarrow \downarrow
 Learning rate TD error

Our TD learning update rule basically implies:

Value of a state = value of a state + learning rate (reward + discount factor(value of next state) - value of a state)

Slides prepared by Dr J Alamelu Mangai

TD-Prediction using TD-learning

- A policy is taken as input, using the update rule of TD-learning, $V(s)$ is updated.
- Finally we get the expected return an agent can obtain in each state, if it acts according to the given policy
- The update rule of TD-learning is

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

Slides prepared by Dr J Alamelu Mangai

TD-Prediction in FZLE

- States are represented as numbers as the first state S is denoted by (1,1) and the second state F is denoted by (1,2) and so on to the last state G, which is denoted by (4,4).
- the goal of the agent is to reach the goal state G from the starting state S without visiting the hole states H
- Reward is 1, if the agent reaches the goal state, else reward is 0
- Actions : left, right, up, down

	1	2	3	4
1	S ♀	F	F	F
2	F	H	F	H
3	F	F	F	H
4	H	F	F	G

Dr J Alamelu Mangai

- Take a policy as input and evaluate $V(s)$ using this policy
- Assume input policy is

State	Action
(1,1)	Right
(1,2)	Right
(1,3)	Left
:	:
(4,4)	Down

- Find $V(s)$ for this input policy using TD-learning

Slides prepared by Dr J Alamelu Mangai

- Initialize the value of all states to some random values.

State	Value
(1,1)	0.9
(1,2)	0.6
(1,3)	0.8
:	:
(4,4)	0.7

- Step 1: Current state $s = (1,1)$ $a = \text{right}$ as per the input policy
 next state $s' = (1,2)$
 update $V((1,1))$ using TD -learning update rule as

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

Assume $\gamma = 1$ and $\alpha = 0.1$

Slides prepared by Dr J Alamelu Mangai

Substituting the value of state $V(s)$ with $V(1,1)$ and the next state $V(s')$ with $V(1,2)$ in the preceding equation, we can write:

$$V(1, 1) = V(1, 1) + \alpha(r + \gamma V(1, 2) - V(1, 1))$$

Substituting the reward $r = 0$, the learning rate $\alpha = 0.1$, and the discount factor $\gamma = 1$, we can write:

$$V(1, 1) = V(1, 1) + 0.1(0 + 1 \times V(1, 2) - V(1, 1))$$

We can get the state values from the value table shown earlier. That is, from the preceding value table, we can observe that the value of state **(1,1)** is 0.9 and the value of the next state **(1,2)** is 0.6. Substituting these values in the preceding equation, we can write:

$$V(1, 1) = 0.9 + 0.1(0 + 1 \times 0.6 - 0.9)$$

Thus, the value of state **(1,1)** becomes:

$$V(1, 1) = 0.87$$

Slides prepared by Dr J Alamelu Mangai



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



- Updated value table after step1 is

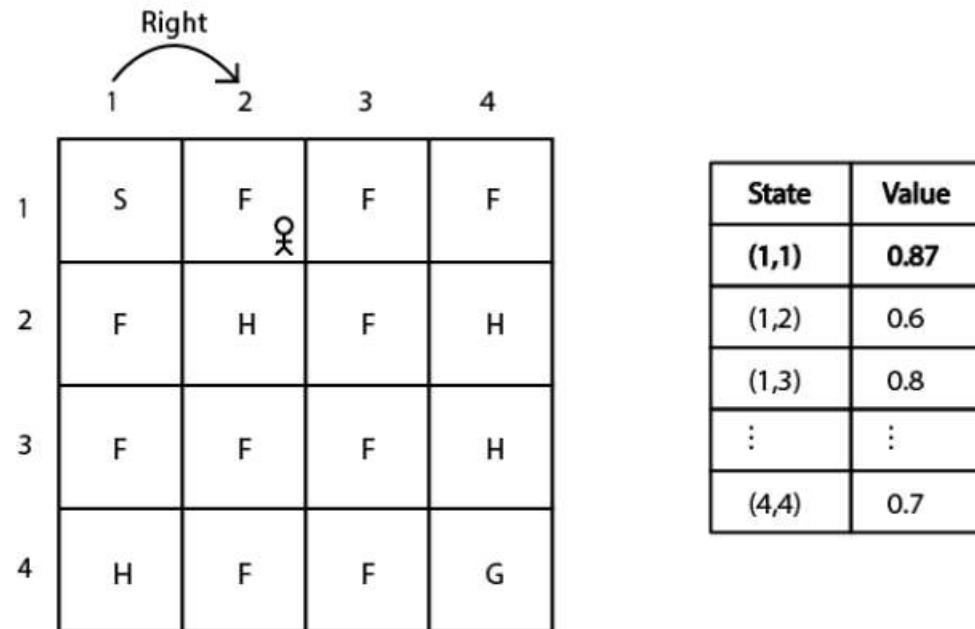


Figure 5.4: The value of state (1,1) is updated

Slides prepared by Dr J Alamelu Mangai

- Step 2 : $s = (1,2)$ $a = \text{right}$ $s' = (1,3)$ $r = 0$ $\alpha = 0.1$ and $\gamma = 1$
 current value of $V((1,2)) = 0.6$
 update $V((1,2))$ using TD-learning

$$V(1,2) = V(1,2) + \alpha(r + \gamma V(1,3) - V(1,2))$$

$$V(1,2) = V(1,2) + 0.1(0 + 1 \times V(1,3) - V(1,2))$$

$$V(1,2) = 0.62$$

Value table after step 2 is

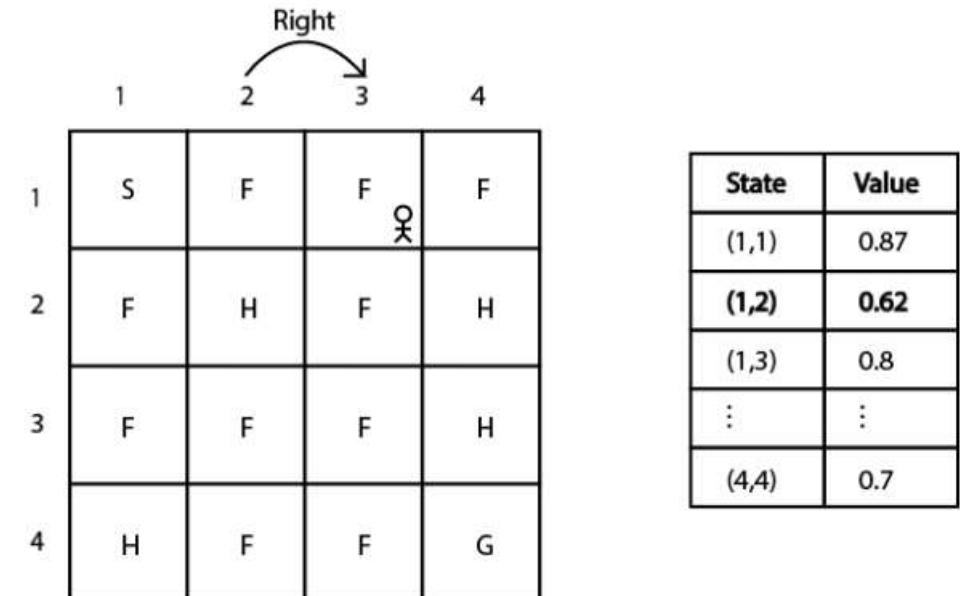


Figure 5.5: The value of state (1,2) is updated

- Step 3 : $s = (1,3)$ $a = \text{left}$ $r = 0$ $s' = (1,2)$ current $V(1,3) = 0.8$ and $V(1,2) = 0.62$

Update $V(1,3)$

$$V(1,3) = V(1,3) + \alpha(r + \gamma V(1,2) - V(1,3))$$

$$V(1,3) = V(1,3) + 0.1(0 + 1 \times V(1,2) - V(1,3))$$

$$V(1,3) = 0.8 + 0.1(0 + 1 \times 0.62 - 0.8)$$

Thus, the value of state **(1,3)** becomes:

$$V(1,3) = 0.782$$

So, we update the value of state **(1,3)** to **0.782** in the value table, as *Figure 5.6* shows:

		Left		
		1	2	3
1		S	F	F
2		F	H	F
3		F	F	F
4		H	F	F

State	Value
(1,1)	0.87
(1,2)	0.62
(1,3)	0.782
:	:
(4,4)	0.7

elu Mangai

- Thus, in this way, we compute the value of every state using the given policy.
- However, computing the value of the state just for one episode will not be accurate.
- So, we repeat these steps for several episodes and compute the accurate estimates of the state value (the value function).

Slides prepared by Dr J Alamelu Mangai

TD-Prediction Algorithm

The TD prediction algorithm is given as follows:

1. Initialize a value function $V(s)$ with random values. A policy π is given.
2. For each episode:
 1. Initialize state s
 2. For each step in the episode:
 1. Perform an action a in state s according to given policy π , get the reward r , and move to the next state s'
 2. Update the value of the state to
$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$
 3. Update $s = s'$ (this step implies we are changing the next state s' to the current state s)
 4. If s is not the terminal state, repeat steps 1 to 4

Slides prepared by Dr J Alamelu Mangai

Using TD-Prediction in FZLE

- For an input random policy π , predict the value of the states $V(s)$ using TD prediction

```
import gymnasium as gym
```

```
import pandas as pd
```

#create the envt

```
env = gym.make("FrozenLake-v1", render_mode = "human")
```

```
env.reset()
```

```
env.render()
```

Slides prepared by Dr J Alamelu Mangai

#define a random input policy

```
def random_policy():  
    return env.action_space.sample()
```

#initialize the value of all states to zeros

```
V = {}  
for s in range(env.observation_space.n):  
    V[s] = 0.0  
  
#initialize the parameters  
alpha = 0.85  
gamma = 0.90  
num_eps = 50  
num_steps = 10
```

#generating episodes

```
for i in range(num_eps):
```

```
    s = env.reset()
```

```
    s = s[0]
```

```
    for t in range(num_steps):
```

```
        a = random_policy()
```

```
        s_, r, done, _ = env.step(a)
```

```
        #use TD-update rule
```

```
        V[s] += alpha * (r + gamma * V[s_] - V[s])
```

```
        s = s_
```

```
    if done:
```

```
        break
```

#convert the dictionary to a data frame

```
df = pd.DataFrame(list(V.items()), columns = ['state', 'value'])  
print(df)
```

Output :

- Value of state 14 is maximum
- Value of all terminal states(hole state and goal state) are zeroes

Note that since we have initialized a random policy,
We might get varying results every time we
run the previous code

S	0	1	2	3
F	4	5	6	7
H				
F	8	9	10	11
F				
H	12	13	14	15
F				

Figure 5.7: States encoded as numbers

	state	value
0	0	0.1241807
1	1	0.0024911
2	2	0.0001897
3	3	0.0000000
4	4	0.0242708
5	5	0.0000000
6	6	0.0008208
7	7	0.0000000
8	8	0.1605379
9	9	0.0230677
10	10	0.0035581
11	11	0.0000000
12	12	0.0000000
13	13	0.4063436
14	14	0.8770302
15	15	0.0000000

Figure 5.8: Value table

TD-Control

- Start with a random policy and find an optimal policy iteratively
- Types – **on-policy and off-policy TD-control**
- **On-policy control :**
 - the agent behaves using one policy and tries to improve the same policy.
 - That is, in the on-policy method, we generate episodes using one policy and improve the same policy iteratively to find the optimal policy
- **Off-policy control:**
 - the agent behaves using one policy and tries to improve a different policy.
 - That is, in the off-policy method, we generate episodes using one policy and we try to improve a different policy iteratively to find the optimal policy.

Slides prepared by Dr J Alamelu Mangai

On-policy TD-Control - SARSA

- **SARSA** – state-action-reward-state-action
- in TD control our goal is to find the optimal policy.
- how can we extract a policy?
 - We can extract the policy from the Q function.
 - once we have the Q function then we can extract policy by selecting the action in each state that has the maximum Q value.
- how can we compute the Q function in TD learning?
 - Recall, in TD learning, the value function is computed as:

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

- We can just rewrite this update rule in terms of the Q function as:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

Slides prepared by Dr J Alamelu Mangai

- we compute the Q function using the preceding TD learning update rule, and then we extract a policy from them.
- the preceding update rule is also known as **the SARSA update rule**
- In the prediction method, we were given a policy as input, so we acted in the environment using that policy and computed the value function.
- But here, we don't have a policy as input. So how can we act in the environment?

- So, first we initialize the Q function with random values or with zeros.
- Then we extract a policy from this randomly initialized Q function and act in the environment.
- Our initial policy will definitely not be optimal as it is extracted from the randomly initialized Q function, but on every episode, we will update the Q function (Q values).
- So, on every episode, we can use the updated Q function to extract a new policy.
- Thus, we will obtain the optimal policy after a series of episodes.

- One important point we need to note is that in the SARSA method, instead of making our policy act greedily, we use the epsilon-greedy policy.
- In a greedy policy, we always select the action that has the maximum Q value. But, with the epsilon-greedy policy we select a random action with probability epsilon, and we select the best action (the action with the maximum Q value) with probability of 1-epsilon.

SARSA in the FZLE

- Initialize the Q function with random values

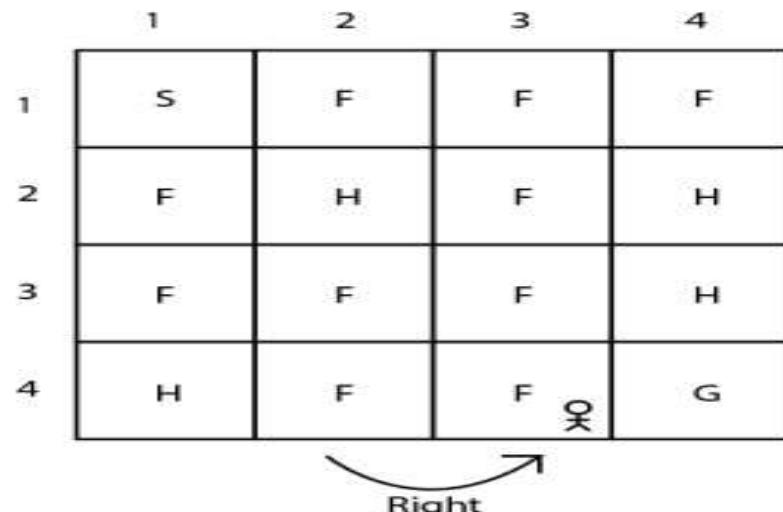
	1	2	3	4
1	S	F	F	F
2	F	H	F	H
3	F	F	F	H
4	H	F	F	G

State	Action	Value
(1,1)	Up	0.5
⋮	⋮	⋮
(4,2)	Up	0.3
(4,2)	Down	0.5
(4,2)	Left	0.1
(4,2)	Right	0.8
⋮	⋮	⋮
(4,4)	Right	0.5

Figure 5.9: The Frozen Lake environment and Q table with random values

Slides prepared by Dr J Alamelu Mangai

- Assume we are in state (4,2)
 - **Select an action using epsilon greedy policy.**
 - With probability epsilon, we select a random action and with probability 1-epsilon we select the best action (the action that has the maximum Q value).
 - **Suppose we use a probability 1-epsilon and select the best action.**
 - So, in state (4,2), we move right as it has the highest Q value compared to the other actions.
 - so, we perform the right action in state (4,2) and move to the next state (4,3) as shown in the figure



State	Action	Value
(1,1)	Up	0.5
:	:	:
(4,2)	Up	0.3
(4,2)	Down	0.5
(4,2)	Left	0.1
(4,2)	Right	0.8
:	:	:
(4,4)	Right	0.5

J Mangai

Figure 5.11: We perform the action with the maximum Q value in state (4,2)

- we moved right in state $(4,2)$ to the next state $(4,3)$ and received a reward r of 0, next state $s' = (4,3)$
- Assume learning rate α at 0.1, and the discount factor γ at 1
- Update the $Q((4,2), \text{right})$, using **SARSA update rule**

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

$$Q((4, 2), \text{right}) = Q((4, 2), \text{right}) + \alpha(r + \gamma Q((4, 3), a') - Q(4, 2), \text{right})$$

- Substituting α and γ

$$Q((4, 2), \text{right}) = Q((4, 2), \text{right}) + 0.1(0 + 1 \times Q((4, 3), a') - Q(4, 2), \text{right})$$

- Substituting $Q((4,2), \text{right}) = 0.8$, from the previous Q -table :

$$Q((4, 2), \text{right}) = 0.8 + 0.1(0 + 1 \times Q((4, 3), a') - 0.8)$$

Slides prepared by Dr J Alamelu Mangai

- what about the term $Q((4, 3), a')$? – Q-value of the next state-action pair?
- We need to choose the action a' in the next state using epsilon-greedy policy
- we select a random action with a probability of epsilon, or we select the best action that has the maximum Q value with a probability of 1-epsilon.
- Suppose we use probability epsilon and select the random action. In state $(4,3)$, we select the right action randomly, as Figure 5.12 shows.
- As you can see, although the right action does not have the maximum Q value, we selected it randomly with probability epsilon

	1	2	3	4
1	S	F	F	F
2	F	H	F	H
3	F	F	F	H
4	H	F	F	G

State	Action	Value
(1,1)	Up	0.5
⋮	⋮	⋮
(4,2)	Left	0.1
(4,2)	Right	0.8
(4,3)	Up	0.1
(4,3)	Down	0.3
(4,3)	Left	1.0
(4,3)	Right	0.9
⋮	⋮	⋮
(4,4)	Right	0.5

- Use the update rule now :

$$Q((4, 2), \text{right}) = 0.8 + 0.1(0 + 1 \times Q((4, 3), \text{right}) - 0.8)$$

- substituting the value of $Q((4, 3), \text{right})$ with 0.9

$$Q((4, 2), \text{right}) = 0.8 + 0.1(0 + 1(0.9) - 0.8)$$

- Hence the updated Q-value is

$$Q((4, 2), \text{right}) = 0.81$$

- in this way, we update the Q function by updating the Q value of the state-action pair in each step of the episode.
- After completing an episode, we extract a new policy from the updated Q function and uses this new policy to act in the environment. (Remember that our policy is always an epsilon-greedy policy).
- We repeat this steps for several episodes to find the optimal policy.

Slides prepared by Dr J Alamelu Mangai

SARSA algorithm

The SARSA algorithm is given as follows:

1. Initialize a Q function $Q(s, a)$ with random values
2. For each episode:
 1. Initialize state s
 2. Extract a policy from $Q(s, a)$ and select an action a to perform in state s
 3. For each step in the episode:
 1. Perform the action a and move to the next state s' and observe the reward r
 2. In state s' , select the action a' using the epsilon-greedy policy
 3. Update the Q value to
$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$
 4. Update $s = s'$ and $a = a'$ (update the next state s' -action a' pair to the current state s -action a pair)
 5. If s is not a terminal state, repeat steps 1 to 5

Slides prepared by Dr J Alamelu Mangai

Implement SARSA in FZLE

- Find an optimal policy for the FZLE using SARSA

#import libraries

```
import gymnasium as gym
```

```
import random
```

#create the envt

```
env = gym.make("FrozenLake-v1", render_mode = "human")
```

```
env.reset()
```

```
env.render()
```

Slides prepared by Dr J Alamelu Mangai

```
#define a dictionary for the Q table. Initialize the Q value of all (s,a)  
#pairs to 0.0
```

```
Q = {}
```

```
for s in range(env.observation_space.n):
```

```
    for a in range(env.action_space.n):
```

```
        Q[(s,a)] = 0.0
```

Slides prepared by Dr J Alamelu Mangai

#define the epsilon-greedy policy. We generate a random number
#from the uniform distribution of 0 to 1.

#If the random number is less than epsilon, we select a random
#action, else we select the best action.

```
def epsilon_greedy(state, epsilon):  
    if random.uniform(0,1) < epsilon:  
        return env.action_space.sample()  
    else:  
        return max(list(range(env.action_space.n)), key = lambda x :  
                    Q[(state,x)])
```

Slides prepared by Dr J Alamelu Mangai

```
# initialize alpha, gamma and epsilon
```

```
alpha = 0.85
```

```
gamma = 0.90
```

```
epsilon = 0.8
```

```
#set the no. of episodes and the no. of steps in each episode
```

```
num_eps = 500
```

```
num_steps = 100
```

Slides prepared by Dr J Alamelu Mangai

```
#compute the policy for each episode
for i in range(num_eps):
    s = env.reset()
    s = s[0]
    a = epsilon_greedy(s,epsilon)
    for t in range(num_steps):
        s_, r, done, _ = env.step(a)
        a_ = epsilon_greedy(s_, epsilon)
        predict = Q[(s,a)]
        target = r + gamma * Q[(s_, a_)]
        Q[(s,a)] = Q[(s,a)] + alpha * (target - predict)
        s = s_
        a = a_
        if done:
            break
```

Slides prepared by Dr J Alamelu Mangai

- Convert the Q-table into a frame and print
- Note that on every iteration we update the Q function.
- After all the iterations, we will have the optimal Q function.
- Once we have the optimal Q function then we can extract the optimal policy by selecting the action that has the maximum Q value in each state.

Slides prepared by Dr J Alamelu Mangai

Off-policy TD control – Q-learning

- Q learning is an off-policy algorithm,
- it uses two different policies, one policy for behaving in the environment (selecting an action in the environment) and the other for finding the optimal policy.
- In SARSA, we use epsilon-greedy policy to choose an action in the current state s , then used the SARSA update rule for $Q(s,a)$
- Also in the next state-action pair $Q(s',a')$, we used the same epsilon greedy policy to select an action in s' and then updated $Q(s',a')$

Slides prepared by Dr J Alamelu Mangai

- unlike SARSA, in Q learning, we use two different policies.
- One is the epsilon-greedy policy and the other is a greedy policy.
- To select an action in the environment we use an epsilon-greedy policy, but while updating the Q value of the next state-action pair $Q(s', a')$ we use a greedy policy

Slides prepared by Dr J Alamelu Mangai

- we select action a in state s using the epsilon-greedy policy and move to the next state s' and update the Q value using the update rule shown below:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

- To find $Q(s', a')$ we select an action using the greedy-policy, the policy that has the maximum Q -value in s'
- So the update rule of Q learning is given as:

$$Q(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

Slides prepared by Dr J Alamelu Mangai

Q-learning in FZLE

- Initialise the Q-table to random values

	1	2	3	4
1	S	F	F	F
2	F	H	F	H
3	F	F	F	H
4	H	F	F	G

State	Action	Value
(1,1)	Up	0.5
:	:	:
(3,2)	Up	0.1
(3,2)	Down	0.8
(3,2)	Left	0.5
(3,2)	Right	0.6
:	:	:
(4,4)	Right	0.5

Figure 5.13: The Frozen Lake environment with a randomly initialized Q table

- Assume we are in state (3,2)
- Select an action using epsilon – greedy policy. Assume with prob of 1-epsilon we choose the best action 'down'

	1	2	3	4
1	S	F	F	F
2	F	H	F	H
3	F	F	X	H
4	H	F	F	G

State	Action	Value
(1,1)	Up	0.5
:	:	:
(3,2)	Up	0.1
(3,2)	Down	0.8
(3,2)	Left	0.5
(3,2)	Right	0.6
:	:	:
(4,4)	Right	0.5

Figure 5.14: We perform the action with the maximum Q value in state (3,2)

- we perform the down action in state (3,2) and move to the next state (4,2). with $r = 0$. Assume learning rate α as 0.1, and the discount factor γ as 1

	1	2	3	4
1	S	F	F	F
2	F	H	F	H
3	F	F	F	H
4	H	F	X	G

State	Action	Value
(1,1)	Up	0.5
:	:	:
(3,2)	Up	0.1
(3,2)	Down	0.8
(3,2)	Left	0.5
(3,2)	Right	0.6
:	:	:
(4,4)	Right	0.5

- Update $Q((3,2), \text{down})$ using Q-learning update rule as

$$Q(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

Substituting the state-action pair $Q(s, a)$ with $Q((3,2), \text{down})$ and the next state s' with **(4,2)** in the preceding equation, we can write:

$$Q((3, 2), \text{down}) = Q((3, 2), \text{down}) + \alpha \left(r + \gamma \max_{a'} Q((4, 2), a') - Q(3, 2), \text{down} \right)$$

Substituting the reward, $r = 0$, the learning rate $\alpha = 0.1$, and the discount factor $\gamma = 1$, we can write:

$$Q((3, 2), \text{down}) = Q((3, 2), \text{down}) + 0.1 \left(0 + 1 \times \max_{a'} Q((4, 2), a') - Q(3, 2), \text{down} \right)$$

From the previous Q table, we can observe that the Q value of $Q((3,2), \text{down})$ is 0.8.

$$Q((3, 2), \text{down}) = 0.8 + 0.1 \left(0 + 1 \times \max_{a'} Q((4, 2), a') - 0.8 \right)$$

Slides prepared by Dr J Alamelu Mangai

- Use greedy-policy to select a' in s' .
- the right action has the maximum Q value in state (4,2).
- So, we select the right action and update the Q value of the next state-action pair:

	1	2	3	4
1	S	F	F	F
2	F	H	F	H
3	F	F	F	H
4	H	F	O	G

State	Action	Value
(1,1)	Up	0.5
:	:	:
(4,2)	Up	0.3
(4,2)	Down	0.5
(4,2)	Left	0.1
(4,2)	Right	0.8
:	:	:
(4,4)	Right	0.5

Figure 5.16: We perform the action with the maximum Q value in state (4,2)

Slides prepared by Dr J Alamelu Mangai

Thus, now our update rule becomes:

$$Q((3,2), \text{down}) = 0.8 + 0.1(0 + 1 \times Q((4, 2), \text{right}) - 0.8)$$

From the previous Q table, we can observe that the Q value of $Q((4,2), \text{right})$ is **0.8**. Thus, substituting the value of $Q((4,2), \text{right})$ with **0.8**, we can rewrite the above equation as:

$$Q((3,2), \text{down}) = 0.8 + 0.1(0 + 1 \times 0.8 - 0.8)$$

Thus, our Q value becomes:

$$Q((3, 2), \text{down}) = 0.8$$

Similarly, we update the Q value for all state-action pairs.

Slides prepared by Dr J Alamelu Mangai

- in this way, we update the Q function by updating the Q value of the state-action pair in each step of the episode.
- We will extract a new policy from the updated Q function on every step of the episode and uses this new policy.
- Remember that we select an action in the environment using epsilon-greedy policy but while updating Q value of the next state-action pair we use the greedy policy.
- After several episodes, we will have the optimal Q function

Slides prepared by Dr J Alamelu Mangai

Q-learning algorithm

The Q learning algorithm is given as follows:

1. Initialize a Q function $Q(s, a)$ with random values
2. For each episode:
 1. Initialize state s
 2. For each step in the episode:
 1. Extract a policy from $Q(s, a)$ and select an action a to perform in state s
 2. Perform the action a , move to the next state s' , and observe the reward r
 3. Update the Q value as
$$Q(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s' a') - Q(s, a) \right)$$
 4. Update $s = s'$ (update the next state s' to the current state s)
 5. If s is not a terminal state, repeat steps 1 to 5

Slides prepared by Dr J Alamelu Mangai

Implement Q-learning in FZLE

```
import gymnasium as gym
```

```
import numpy as np
```

```
import random
```

```
env = gym.make("FrozenLake-v1", render_mode = "human")
```

```
Q = {}
```

```
for s in range(env.observation_space.n):
```

```
    for a in range(env.action_space.n):
```

```
        Q[(s,a)] = 0.0
```

Slides prepared by Dr J Alamelu Mangai

```
def epsilon_greedy(state, epsilon):
    if random.uniform(0,1) < epsilon:
        return env.action_space.sample()
    else:
        return max(list(range(env.action_space.n)), key = lambda x: Q[(state,x)])
```

alpha = 0.85

gamma = 0.90

epsilon = 0.8

num_eps = 500

num_steps = 100

Slides prepared by Dr J Alamelu Mangai

```
for i in range(num_eps):  
    s = env.reset()  
    s = s[0]  
    for t in range(num_steps):  
        a = epsilon_greedy(s,epsilon)
```

```
s_, r, done, _ _ = env.step(a)  
a_ = np.argmax([Q[(s,a)] for a in range(env.action_space.n)])
```

```
Q[(s,a)] += alpha * ( r + gamma * Q[(s,a_)] - Q[(s,a)])  
s = s_  
if done:  
    break
```

Slides prepared by Dr J Alamelu Mangai

Differences between SARSA and Q-learning

S.No	SARSA	Q-learning
1	SARSA is an on-policy algorithm	Q learning is an off-policy algorithm
2	we use a single epsilon-greedy policy for selecting an action in the environment and also to compute the Q value of the next state-action pair	we use an epsilon-greedy policy for selecting an action in the environment, but to compute the Q value of next stateaction pair we use a greedy policy
3	The SARSA update rule is : $Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$	The Q-learning update rule is : $Q(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$

Slides prepared by Dr J Alamelu Mangai

Classical RL algorithms

- To learn an optimal policy :
 - DP – value and policy iteration, MC and TD
- **Comparison of DP, MC and TD methods:**
- Dynamic programming (DP), that is, the value and policy iteration methods,
 - is a model-based method, meaning that we compute the optimal policy using the model dynamics of the environment.
 - We cannot apply the DP method when we don't have the model dynamics of the environment.
- MC
 - is a model-free method, meaning that we compute the optimal policy without using the model dynamics of the environment.
 - But one problem we face with the MC method is that it is applicable only to episodic tasks and not to continuous tasks.
- TD :
 - a model-free method.
 - TD learning takes advantage of both DP by bootstrapping and the MC method by being model free.

Slides prepared by Dr J Alamelu Mangai

Module 4

CSE3011 Reinforcement Learning

Credit Structure : 2-2-3



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Case Study – The MAB Problem

Topics : The MAB problem, The epsilon-greedy method, Softmax exploration, The upper confidence bound algorithm, The Thompson sampling algorithm, Applications of MAB, Finding the best advertisement banner using MAB, Contextual bandits.



Introduction

- Model-free methods do not require the model dynamics of the environment to compute the value and Q functions in order to find the optimal policy.
- One such popular model-free method is the Monte Carlo (MC) method.
- The Monte Carlo method is a statistical technique used to find an approximate solution through sampling.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



The MAB Problem

- The MAB problem is one of the classic problems in reinforcement learning. A MAB is a slot machine where we pull the arm (lever) and get a payout (reward) based on some probability distribution.
- A single slot machine is called a one-armed bandit and when there are multiple slot machines it is called a MAB or k-armed bandit, where k denotes the number of slot machines.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction



Figure 6.1: 3-armed bandit slot machines



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- Each arm has its own probability distribution indicating the probability of winning and losing the game.
- For example, let's suppose we have two arms. Let the probability of winning if we pull arm 1 (slot machine 1) be 0.7 and the probability of winning if we pull arm 2 (slot machine 2) be 0.5.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- Thus, we can say that pulling arm 1 is desirable as it makes us win the game 70% of the time. However, this probability distribution of the arm (slot machine) will not be given to us.
- We need to find out which arm helps us to win the game most of the time and gives us a good reward.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- Let's denote the arm by a and define the average reward by pulling the arm a as:

$$Q(a) = \frac{\text{Sum of rewards obtained from the arm}}{\text{Number of times the arm was pulled}}$$

Where $Q(a)$ denotes the average reward of arm a .

The optimal arm a^* is the one that gives us the maximum average reward, that is:

$$a^* = \arg \max_a Q(a)$$

- The arm that gives the maximum average reward is the optimal arm.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Introduction

- Along with finding the best arm, our goal should be to minimize the cost of identifying the best arm, and this is usually referred to as regret.
- Thus, we need to find the best arm while minimizing regret. That is, we need to find the best arm, but we don't want to end up selecting the arms that make us lose the game in most of the rounds.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Exploration strategies

To overcome exploration-exploitation dilemma in the MAB problem, we use the following exploration strategies:

- Epsilon-greedy
- Softmax exploration
- Upper confidence bound
- Thomson sampling



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Epsilon-greedy

- With epsilon-greedy, we select the best arm with a probability 1-epsilon and we select a random arm with a probability epsilon.
- Say we have two arms—arm 1 and arm 2. Suppose with arm 1 we win the game 80% of the time and with arm 2 we win the game 20% of the time. So, we can say that arm 1 is the best arm as it makes us win the game 80% of the time.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Epsilon-greedy method

First, we initialize the `count` (the number of times the arm is pulled), `sum_rewards` (the sum of rewards obtained from pulling the arm), and `Q` (the average reward obtained by pulling the arm), as *Table 6.1* shows:

arm	count	sum_rewards	Q
arm 1	0	0	0
arm 2	0	0	0

Table 6.1: Initialize the variables with zero



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Epsilon-greedy method

Round 1:

Say, in round 1 of the game, we select a random arm with a probability epsilon, and suppose we randomly pull arm 1 and observe the reward. Let the reward obtained by pulling arm 1 be 1. So, we update our table with `count` of arm 1 set to 1, and `sum_rewards` of arm 1 set to 1, and thus the average reward Q of arm 1 after round 1 is 1 as *Table 6.2* shows:

arm	count	sum_rewards	Q
arm 1	1	1	1
arm 2	0	0	0

Table 6.2: Results after round 1



Epsilon-greedy method

Round 2:

Say, in round 2, we select the best arm with a probability $1-\epsilon$. The best arm is the one that has the maximum average reward. So, we check our table to see which arm has the maximum average reward. Since arm 1 has the maximum average reward, we pull arm 1 and observe the reward and let the reward obtained from pulling arm 1 be 1.

So, we update our table with count of arm 1 to 2 and sum_rewards of arm 1 to 2, and thus the average reward Q of arm 1 after round 2 is 1 as *Table 6.3* shows:

arm	count	sum_rewards	Q
arm 1	2	2	1
arm 2	0	0	0

Table 6.3: Results after round 2



Epsilon-greedy method

Round 3:

Say, in round 3, we select a random arm with a probability epsilon. Suppose we randomly pull arm 2 and observe the reward. Let the reward obtained by pulling arm 2 be 0. So, we update our table with count of arm 2 set to 1 and sum_rewards of arm 2 set to 0, and thus the average reward Q of arm 2 after round 3 is 0 as

Table 6.4 shows:

arm	count	sum_rewards	Q
arm 1	2	2	1
arm 2	1	0	0

Table 6.4: Results after round 3



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Epsilon-greedy method

Round 4:

Say, in round 4, we select the best arm with a probability $1-\epsilon$. So, we pull arm 1 since it has the maximum average reward. Let the reward obtained by pulling arm 1 be 0 this time. Now, we update our table with count of arm 1 to 3 and sum_rewards of arm 2 to 2, and thus the average reward Q of arm 1 after round 4 will be 0.66 as *Table 6.5* shows:

arm	count	sum_rewards	Q
arm 1	3	2	0.66
arm 2	1	0	0

Table 6.5: Results after round 4



Epsilon-greedy method

- We repeat this process for several rounds; that is, for several rounds of the game, we pull the best arm with a probability 1-epsilon and we pull a random arm with probability epsilon.

Table 6.6 shows the updated table after 100 rounds of the game:

arm	count	sum_rewards	Q
arm 1	75	61	0.81
arm 2	25	2	0.08

Table 6.6: Results after 100 rounds

From *Table 6.6*, we can conclude that arm 1 is the best arm since it has the maximum average reward.

Softmax exploration

- Softmax exploration, also known as Boltzmann exploration, is another useful exploration strategy for finding the optimal arm.
- In the epsilon-greedy policy, all the non-best arms are explored equally. That is, all the non-best arms have a uniform probability of being selected.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration

For instance, as *Table 6.7* shows, arm 1 is the best arm as it has a high average reward Q . So, we assign a high probability to arm 1. Arms 2, 3, and 4 are the non-best arms, and we need to explore them. As we can observe, arm 3 has an average reward of 0. So, instead of selecting all the non-best arms uniformly, we give more priority to arms 2 and 4 than arm 3. So, the probability of arm 2 and 4 will be high compared to arm 3:

arm	Q
arm 1	0.84
arm 2	0.24
arm 3	0.0
arm 4	0.13

Table 6.7: Average reward for a 4-armed bandit



Softmax exploration

Thus, in softmax exploration, we select the arms based on a probability. The probability of each arm is directly proportional to its average reward:

$$p_t(a) \propto Q_t(a)$$

But wait, the probabilities should sum to 1, right? The average reward (Q value) will not sum to 1. So, we convert them into probabilities with the softmax function, as shown here:

$$P_t(a) \propto \frac{\exp(Q_t(a))}{\sum_{i=1}^n \exp(Q_t(i))} \quad (1)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration

- So, now the arm will be selected based on the probability. However, in the initial rounds we will not know the correct average reward of each arm, so selecting the arm based on the probability of average reward will be inaccurate in the initial rounds.
- To avoid this, we introduce a new parameter called T . T is called the temperature parameter.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration

We can rewrite the preceding equation with the temperature T , as shown here:

$$P_t(a) \propto \frac{\exp(Q_t(a)/T)}{\sum_{i=1}^n \exp(Q_t(i)/T)} \quad (2)$$

- When T is high, all the arms have an equal probability of being selected and when T is low, the arm that has the maximum average reward will have a high probability.
- So, we set T to a high number in the initial rounds, and after a series of rounds we reduce the value of T . This means that in the initial round we explore all the arms equally and after a series of rounds, we select the best arm that has a high probability.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

Let's understand this with a simple example. Say we have four arms, arm 1 to arm 4. Suppose we pull arm 1 and receive a reward of 1. Then the average reward of arm 1 will be 1 and the average reward of all other arms will be 0, as *Table 6.8* shows:

arm	Q
arm 1	1
arm 2	0
arm 3	0
arm 4	0

Table 6.8: Average reward for each arm



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

Now, if we convert the average reward to probabilities using the softmax function given in equation (1), then our probabilities look like the following:

arm	Q	probability
arm 1	1	0.475
arm 2	0	0.174
arm 3	0	0.174
arm 4	0	0.174

Table 6.9: Probability of each arm

As we can observe, we have a 47% probability for arm 1 and a 17% probability for all other arms. But we cannot assign a high probability to arm 1 by just pulling arm 1 once.



Softmax exploration example

So, we set T to a high number, say $T = 30$, and calculate the probabilities based on equation (2). Now our probabilities become:

arm	Q	probability
arm 1	1	0.253
arm 2	0	0.248
arm 3	0	0.248
arm 4	0	0.248

Table 6.10: Probability of each arm with $T=30$

As we can see, now all the arms have equal probabilities of being selected. Now we explore the arms based on this probability and over a series of rounds, the T value will be reduced, and we will have a high probability to the best arm.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

Let's suppose after some 30 rounds, the average reward of all the arms is

arm	Q
arm 1	0.84
arm 2	0.24
arm 3	0.0
arm 4	0.13

Table 6.11: Average reward for each arm after 30+ rounds



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

We learned that the value of T is reduced over several rounds. Suppose the value of T is reduced and it is now 0.3 ($T=0.3$); then the probabilities will become:

arm	Q	probability
arm 1	0.84	0.775
arm 2	0.24	0.104
arm 3	0.0	0.047
arm 4	0.13	0.072

Table 6.12: Probabilities for each arm with T now set to 0.3

As we can see, arm 1 has a high probability compared to other arms. So, we select arm 1 as the best arm and explore the non-best arms - [arm 2, arm 3, arm 4] - based on their probabilities in the next rounds.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Softmax exploration example

- Thus, in the initial round, we don't know which arm is the best arm. So instead of assigning a high probability to the arm based on the average reward, we assign an equal probability to all the arms in the initial round with a high value of T and over a series of rounds, we reduce the value of T and assign a high probability to the arm that has a high average reward.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Upper confidence bound

- The confidence interval denotes the interval within which the true value lies. So, in our setting, the confidence interval denotes the interval within which the true mean reward of the arm lies.

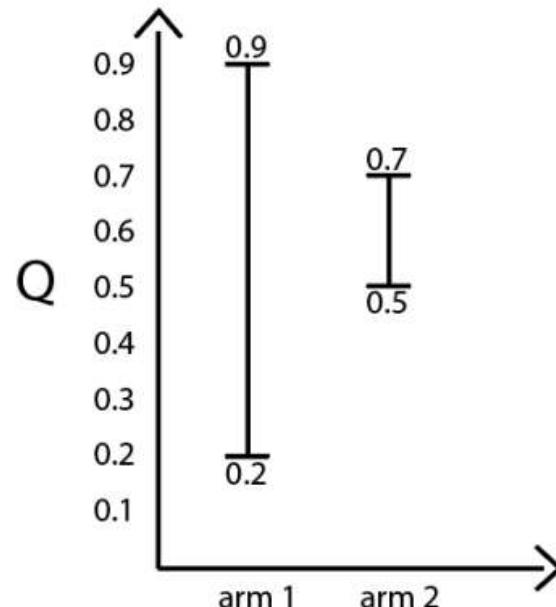


Figure 6.2: Confidence intervals for arms 1 and 2

Upper confidence bound

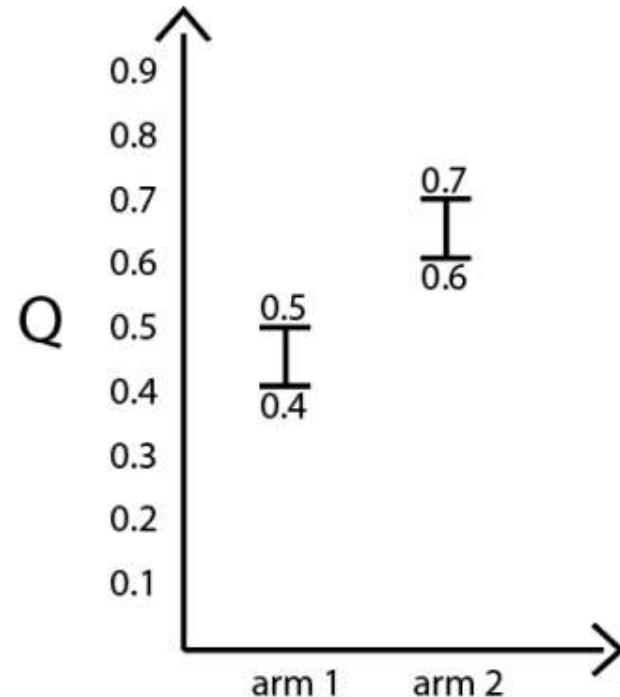


Figure 6.3: Confidence intervals for arms 1 and 2 after several rounds



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Upper confidence bound

- As we play the game for several rounds by selecting the arm that has a high UCB, our confidence interval of both arms will get narrower and denote a more accurate mean value.
- For instance, as we can see in Figure 6.3, after playing the game for several rounds, the confidence interval of both the arms becomes small and denotes a more accurate mean value.
- The confidence interval of both arms is small and we have a more accurate mean, and since in UCB we select arm that has the highest UCB, we select arm 2 as the best arm.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Upper confidence bound

Let $N(a)$ be the number of times arm a was pulled and t be the total number of rounds, then the upper confidence bound of arm a can be computed as:

$$\text{UCB}(a) = Q(a) + \sqrt{\frac{2\log(t)}{N(a)}} \quad (3)$$

We select the arm that has the highest upper confidence bound as the best arm:

$$a^* = \arg \max_a \text{UCB}(a)$$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Upper confidence bound

The algorithm of UCB is given as follows:

1. Select the arm whose upper confidence bound is high
2. Pull the arm and receive a reward
3. Update the arm's mean reward and confidence interval
4. Repeat steps 1 to 3 for several rounds



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

- Thompson sampling (TS) is another interesting exploration strategy to overcome the exploration-exploitation dilemma and it is based on a beta distribution.
- So, before diving into Thompson sampling, let's first understand the beta distribution. The beta distribution is a probability distribution function and it is expressed as:

$$f(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1}$$

Where $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)}$ and $\Gamma(\cdot)$ is the gamma function.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Beta distribution

- The shape of the distribution is controlled by the two parameters α and β . When the values of α and β are the same, then we will have a symmetric distribution.

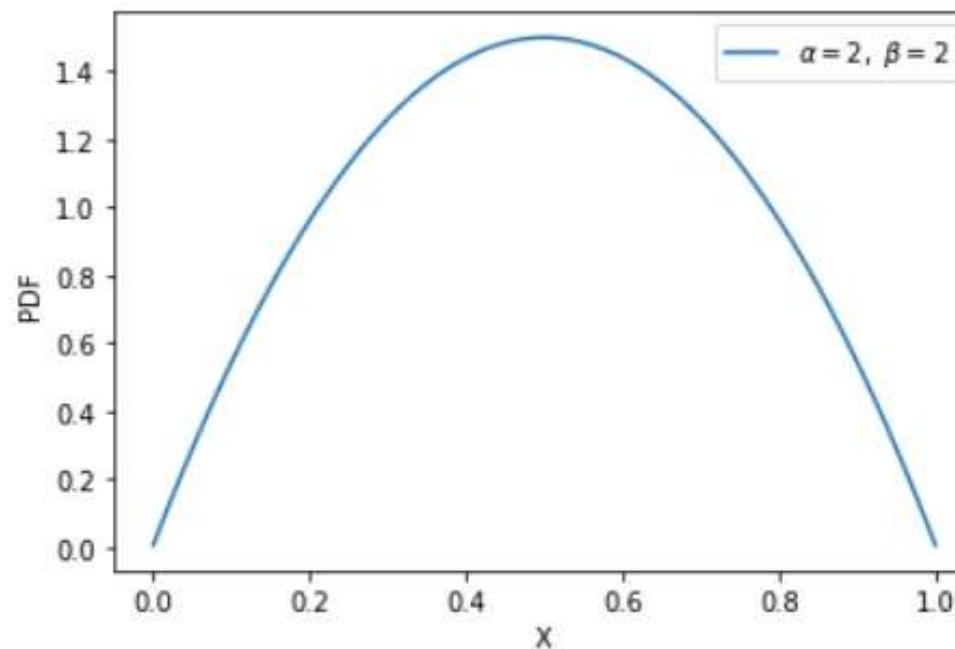


Figure 6.4: Symmetric beta distribution

Beta distribution

- When the value of α is higher than β then we will have a probability closer to 1 than 0. For instance, as Figure 6.5 shows, since the value of $\alpha=9$ and $\beta=2$, we have a high probability closer to 1 than 0:

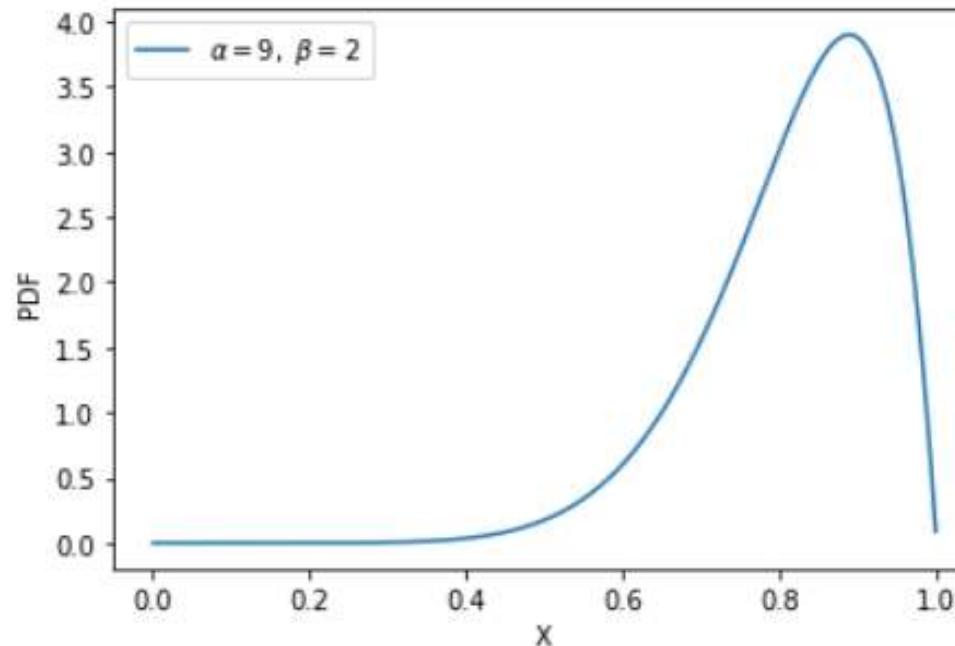


Figure 6.5: Beta distribution where $\alpha > \beta$



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Beta distribution

- When the value of β is higher than α then we will have a high probability closer to 0 than 1. For instance, as shown in the following plot, since the value of $\alpha=2$ and $\beta=9$, we have a high probability closer to 0 than 1:

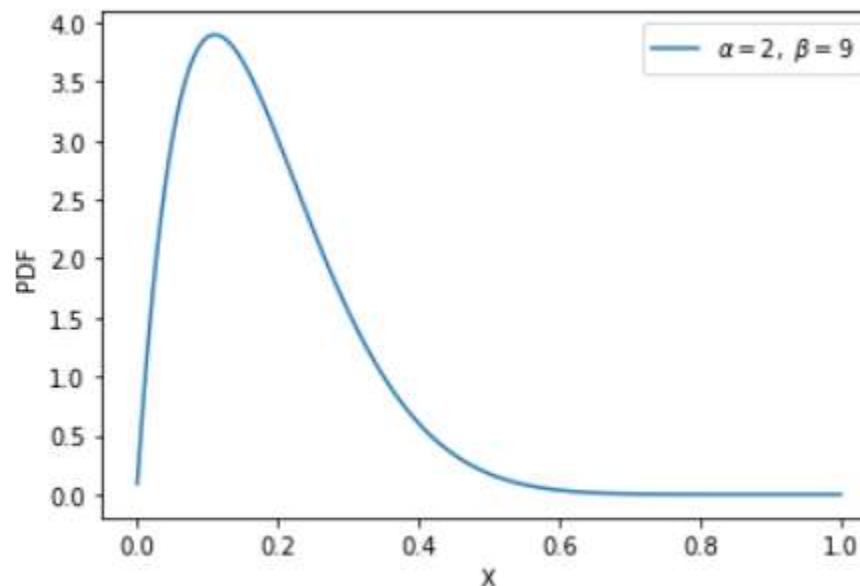


Figure 6.6: Gamma distribution where $\alpha < \beta$

Beta distribution

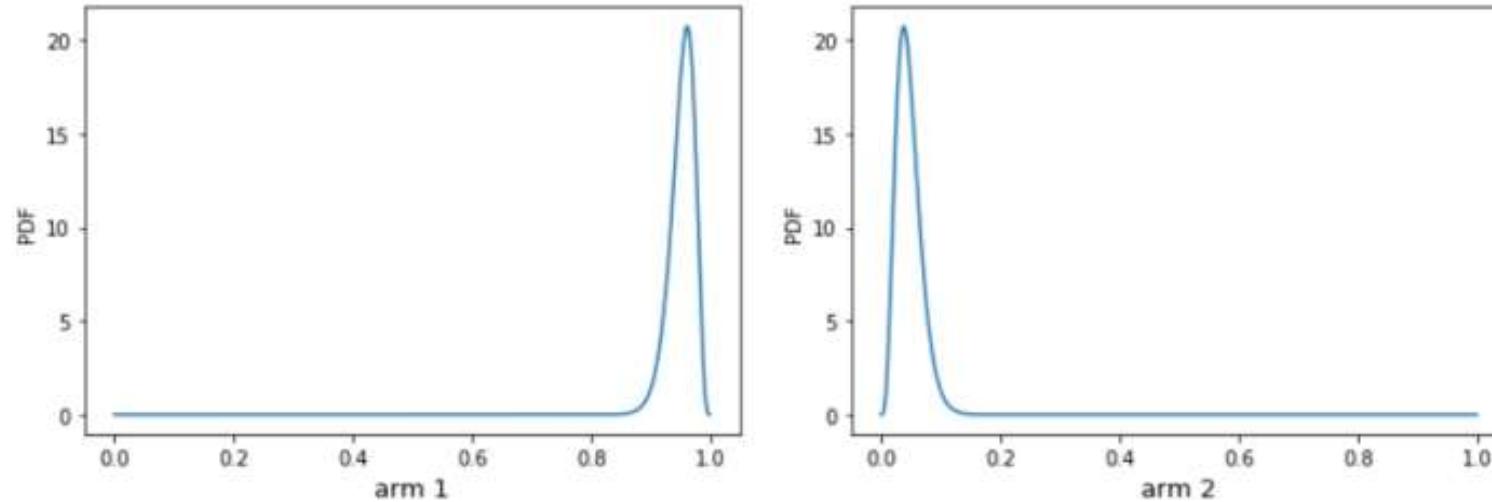


Figure 6.7: True distributions for arms 1 and 2



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Beta distribution

- From Figure 6.7, we can see that it is better to pull arm 1 than arm 2 because arm 1 has a high probability close to 1, but arm 2 has a high probability close to 0.
- So, if we pull arm 1, we get a reward of 1 and win the game, but if we pull arm 2 we get a reward of 0 and lose the game. Thus, once we know the true distribution of the arms then we can understand which arm is the best arm.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

- Thompson sampling is a probabilistic method and it is based on a prior distribution.
- Here, we use the beta distribution as a prior distribution. Say we have two arms, so we will have two beta distributions (prior distributions), and we initialize both α and β to the same value, say 3, as Figure 6.10 shows:

Thompson sampling

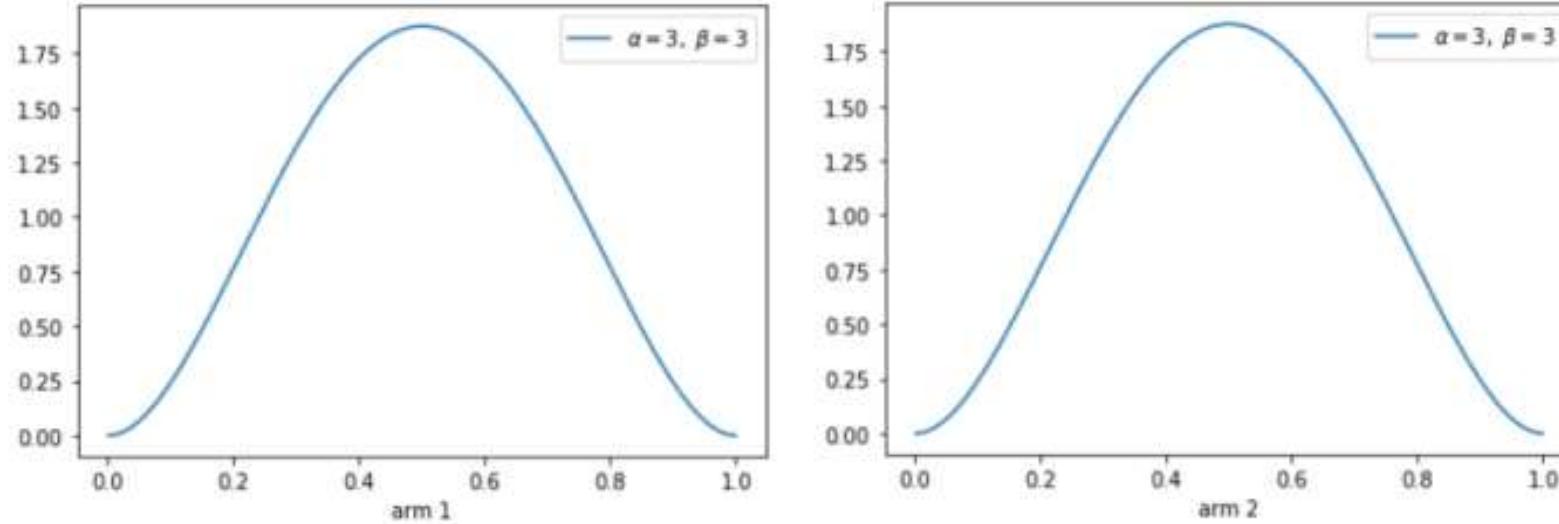


Figure 6.10: Initialized prior distributions for arms 1 and 2 look the same



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

- In the first round, we just randomly sample a value from these two distributions and select the arm that has the maximum sampled value. Let's say the sampled value of arm 1 is high, so in this case, we pull arm 1.
- Say we win the game by pulling arm 1, then we update the distribution of arm 1 by incrementing the alpha value of the distribution by 1; that is, we update the alpha value as $\alpha = \alpha + 1$.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Thompson sampling

As Figure 6.11 shows, the alpha value of the distribution of arm 1 is incremented, and as we can see, arm 1's beta distribution has slightly high probability closer to 1 compared to arm 2:

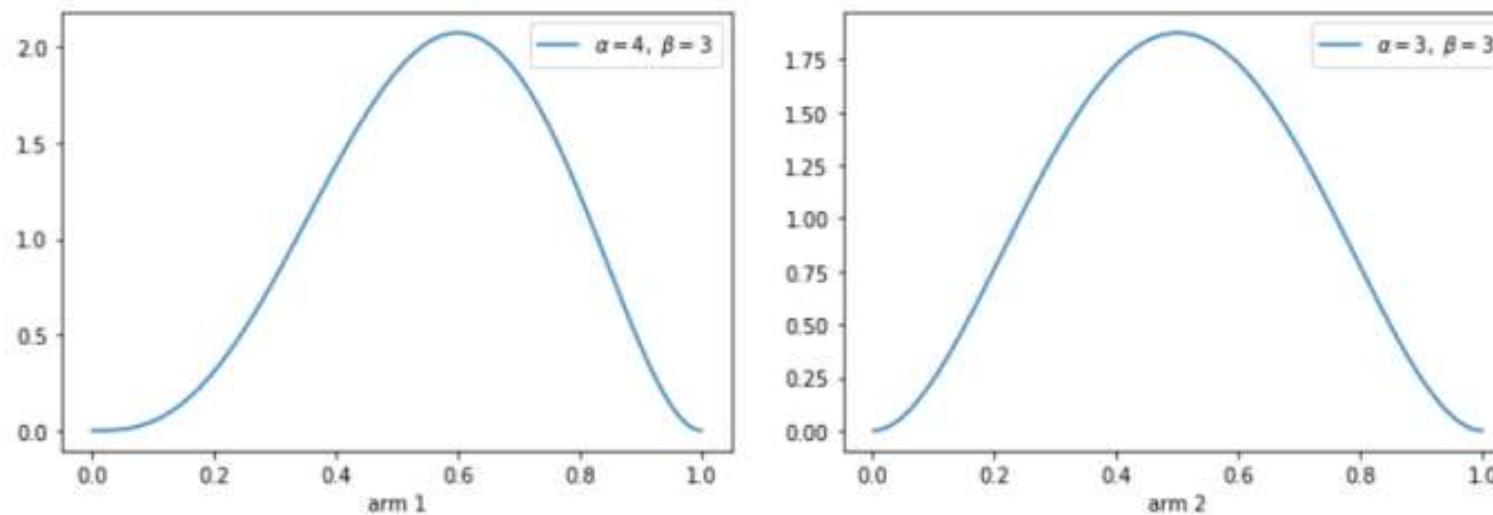


Figure 6.11: Prior distributions for arms 1 and 2 after round 1

Thompson sampling

- As Figure 6.12 shows, the alpha value of arm 1's distribution is incremented, and arm 1's beta distribution has a slightly high probability close to 1

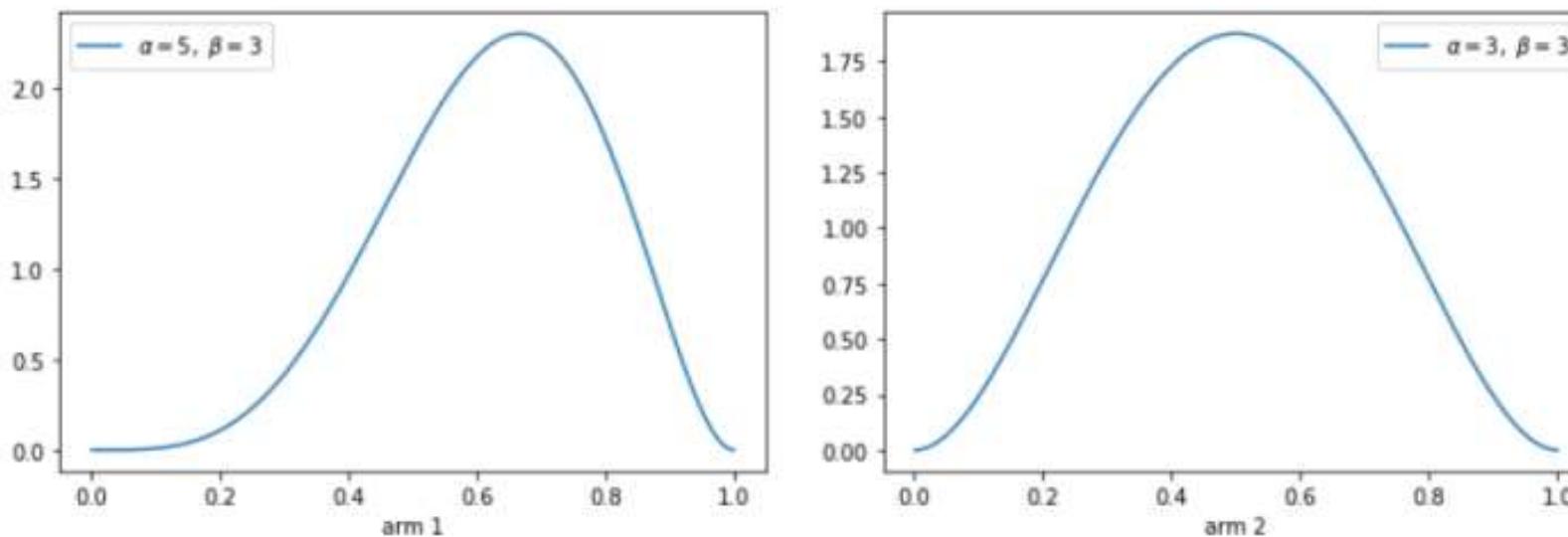


Figure 6.12: Prior distributions for arms 1 and 2 after round 2

Thompson sampling

- As Figure 6.13 shows, the beta value of arm 2's distribution is incremented and the beta distribution of arm 2 has a slightly higher probability close to 0

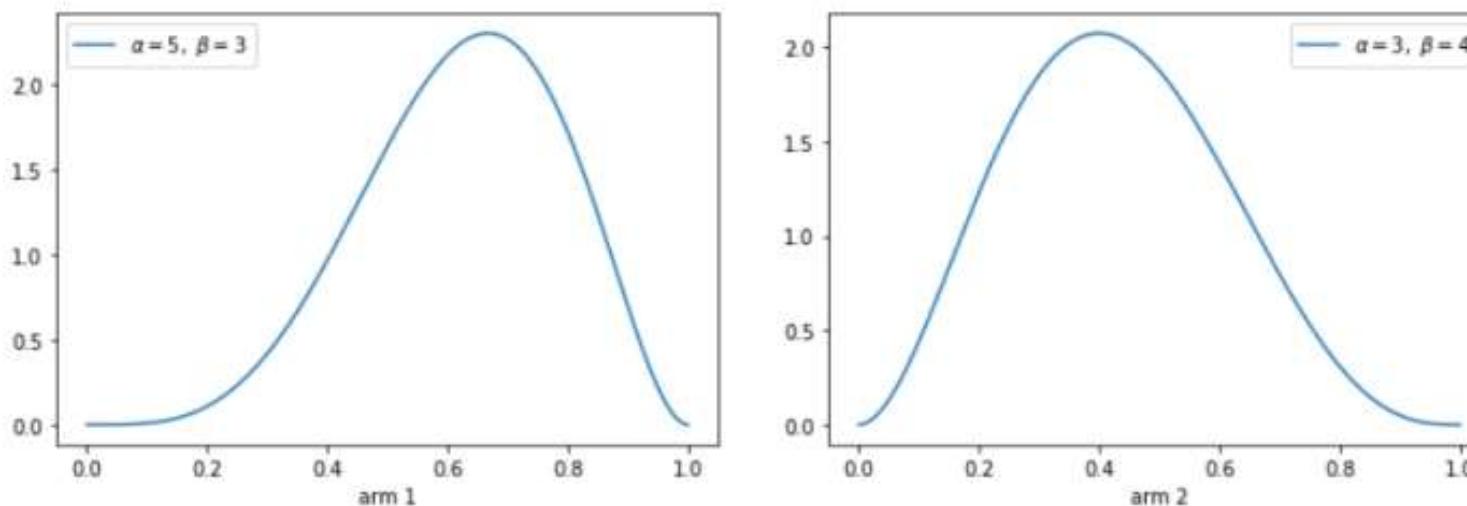


Figure 6.13: Prior distributions for arms 1 and 2 after round 3

Thompson sampling

- As Figure 6.14 shows, the beta value of arm 2's distribution is incremented by 1 and also arm 2's beta distribution has a slightly high probability close to 0:

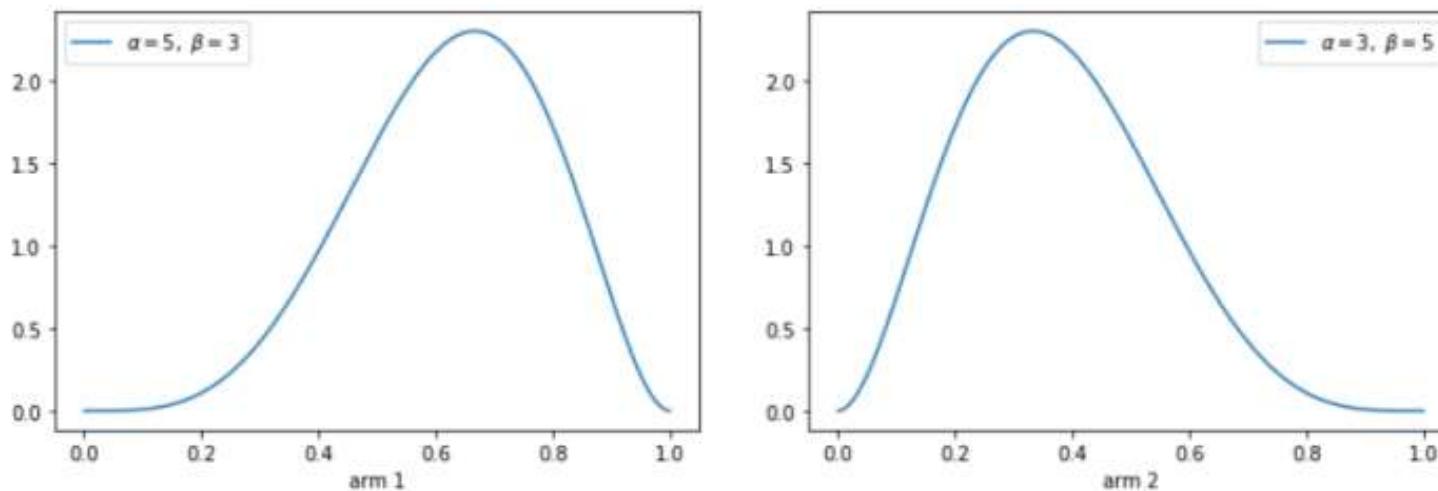


Figure 6.14: Prior distributions for arms 1 and 2 after round 4

Thompson sampling

If we do this repeatedly for several rounds, then we can learn the true distribution of the arm. Say after several rounds, our distribution will look like Figure 6.15. As we can see, the distributions of both arms resemble the true distributions:

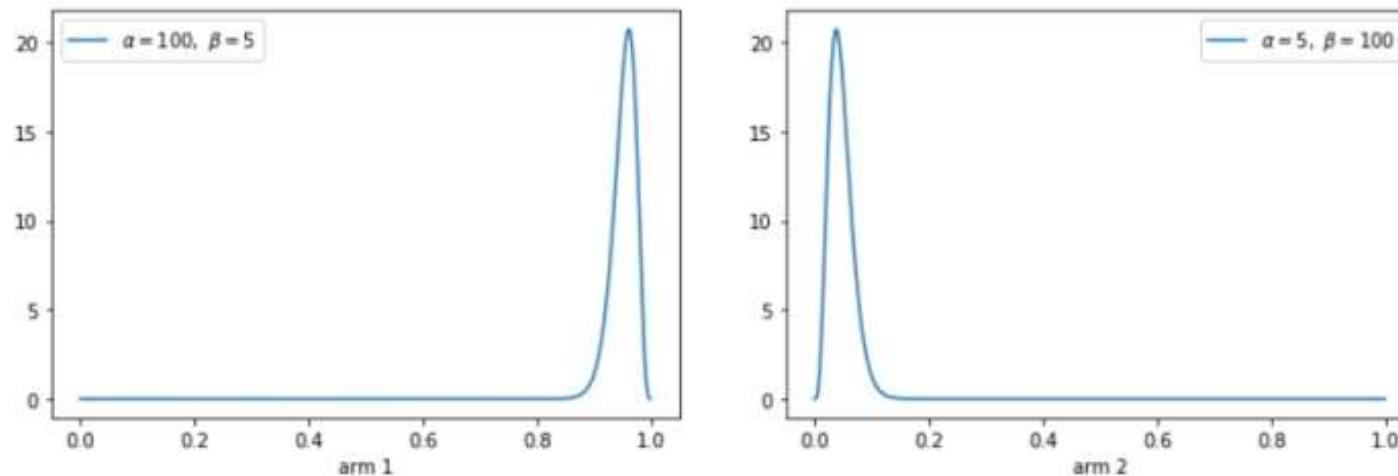


Figure 6.15: Prior distributions for arms 1 and 2 after several rounds

Thompson sampling

Now if we sample a value from each of these distributions, then the sampled value will always be high from arm 1 and we always pull arm 1 and win the game.

The steps involved in the Thomson sampling method are given here:

1. Initialize the beta distribution with alpha and beta set to equal values for all k arms
2. Sample a value from the beta distribution of all k arms
3. Pull the arm whose sampled value is high
4. If we win the game, then update the alpha value of the distribution to $\alpha = \alpha + 1$
5. If we lose the game, then update the beta value of the distribution to $\beta = \beta + 1$
6. Repeat steps 2 to 5 for many rounds



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Applications of MAB

Bandits are widely used for:

- Website optimization
- Maximizing conversion rates
- Online advertisements
- Campaigning



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013



Contextual bandits

- In the MAB problem, we just perform the action and receive a reward. But with contextual bandits, we take actions based on the state of the environment and the state holds the context.
- For instance, in the advertisement banner example, the state specifies the user behavior and we will take action (show the banner) according to the state (user behavior) that will result in the maximum reward (ad clicks).
- Contextual bandits are widely used for personalizing content according to the user's behavior. They are also used to solve the cold-start problems faced by recommendation systems. Netflix uses contextual bandits for personalizing the artwork for TV shows according to user behavior.



**PRESIDENCY
UNIVERSITY**

Private University Estd. in Karnataka State by Act No. 41 of 2013

