**Deterministic**

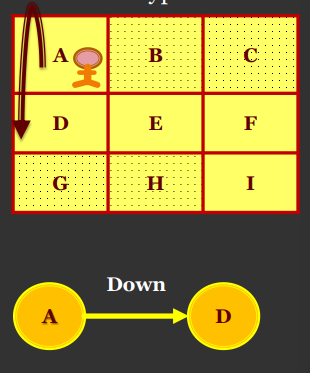
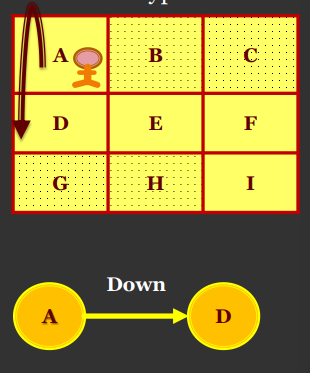
In a deterministic environment it is certain that an agent performs action a in the state s, it always reaches state s′

1. The current state and action can completely determine the next state of the environment.
2. No uncertainty.
3. Next state is observable.
4. Tic-tac-toe a fully observable, deterministic. It is a very small problem.
5. However, most of AI based problem solving are not deterministic.
6. On the contrary is a non-deterministic environment, the same task performed twice may produce different results or may even fail completely+.
7. Examples:

Non-deterministic environment: physical world: Robot on Mars

Deterministic environment: Tic Tac Toe game

For example: Consider the grid world environment, lets say the agent is in state A, when it moves down from state A, it always reaches the state D

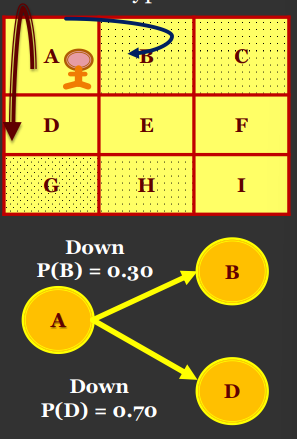
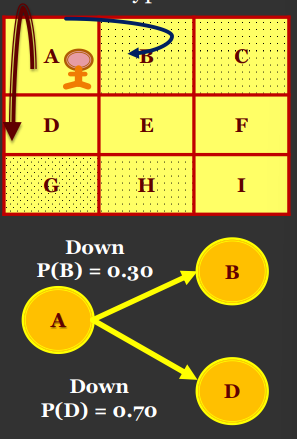


**Stochastic**

In a STOCHASTIC environment can’t say the agent performing action a in state s it always reaches state s′ as there is randomness.

1. It is random in nature and cannot be determined completely by the agent.
2. Uncertain
3. Next state is totally unpredictable.
4. Self-driving vehicles are classical example of stochastic AI process.

For example: Consider the grid world environment, lets say the agent is in state A, now if it moves down from state A, then the agent does not always reach D, it reaches state D 70% (Probability) of time and state B 30% (Probability) of time.



**Discrete**

An environment is said to be discrete if there are a finite number of actions that can be performed within it.

Environment’s action space is discrete.

1. In grid world environment, there is a discrete action space, which consists of the actions [up, down, left, right] and thus grid world environment is discrete
2. A game of chess or checkers is discrete where there are a set number of moves.

**Continuous**

In a continuous environment, environment action space is continuous.

1. The most complex environment is one that is inaccessible, non-deterministic, non-episodic, dynamic and continuous.
2. The more complex an environment is, the harder it is to decide which action to perform.
3. Taxi driving: There could be a route from to anywhere to anywhere else.
4. For example, a agent is trained to drive a car, then the action space is continuous, with several continuous actions such as changing car’s speed, the number of degrees it needs to rotate the wheel, and so on.

**Episodic**

1. In an episodic environment, each agent’s performance is the result of a series of independent tasks performed. There is no link between the agent’s performance and other different scenarios. In other words, the agent decides which action is best to take, it will only consider the task at hand and doesn’t have to consider the effect it may have on future tasks
2. Episodic is an environment where each state is independent of each other.
3. The action on a state has nothing to do with the next state.
4. Real-life Example: A support bot (agent) answer to a question and then answer to another question and so on. So each question-answer is a single episode.
5. Episodic environment: mail sorting system.

**Non-Episodic or Sequential**

1. The sequential environment is an environment where the next state is dependent on the current action.
2. So, agent current action can change all of the future states of the environment.
3. Real-life Example: Playing tennis is a perfect example where a player observes the opponent’s shot and takes action.
4. Non-episodic environment: chess game

**Single - Agent**

1. In single agent environment all actions are performed by a single agent in the environment.
2. Playing tennis against the ball is a single agent environment where there is only one player.

**Multi-Agent**

1. In multi agent environment all actions are performed by two or more agents, in the environment.
2. Playing a soccer match is a multi-agent environment.

**Version 2:**

\*\***Deterministic Environment**:\*\*

- In a deterministic environment, the outcome is predictable and certain when the agent performs a specific action in a given state.

- The current state and action fully determine the next state.

- No randomness or uncertainty is involved.

- The next state is observable and can be precisely calculated.

- Examples include games like Tic-Tac-Toe, where each move leads to a known outcome.

- However, most real-world AI problems are not deterministic; they involve uncertainty and variability.

\*\***Stochastic Environment**:\*\*

- In a stochastic environment, outcomes are uncertain and can vary even if the same action is repeated in the same state.

- Randomness is a factor, making predictions difficult.

- The next state is unpredictable and may vary based on chance.

- Examples include self-driving vehicles navigating through traffic, where external factors introduce randomness.

\*\***Discrete Environment**:\*\*

- A discrete environment has a finite number of actions that can be performed within it.

- The set of possible actions is limited and countable.

- Examples include games like chess or checkers, where a specific set of moves can be made.

\*\***Continuous Environment**:\*\*

- In a continuous environment, the action space is continuous, involving a range of possible actions.

- Actions are not limited to specific choices but can have a wide spectrum of values.

- Examples include training an agent to drive a car, where actions like changing speed or turning the wheel involve continuous values.

- Complex environments with continuous actions pose challenges for decision-making.

* deterministic environments have certain outcomes based on actions,
* while stochastic environments introduce randomness.
* Discrete environments have a finite set of actions,
* while continuous environments involve a wide range of continuous actions.

\*\***Episodic Environment**:\*\*

- In an episodic environment, each state or task is independent of others.

- The action taken in one state does not affect the outcome or states of future tasks.

- The agent's decision-making is isolated to the current task and does not need to consider future consequences.

- An example is a customer support chatbot responding to different customer queries independently.

- Each episode is like a standalone task with no impact on the next episode.

\*\***Non-Episodic (Sequential) Environment**:\*\*

- In a non-episodic or sequential environment, the outcome of one state is dependent on the actions taken in previous states.

- The agent's current action can influence all future states and outcomes.

- Decision-making involves considering the consequences of actions on future states.

- An example is playing a game of chess, where each move affects the entire sequence of moves and the final outcome.

- In a non-episodic environment, actions have a lasting impact on the entire sequence of events.

In an episodic environment, each task is independent, and the agent's actions don't affect future tasks. It's like solving individual puzzles.

In a non-episodic or sequential environment, the current action has consequences that impact all future states, making decision-making more complex and interconnected, much like playing a game where each move influences the entire gameplay.

\*\***Single-Agent Environment**:\*\*

- In a single-agent environment, there is only one entity or player making decisions and interacting with the environment.

- For example, playing a game of tennis against a ball machine represents a single-agent environment. The player is the sole agent involved in decision-making and actions within the environment.

\*\***Multi-Agent Environment**:\*\*

- In a multi-agent environment, there are multiple entities or players, each making their own decisions and interacting with the environment.

- For instance, playing a soccer match involves multiple players on each team, making decisions and competing against each other. The interactions between players and their coordinated actions create a complex multi-agent environment.

**Markov Chain**

1. The Markov property states that the future depends only on the present and not on the past.
2. The Markov chain is a probabilistic model that solely depends on the current state and not the previous states, future is conditionally independent of past.
3. Moving from current state s to the next state s’ is called transition , denoted as (P (s|′ s)) and its probability is called a transition probability.
4. Markov property does not hold for all processes.
5. Markov chain or Markov process consists of a set of states along with their transition probabilities.

**Markov Decision Process (MDP)**

• MDP is an extension of the Markov chain. It provides a mathematical framework for modelling decision-making situations.

• All reinforcement learning problems can be modelled as MDP.

• The key elements of MDP are as follows.

1. A set of states (S) the agent can be in.
2. A set of actions (A) that can be performed by an agent, for moving from one state to

another.

1. A transition probability (Pas1s2), which is the probability of moving from one state to

another state by performing some action a.

1. A reward probability (Ras1s2), which is the probability of a reward acquired by the agent for moving from one state to another state by performing some action a.
2. A discount factor (γ), which controls the importance of immediate and future rewards.

**Version 2:**

\*\***Markov Chain**:\*\*

1. The Markov property means that the future only depends on the current state, not on the past states.

2. Markov chain is a model that relies solely on the current state for making predictions, making future outcomes independent of past events.

3. Transitioning from the current state to the next state is represented as (P(s'|s)), and this transition has a probability called transition probability.

4. The Markov property doesn't apply to all processes; it specifically refers to situations where future outcomes are determined only by the present.

5. Markov chain consists of a set of states along with their associated transition probabilities.

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

- MDP is an extension of the Markov chain and provides a mathematical framework for modeling decision-making problems.

- It encompasses various reinforcement learning scenarios.

- Key components of MDP include:

1. A set of states (S) that the agent can be in.

2. A set of actions (A) the agent can perform to transition between states.

3. Transition probabilities (P(s'|s)) indicating the likelihood of moving from one state to another when taking an action.

4. Reward probabilities (R(s,a,s')) indicating the likelihood of receiving a reward when transitioning between states with a certain action.

5. A discount factor (γ) that balances the importance of immediate and future rewards in decision-making.

This approach makes sense in applications when agent interacts in an environment and breaks the interactions naturally into sub sequences called Episodes.

Tasks with episodes are called Episodic tasks

Playing a maze game: Trip through maze, or any repeated interactions.

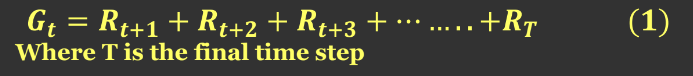
Each episode ends in a special state called the terminal state, followed by a reset to a standard starting state or to a sample from a standard distribution of starting state

Goal is to maximize the expected return, where the return denoted as Gt is defined as some specific function of a reward sequence.

How to define formally?

Rt+1,Rt+2 ,Rt+3 ,Rt+4, . , . , . ,

Gt = Rt+1 + R t+2  + R t+3  + ⋯ +R t  -- (1)

 //////

A return can be defined as the sum of rewards obtained by the agent in an episode.

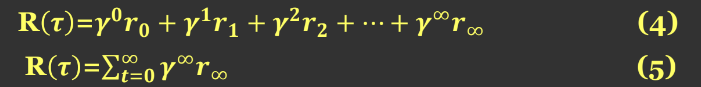
The return is often denoted by R or G

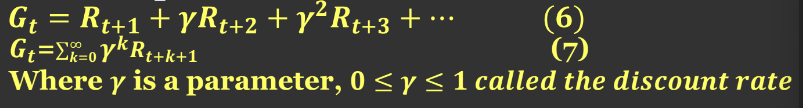
Consider an agent in an episode τ start from initial state at time step t=0, reaches the final state at time step T, then the reward obtained by the agent is

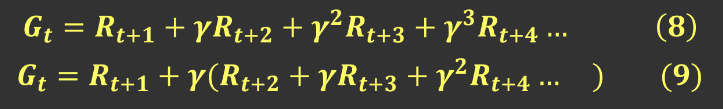


In case of continuous tasks, there is no terminal state, so define the return as sum of

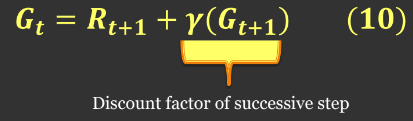
rewards up to infinity.

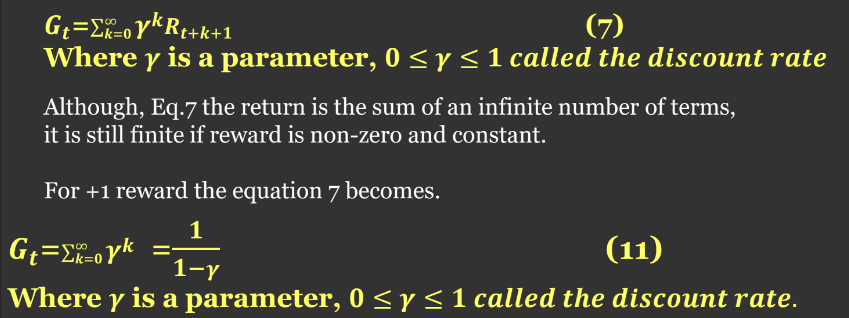






Taking γ out from second term





**How can we maximize the return?**

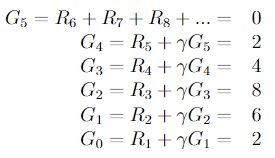
If we can perform correct action in each state, we can maximize the return.

**How can we perform correct action in each state?**

1. Using optimal policy correct action can be performed in each state.
2. Thus, optimal policy is the policy that gets our agent the maximum return (sum of rewards) by performing the correct action in each state.

**Suppose γ = 0.5 and the following sequence of rewards is received R1 = 1, R2 = 2, R3 = 6, R4 = 3, and R5 = 2, with T = 5. What are G0, G1, ..., G5?**

*Hint: Work backwards.*



TrashBot Finite MDP:

- TrashBot is operating in an environment where it needs to decide whether to actively search for waste, wait for waste to be brought to it, or recharge its battery.

- The system satisfies the Markov property as the future actions and outcomes depend solely on the current state.

- The system can be modeled as a finite Markov Decision Process (MDP) due to its discrete states s’, actions a, and rewards.

State Transition Diagram:

1. States: There are two distinct charge levels: high and low. S = {high, low}.

2. Actions: Depending on the current charge level, TrashBot can take different actions:

- In state "high": A(high) = {search, wait}

- In state "low": A(low) = {search, wait, recharge}

3. Rewards:

- Rewards for the TrashBot include positive values when it collects waste, a reward of 0 most of the time, and a negative reward (-3) if the battery gets depleted and it needs to be rescued.

Reward = {Rsearch, Rwait ,-3, 0}

4. Transition Probabilities:

- When the TrashBot is searching, the battery might get depleted, leading to a risk of needing to be rescued. The probabilities of these transitions depend on the charge level and the action chosen.

Probabilities = { α , 1 − α, β, 1 − β, 0, 1}

- The transition probabilities include:

- α: Probability of staying high after searching from high.

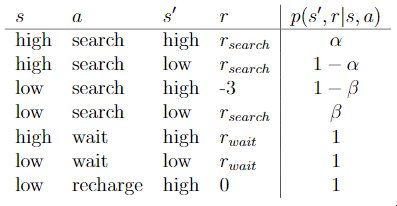
- 1 - α: Probability of going from high to low after searching.

- β: Probability of staying low after searching from low.

- 1 - β: Probability of depleting the battery after searching from low.

- 0: No transition when recharging.

- 1: Certain transition when being rescued.



**Version 2**:

\*\***TrashBot Finite MDP**:\*\*

- TrashBot is operating in an environment where it needs to decide whether to actively search for waste, wait for waste to be brought to it, or recharge its battery.

- The system satisfies the Markov property as the future actions and outcomes depend solely on the current state.

- The system can be modeled as a finite Markov Decision Process (MDP) due to its discrete states, actions, and rewards.

\*\***State Transition Diagram**:\*\*

1. \*\*States:\*\* There are two distinct charge levels: high and low. These serve as the states of the TrashBot: S = {high, low}.

2. \*\*Actions:\*\* Depending on the current charge level, TrashBot can take different actions:

- In state "high": A(high) = {search, wait}

- In state "low": A(low) = {search, wait, recharge}

3. \*\*Rewards:\*\*

- Rewards for the TrashBot include positive values when it collects waste, a reward of 0 most of the time, and a negative reward (-3) if the battery gets depleted and it needs to be rescued.

4. \*\*Transition Probabilities:\*\*

- When the TrashBot is searching, the battery might get depleted, leading to a risk of needing to be rescued. The probabilities of these transitions depend on the charge level and the action chosen.

- The transition probabilities include:

- α: Probability of staying high after searching from high.

- 1 - α: Probability of going from high to low after searching.

- β: Probability of staying low after searching from low.

- 1 - β: Probability of depleting the battery after searching from low.

- 0: No transition when recharging.

- 1: Certain transition when being rescued.

\*\*State Transition Diagram:\*\*

```

(high) (low)

+--------+ +--------+

| search | --> | search |

+---+----+ +---+----+

| |

v v

+---+----+ +--------+

| wait | | wait |

+---+----+ +---+----+

| |

v v

+-----> (high) -----> (low)

recharge rescue

```

This diagram shows the possible state transitions and the actions that lead to those transitions. The values of transition probabilities and rewards can be filled in based on the given probabilities and rewards associated with each action and state transition.

**How is the discount factor helping?**

It helps in preventing the return from reaching infinity by deciding how much importance we give to future and immediate rewards.

Equation of Reward at each time step is weighted by a discount factor.

* The value of discount factor ranges from 0 to 1.
* When discount factor is small value (close to 0) it implies more importance given to

immediate rewards than future rewards.

* When discount factor is high (close to 1) it implies much importance given to future

rewards than to immediate rewards.

Case - 1: A small discount factor

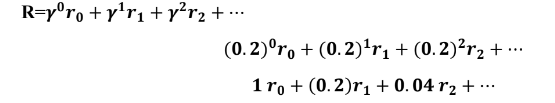
Case – 2: Large discount factor

Case – 3: Set to zero discount factor

Case – 4: Set discount factor to 1.

**Case - 1: A small discount factor**

Consider the discount factor γ = 0.2

****

Observation – 1

1. As time step increases, the discount factor (weight) decreases, thus importance of

future rewards also decrease.

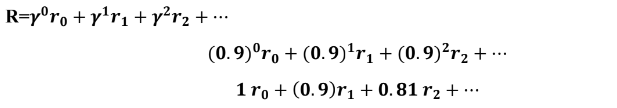
1. At time step 0, reward r0 is weighted by discount factor of 1.
2. At time step 1, reward r1 is weighted by γ =0.2.
3. At time step 2, reward r2 is weighted by γ =0.04.

Thus it can be observed that discount factor is heavily decreased for subsequent time steps and more importance is given to immediate reward r0 than the rewards obtained at future time steps.

When immediate reward is more important set discount factor to a small value.

Case – 2: Large discount factor

Let the discount factor set to a high value say γ = 0.9



From the above equations it can be observed that

Observation – 2

As time step increases, the discount factor (weight) decreases. However, it is slightly

decreasing not drastically (heavily).

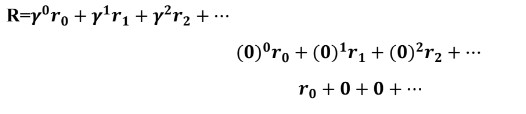
When future reward is more important set discount factor to a high value.

 At time step 1, reward r1 is weighted by slightly decreased discount factor of 0.9.

 At time step 2, reward r2 is weighted by slightly decreased γ =0.81.

Unlike previous case, discount factor is not decreased heavily.

Case – 3: Set to zero discount factor

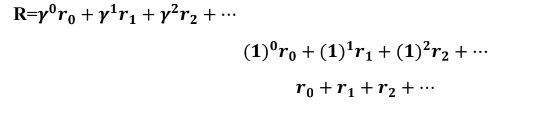


Thus, **R=r0**

**Observation – 3**

When γ = 0 return is just the immediate reward r0

Case – 4: Set discount factor to 1.



**R=r0 + r1 + r2 + ⋯**

**Observation – 4**

When γ = 1, return is the sum of rewards up to infinity.

**Observation – 5 In an environment, an agent might look either for immediate or future rewards.**

When γ = 0 agent never learn, considering only the immediate reward.

When γ = 1 agent learn forever looking for future rewards that leads to infinity.

**What is the significance of immediate and future rewards?**

In some tasks future reward is more desirable than immediate rewards.

For example, Chess game: Defeating the opponent’s king need to know all moves. Hence future reward is important in this case.

On the other hand, would you prefer a promotional gift or an introductory offer given today or 100 days later? Immediate reward is important in this case.

**Version 2**:

\*\***Discount Factor and Its Significance**:\*\*

The discount factor (denoted by γ) plays a crucial role in reinforcement learning by influencing how much importance an agent gives to immediate rewards compared to future rewards. Here's a breakdown of its significance in different cases:

\*\*Case 1: Small Discount Factor (Immediate Reward Focus):\*\*

- When γ is small (close to 0), more importance is given to immediate rewards.

- Future rewards are heavily discounted as time steps increase.

- Immediate reward (r0) is the most important, while future rewards (r1, r2, ...) are much less important.

- Useful when the agent's primary focus is on maximizing immediate gains.

- Equation: Gt = r0 + γ \* r1 + γ^2 \* r2 + ...

\*\*Case 2: Large Discount Factor (Future Reward Focus):\*\*

- When γ is large (close to 1), more importance is given to future rewards.

- Future rewards remain relatively significant even as time steps increase.

- Agent is more concerned with long-term rewards and consequences.

- Equation: Gt = r0 + γ \* r1 + γ^2 \* r2 + ...

\*\*Case 3: Zero Discount Factor (Immediate Reward Only):\*\*

- When γ is set to 0, only the immediate reward matters.

- Agent disregards any future rewards and focuses solely on the current state.

- Equation: Gt = r0

\*\*Case 4: Discount Factor Equal to 1 (Infinite Focus on Future Rewards):\*\*

- When γ is set to 1, all future rewards are treated as equally important as immediate rewards.

- Agent considers infinite time horizon for rewards.

- Equation: Gt = r0 + r1 + r2 + ...

\*\***Significance of Immediate and Future Rewards**:\*\*

- Different tasks require different balances between immediate and future rewards.

- In tasks like chess, where long-term planning is essential, future rewards hold great importance (high γ).

- In situations where instant gratification matters, immediate rewards are prioritized (low γ).

- The choice of γ reflects the agent's strategy based on the task's nature.

\*\***Equality Proof**:\*\*

Starting with Gt = Rt+1 + γ(Gt+1), and assuming a reward sequence of +1, we prove the equality by following these steps:

1. Substitute Rt+1 = 1 and Gt+1 = G into the equation.

2. Solve for G.

3. Rearrange terms to show that G = 1 / (1 - γ).

\*\***Example Calculation**:\*\*

Suppose γ = 0.9 and the reward sequence is R1 = 2 followed by an infinite sequence of 7s.

1. Calculate G1 using the infinite geometric series formula.

G1 = ∑ 7 \* γ^k, where k ranges from 0 to ∞.

G1 = 7 \* (1 / (1 - 0.9)) = 70.

2. Calculate G0 using the equation Gt = Rt+1 + γ(Gt+1).

G0 = 2 + 0.9 \* 70 = 65.

These calculations demonstrate how the discount factor affects the accumulation of rewards over time, based on the given values of γ and the reward sequence.

**What is difference between MAB and MDP?**

**Goal:**

The agent's goal is to find the arm with the highest average reward.

The agent's goal is to find a policy that maximizes its expected return over time.

**MABs are stateless**:

MAB are stateless, the agent does not have any information about its previous actions or rewards.

MDP includes states, the agent has full knowledge of its previous states and actions for future rewards.

**MABs have a fixed number of arms:**

In an MAB, there is a fixed number of arms that the agent can pull.

In an MDP, the number of states and actions can vary depending on the environment.

**MABs are typically used for discrete problems:**

MABs are often used for **discrete** problems where the agent can only choose from a finite number of actions.

MDPs can be used for both discrete and continuous problems.

**There is no state transition in MAB where as there is a state transition in MDP**

**Example:**

For example, an MAB could be used to choose which ad to display to a user. For example, an MDP could be used to control a robot in a physical environment.

Point – 1

1. The goal in MAB is either to identify the arm with highest mean (best arm) with the

smallest number of arm pulls or to maximize the cumulative reward of the agent.

1. An MDP is more complex. The action you choose determines a new state in which you have completely different distribution (best arms).

Point -2

1. In MAB, RL setting can be interpreted as a one-state MDP (thus MAB is stateless).

In MAB you are given a set of k distributions (Whose structure is usually known while

the mean of the distribution is unknown).

1. In MAB there is no state transitions, always to be in same state.

**Version 2** :

Multi-Armed Bandits (MAB) and Markov Decision Processes (MDP) into simple and detailed points for easy understanding:

\*\*Multi-Armed Bandits (MAB):\*\*

Point - 1:

- \*\*Goal:\*\* In MAB, the main goal is to either find the arm (action) with the highest average reward (best arm) with the fewest pulls or to maximize the total reward obtained by the agent.

- MAB focuses on exploring different arms to understand their rewards and exploiting the arms that seem to provide higher rewards.

Point - 2:

- \*\*Interpretation:\*\* MAB can be seen as a simplified version of Reinforcement Learning (RL) where there's just one state. It's like facing multiple slot machines (arms) in a casino where each pull gives a reward.

- In MAB, you are presented with a set of arms, each associated with an unknown reward distribution. The challenge is to decide which arms to pull to maximize your overall reward.

- In MAB, there are no state transitions. You're always in the same context (state), deciding which arm to pull next.

\*\*Markov Decision Processes (MDP):\*\*

- \*\*Goal:\*\* In MDP, the goal is to make a sequence of decisions to maximize the cumulative reward over time. The decisions you make affect not only immediate rewards but also the future states and rewards.

- MDP involves decision-making over time, considering how your actions in one state influence your experiences in future states.

- MDP accounts for the idea that the environment can change based on the actions you take, leading to different states and outcomes.

- MDP is more complex than MAB since it deals with a sequence of interconnected states, actions, and rewards, and aims to optimize long-term outcomes.

\*\*In simple words:\*\*

- \*\*MAB\*\* is like pulling levers (arms) on slot machines, trying to figure out which lever gives the best payout. It's about balancing exploring new levers and exploiting ones that seem promising.

- \*\*MDP\*\* is like playing a strategic game, where your decisions in one turn affect the situation in the next turn. It's about finding the best sequence of actions to maximize your overall gains while considering how your choices influence the future states you encounter.

To sum up, MAB is about choosing the best action in a single context, while MDP involves making a series of actions in different states to achieve a long-term goal.