SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

23AMB201 - MACHINE LEARNING

II YEAR IV SEM

UNIT V – REINFORCEMENT LEARNING

TOPIC 4 – Model Based Learning – Model Free Learning

What is an MDP?

A Markov Decision Process provides a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of the agent.

The agent learns an optimal strategy by interacting with the environment, receiving feedback, and refining its actions based on the results.

A framework for modeling decision making in stochastic environments.

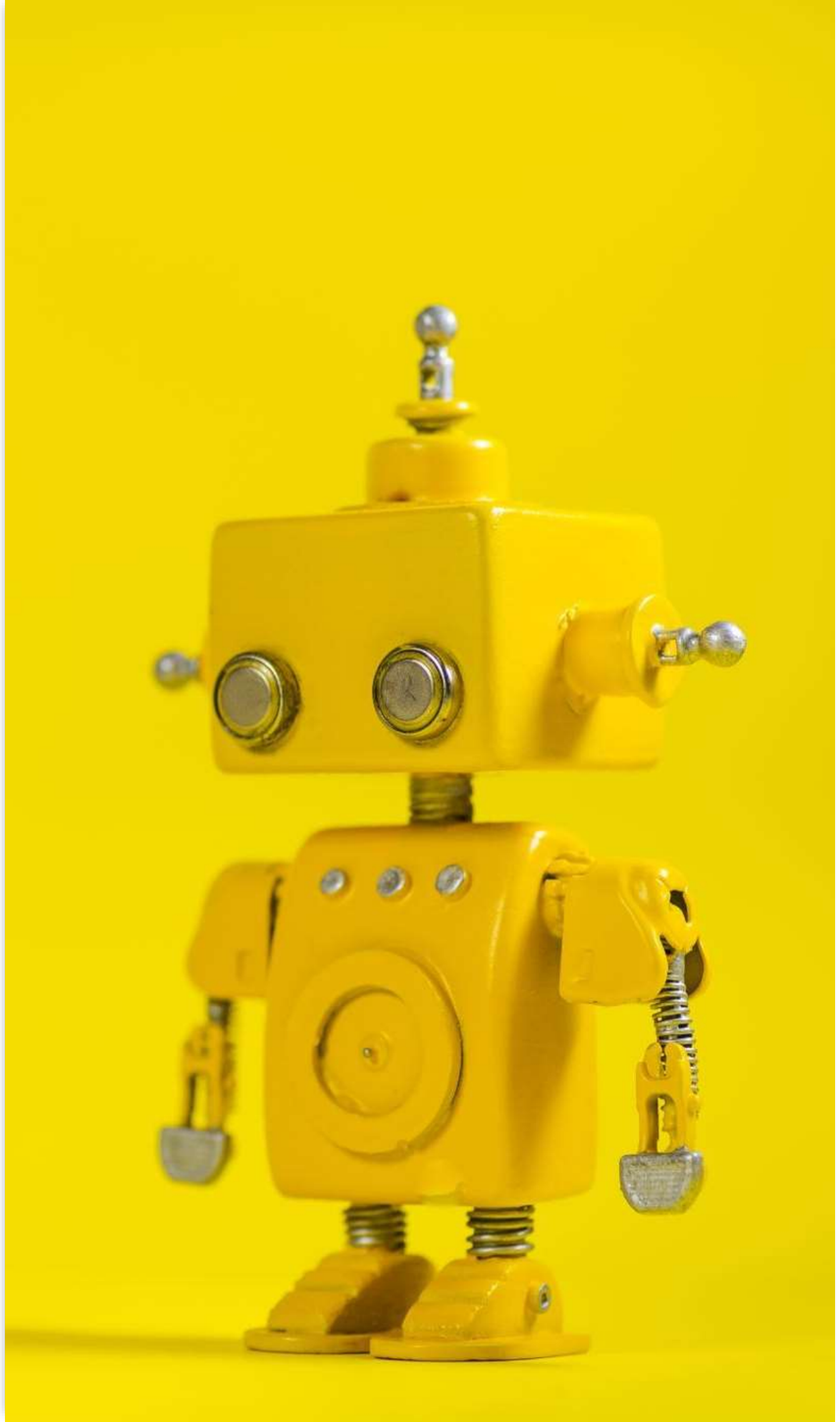
An agent interacts with an environment over time.

It selects actions to maximize cumulative reward.

It's often used in reinforcement learning to find optimal policies for an agent to maximize rewards

Why Do We Need MDPs?

The environment changes based on the agent's **actions**.
The environment has **uncertainty** (taking the same action doesn't always lead to the same result).
We want to **optimize behavior over time**, not just in the current step.
Examples:
A robot navigating a maze.
An AI playing chess or Go.
A delivery drone choosing an efficient and safe path.
Designing quiz games
Determining the number of patients to admit to a hospital
Managing wait time at a traffic intersection



Formal Definition

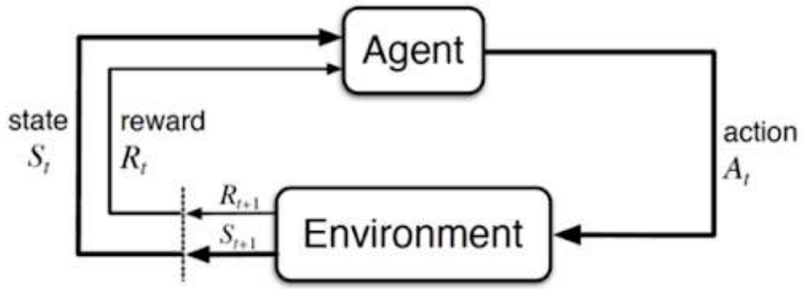
An MDP is defined by a 5-tuple:

$$MDP = (S, A, P, R, \gamma)$$

S – Set of states
A – Set of actions
P(s'|s,a) – Transition probability: probability of moving to state (**s′**) from state (**s**) after action a
R(s,a) – Reward function: expected reward for taking action(**a**) in state (**S**)
γ (gamma) – Discount factor: **0≤γ≤1**, controls the importance of future rewards

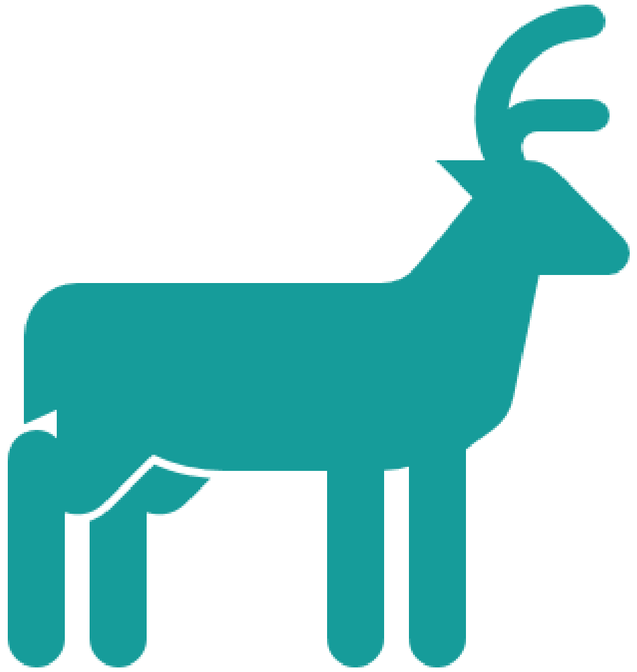
How MDPs Work (Step-by-Step)

1. Observe state
2. Take action
3. Get reward and next state s_{t+1}
4. Repeat



Real-world example to understand the working of MDP

We have a problem where we need to decide whether the tribes should go deer hunting or not in a nearby forest to ensure long-term returns. Each deer generates a fixed return. However, if the tribes hunt beyond a limit, it can result in a lower yield next year. Hence, we need to determine the optimum portion of deer that can be caught while maximizing the return over a longer period.
The problem statement can be simplified in this case: whether to hunt a certain portion of deer or not. In the context of MDP, the problem can be expressed as follows:



➢ States

The number of deer available in the forest in the year under consideration. The four states include empty, low, medium, and high, which are defined as follows-

Empty: No deer available to hunt

Low: Available deer count is below a threshold t_1

Medium: Available deer count is between t_1 and t_2

High: Available deer count is above a threshold t_2

➢ Action

Actions include go-hunt and no-hunting, where go-hunt implies catching certain proportions of deer. It is important to note that for the empty state, the only possible action is no-hunting.

➢ Rewards

- Hunting at each state generates rewards of some kind.
- The rewards for hunting at different states, such as state low, medium, and high, maybe **$5K, $50K, and $100k**, respectively.
- Moreover, if the action results in an empty state, **the reward is -$200K**. This is due to the required e-breeding of new deer, which involves time and money.

➢ State transitions

- Hunting in a state causes the transition to a state with fewer deer.
- Subsequently, the action of no_hunting causes the transition to a state with more deer, except for the ‘high’ state.

Markov Property

The decision process satisfies the **Markov Property**, which means the future is independent of the past given the present

$$P[St+1|St] = P[St+1 |S1,S2,S3……St]$$

According to this equation, the probability of the next state *(P[St+1])* given the present state *(St)* is given by the next state's probability *(P[St+1])* considering all the previous states *(S1,S2,S3……St)*

Imagine that a robot sitting on a chair stood up and put its right foot forward. So, at the moment, it's standing with its right foot forward. This is its current state.

Now, according to the Markov property, the current state of the robot depends only on its immediate previous state (or the previous timestep,) i.e. the state it was in when it stood up. And evidently, it doesn't depend on its prior state– sitting on the chair. Similarly, its next state depends only on its current state.

$$V(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t | s_t]$$

Policy (π)

π(a|s): Probability of taking action a in state s.
Defines agent's behavior.
Goal: Find optimal policy π* maximizing expected return.

Value Functions

State-value function $V^{\pi}(s)$: Expected return from state(s) under policy π.
Action-value function $Q^{\pi}(s, a)$: Expected return from taking action (a) in state(s) under π.
The value function can be divided into two components: the reward of the current state and the discounted reward value of the next state
This breakdown derives **Bellman's equation**, as shown next

Bellman Equations

Value function:

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

Optimal value function (used in Value Iteration)

$$\Pi^*(s) = argmax_a[R(s,a) + \gamma \sum_{s' \epsilon S} P^a_{ss'} V(s')]$$

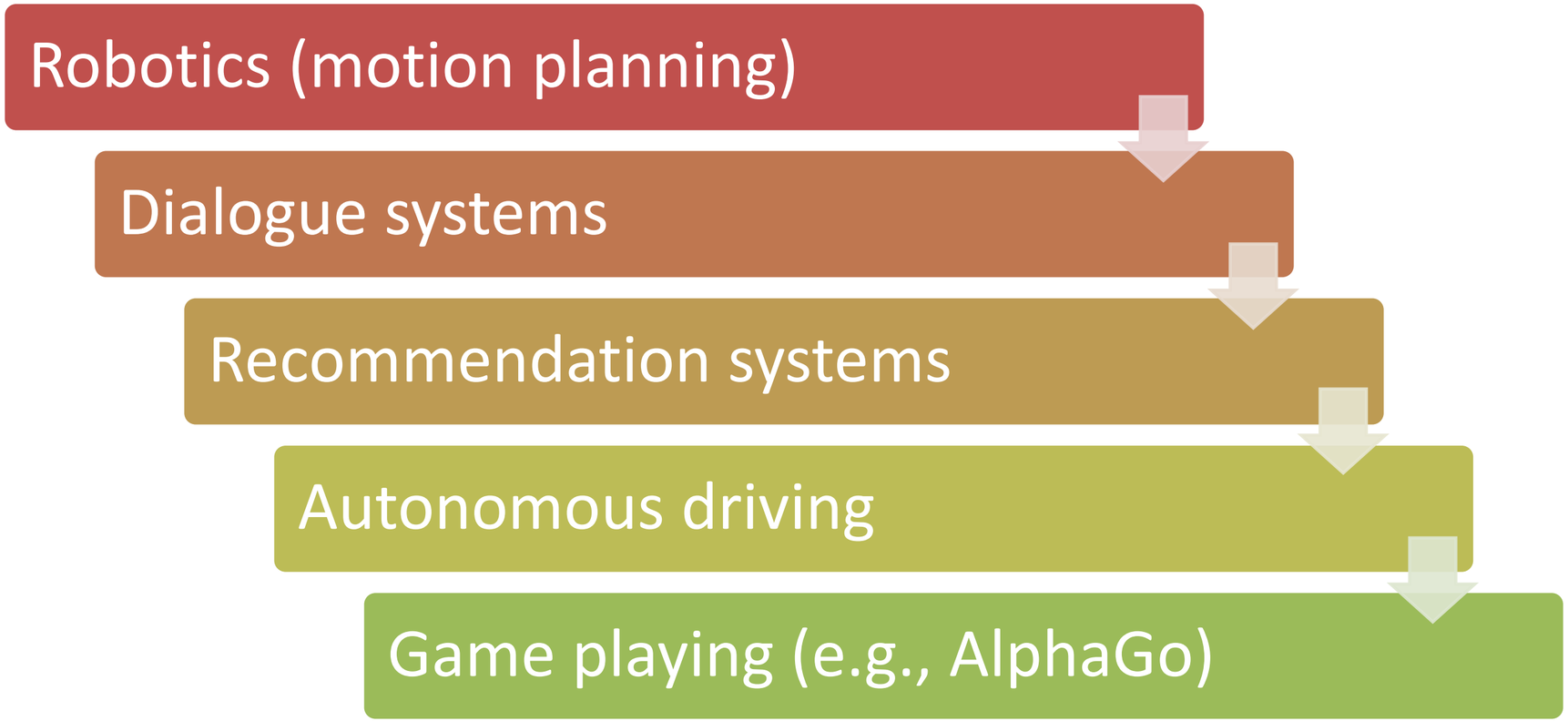
➢Optimal value function can be solved with the help of iterative methods such as Monte-Carlo evaluations, dynamic programming, or temporal-difference learning.

➢The policy that chooses the maximum optimal value while considering the present state is referred to as the optimal policy.

Solving MDPs

Value Iteration: Iteratively compute V*.
Policy Iteration: Alternate evaluation & improvement.
Q-learning (Model-free RL)
Monte Carlo Methods
Deep Reinforcement Learning (when state/action spaces are too large)

Applications



If **γ=0** only immediate rewards matter.

If **γ close to 1**: Future rewards are important.

Helps balance **short-term** vs **long-term gains**

Example Calculation

State
S = {s1, s2}, A = {a1, a2}
$\gamma = 0.9$
Rewards -
R(s1, a1) = 5
R(s1, a2) = 10
Transitions:
P(s2|s1,a1)=1
P(s1|s1,a2)=1
V(s2)=0
We calculate –
$Q(s1,a1)=R(s1,a1)+\gamma V(s2)=5+0.9\cdot 0=5$
$Q(s1,a2)=R(s1,a2)+\gamma V(s1)=10+0.9\cdot V(s1)$

Bellman Equation for Value Function

Let's say we have:
States: S={s1,s2}
Actions: A={a1,a2}
Transition Probabilities

P(s1|s1,a1)=0.5 , P(s2|s1,a1)=0.5
P(s1|s2,a1)=0.7, P(s2|s2,a1)=0.3

Rewards:

R(s1,a1)=5
R(s2,a1)=10

Discount factor: γ=0.9
Policy : Always take action a1 (i.e., π(a1|s)=1)

Apply Bellman Equation:

We want to find V(s1)V(s_1)V(s1) and V(s2)V(s_2)V(s2)

Step 1: Write Bellman equations

$$V(s1)=R(s1,a1)+\gamma[P(s1|s1,a1)V(s1)+P(s2|s1,a1)V(s2)]$$

$$V(s2)=R(s2,a1)+\gamma[P(s1|s2,a1)V(s1)+P(s2|s2,a1)V(s2)]$$

Step 2: Plug in the values

$V(s1) = 5+0.9[0.5V(s1)+0.5V(s2)]$

$V(s2)=10+0.9[0.7V(s1)+0.3V(s2)]$

Step 3: Solve the equations

$V(s1)=5+0.45V(s1)+0.45V(s2)$

$V(s1) - 0.45V(s1) = 5+0.45V(s2) \quad \Rightarrow \quad 0.55V(s1) = 5+0.45V(s2)$

$V(s2) = 10+0.63V(s1)+0.27V(s2)$

$V(s2) - 0.27V(s2) = 10 + 0.63V(s1) \quad \Rightarrow \quad 0.73V(s2) = 10+0.63V(s1)$

From (3):

$$V(s_1) = \frac{5 + 0.45V(s_2)}{0.55}$$

Now plug into (4):

$$0.73V(s_2) = 10 + 0.63\left(\frac{5 + 0.45V(s_2)}{0.55}\right)$$

Solve this algebraically:

$$0.73V(s_2) = 10 + \frac{0.63 \cdot (5 + 0.45V(s_2))}{0.55} \Rightarrow 0.73V(s_2) = 10 + \frac{3.15 + 0.2835V(s_2)}{0.55}$$

$$0.73V(s_2) = 10 + 5.727 + 0.5155V(s_2) \Rightarrow 0.73V(s_2) - 0.5155V(s_2) = 15.727 \Rightarrow 0.2145V(s_2) = 15.727 \Rightarrow V(s_2) \approx 73.34$$

Now find $V(s_1)$:

$$V(s_1) = \frac{5 + 0.45 \cdot 73.34}{0.55} \approx \frac{5 + 33.003}{0.55} \approx \frac{38.003}{0.55} \approx 69.1$$

Final Result

- V(s1) ≈ 69.1
- V(s2) ≈ 73.3

Case Study: Optimizing Elevator Operations using MDPs

Objective: Improve the efficiency and responsiveness of an elevator system in a multi-story building using Markov Decision Processes (MDPs).

Learning Outcomes:

- Participants will be able to model problems using MDPs
- Apply Bellman equations to compute state values
- Understand how MDP-based policies improve real-world systems like elevator control

1. Problem Overview:

- A building has multiple floors (e.g., 10).
- Multiple elevators serve requests from users wanting to **go up or down**.
- Objective: Minimize average waiting time and energy usage while ensuring service efficiency.

2. MDP **Formulation**

States (S):

Current floor of the elevator
Direction (up/down/idle)
Request queue (pending requests on each floor)

Actions (A):

Move up
Move down
Stay idle (do nothing for this timestep)
Open/close doors (optional granularity)

Transition Probabilities (P):

Probability of moving from one floor to another given an action
Changes based on request arrivals and elevator response

Rewards (R):

Negative reward for making passengers wait
Negative reward for energy consumed when moving
Positive reward for serving a request

Discount Factor (γ):

Typically set between 0.9 and 0.99 to prioritize long-term performance over immediate reward

3. Example **Scenario**

Elevator at floor 5, idle
Requests: someone on floor 3 wants to go up, another on floor 7 wants to go down
Action: decide whether to go to floor 3, 7, or stay idle
Bellman Equation:

$$V(s)=\text{Max}[R(s,a)+\sum P(s|s,a)V(s)]$$

4. Policy & Optimization:

Use value iteration or policy iteration to find the optimal policy

The optimal policy maps each elevator state to the best action

5. Benefits of MDP-Based Elevator Control:

Dynamically adapts to changes in demand patterns

Reduces unnecessary movements

Learns over time which strategies lead to higher efficiency

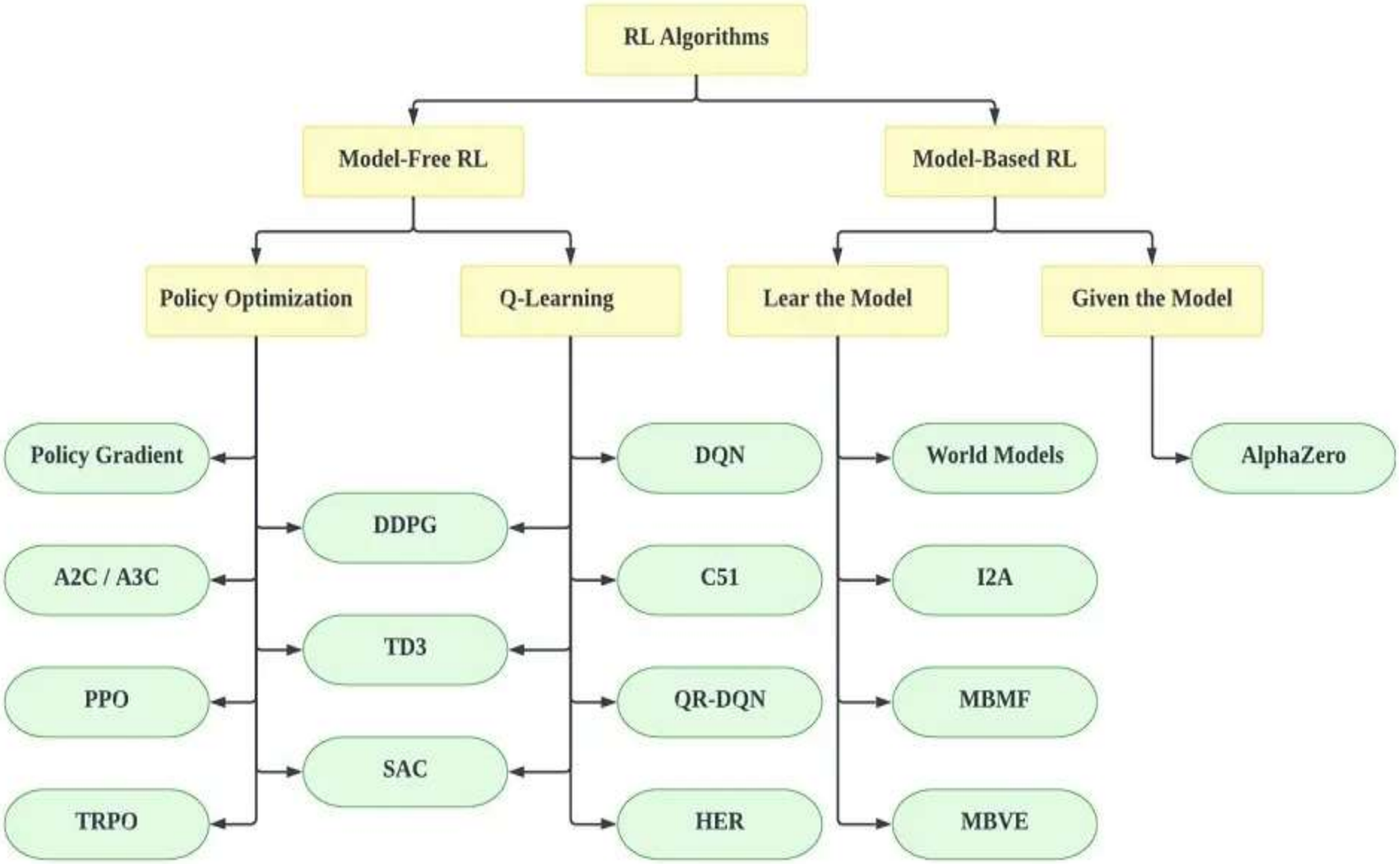
6. Challenges:

- Large state space due to many floors and requests
- Computational complexity
- Real-time response requirements

7. Solution Extensions:

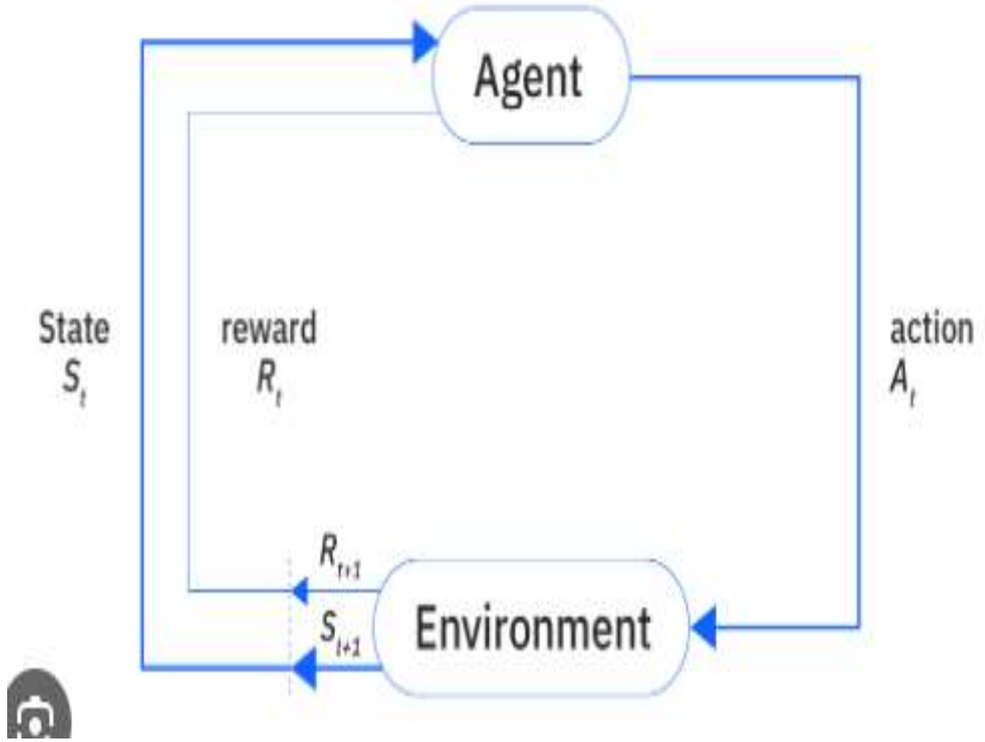
- ✓ Use Approximate Dynamic Programming (ADP) for large-scale systems.
- ✓ Deep Reinforcement Learning for continuous improvement
- ✓ Prioritize requests based on urgency or VIP status

Reinforcement Learning



Introduction to Reinforcement Learning

- Reinforcement learning is a type of machine learning where an **agent learns to make decisions** in a dynamic environment to maximize a reward signal.
- It uses a **trial-and-error** approach, where the agent takes actions, receives feedback in the form of rewards or penalties, and learns to improve its behavior over time.



Core Concepts

- **Agent**: The learner or decision-maker (e.g., a robot, software bot).
- **Environment**: The world the agent interacts with.
- **State (s)**: A snapshot of the current situation.
- **Action (a)**: What the agent can do.
- **Reward (r)**: Feedback from the environment (positive or negative).
- **Policy (π)**: The agent’s strategy, mapping states to actions.
- **Value Function (V)**: Predicts the future reward from a state.
- **Q-Function (Q)**: Predicts the future reward from a state-action pair.

How RL Works

- **Observes** the current state.
- **Receives** a reward and transitions to a new state.
- **Updates** its policy to maximize long-term rewards.
- **Chooses** an action based on its policy.

Types of RL

- **Positive Reinforcement Learning**
- **Negative Reinforcement Learning**
- **Model-Free RL**: Learns directly from experience (e.g. Q-learning, Deep Q-Networks).
- **Model-Based RL**: Builds a model of the environment to plan ahead.(e.g AlphaZero)
- **Value-Based**: Learns value functions (Q-learning).
- **Actor-Critic**: Combines both approaches. (A3C, PPO)
- **Policy-Based**: Learns policies directly (e.g. Policy Gradient).

Real-World Applications

1. Robotics
2. Game playing (e.g., AlphaGo, OpenAI Five)
3. Recommendation systems
4. Finance and trading
5. Autonomous vehicles
6. Stock trading


Application of Reinforcement Learning
Applications of Reinforcement Learning
Business
1
Gaming
2
Recomm-endation System
3
Science
4


Reinforcement Learning in ML
Input Raw Data
Environment
Output
Reward
Best Action
State
Selection of Algorithm
Agent
TechVidvan

Deep Q-Networks (DQN)

Combines Q-learning with neural networks.

Used in: Playing Atari games at human level!

Diagram: Input (game screen) → CNN → Q-values for each action

Example - Q-Learning

- Formula: **$Q(s, a) \rightarrow Q(s, a) + \alpha [r + \gamma \max Q(s', a') - Q(s, a)]$**
- Where:
- **α: learning rate**
- **γ: discount factor**
- **r: reward**
- **s': next state**