

Reinforcement Learning

Reinforcement Learning:

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize cumulative reward.

1. Key Elements of RL:

1. **Agent:** The decision-maker.
2. **Environment:** The world with which the agent interacts.
3. **State (s):** The current situation of the agent.
4. **Action (a):** The set of all possible moves the agent can make.
5. **Reward (r):** The feedback from the environment.

2. Process Flow:

1. The agent observes the current state.
2. It selects an action based on its policy.
3. The action affects the environment, transitioning to a new state.
4. The agent receives a reward based on the action and the new state.
5. The process repeats, and the agent learns to optimize its actions over time.

3. Objective of RL:

1. To find a policy that maximizes the expected cumulative reward over time.

4. Applications of RL:

1. Examples include robotics, game playing (e.g., AlphaGo), autonomous vehicles, and finance.

Overview of RL: Definition and Key Concepts

1. Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make decisions by interacting with an environment to achieve a goal.

1. Key Concepts:

1. **Learning from Interaction:** The agent learns by taking actions and receiving feedback.
2. **Goal-Oriented Learning:** The aim is to maximize cumulative reward over time.
3. **Trial-and-Error:** The agent explores different actions to discover the best strategy.
4. **Delayed Reward:** The effect of actions may not be immediate, requiring the agent to learn long-term strategies.

2. Comparison with Other Learning Types:

1. **Supervised Learning:** Learning from labeled data.
2. **Unsupervised Learning:** Finding patterns in data without labels.
3. **RL:** Learning from interaction with the environment to maximize rewards.

3. Key Challenges in RL:

1. **Exploration vs. Exploitation:** Balancing trying new actions and leveraging known rewards.
2. **Credit Assignment:** Determining which actions lead to high rewards.
3. **Scalability:** Handling large state and action spaces.

Components of RL: Agent, Environment, State, Action, Reward

1. Agent:

1. The learner or decision-maker that interacts with the environment.
2. *Example:* A robot, a game-playing algorithm.

2. Environment:

1. The external system with which the agent interacts.
2. *Example:* A physical world for a robot, a game board for an algorithm.

3. State (s):

1. A representation of the current situation of the agent within the environment.
2. *Example:* The position and velocity of a robot, the current board configuration in a game.

4. Action (a):

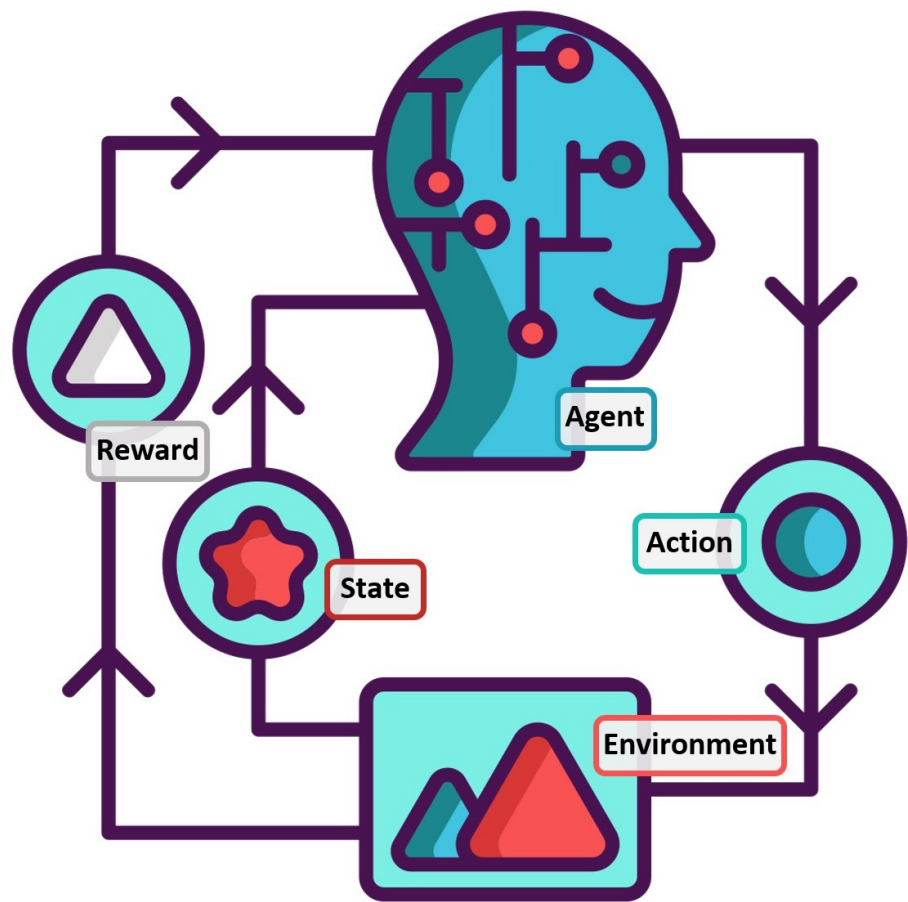
1. The set of all possible moves or decisions the agent can make.
2. *Example:* Moving forward or backward, placing a piece in a game.

5. Reward (r):

1. The feedback from the environment based on the action taken.
2. *Example:* A point score in a game, a measure of performance for a task.

6. Interaction Flow:

1. The agent observes the current state s .
2. Chooses an action a based on a policy.
3. The environment transitions to a new state s' .
4. The agent receives a reward r .



Markov Decision Process (MDP): Definition and Components

1. A Markov Decision Process (MDP) is a mathematical framework used to describe an environment in reinforcement learning, where outcomes are partly random and partly under the control of a decision-maker (agent).

1. Components of MDP:

1. States (S):

1. A finite set of states that represent all possible situations the agent can be in.
2. *Example:* Positions on a game board, different locations a robot can occupy.

2. Actions (A):

1. A finite set of actions available to the agent from each state.
2. *Example:* Move left, right, up, down in a grid world.

3. Transition Model (P):

1. The probability $P(s'|s,a)$ of transitioning to state s' from state s when action a is taken.
2. *Example:* Probability of moving to a neighboring cell in a grid.

4. Reward Function (R):

1. The immediate reward received after transitioning from state s to state s' due to action a .
2. *Example:* Points gained or lost after a move in a game.

5. Policy (π):

1. A strategy or rule that defines the action $a=\pi(s)$ the agent takes in each state s .
2. *Example:* A set of rules guiding a robot's movement based on its current position.

1. Properties of MDP:

1. Markov Property:

1. The future state depends only on the current state and action, not on the sequence of events that preceded it.

2. Formulation:

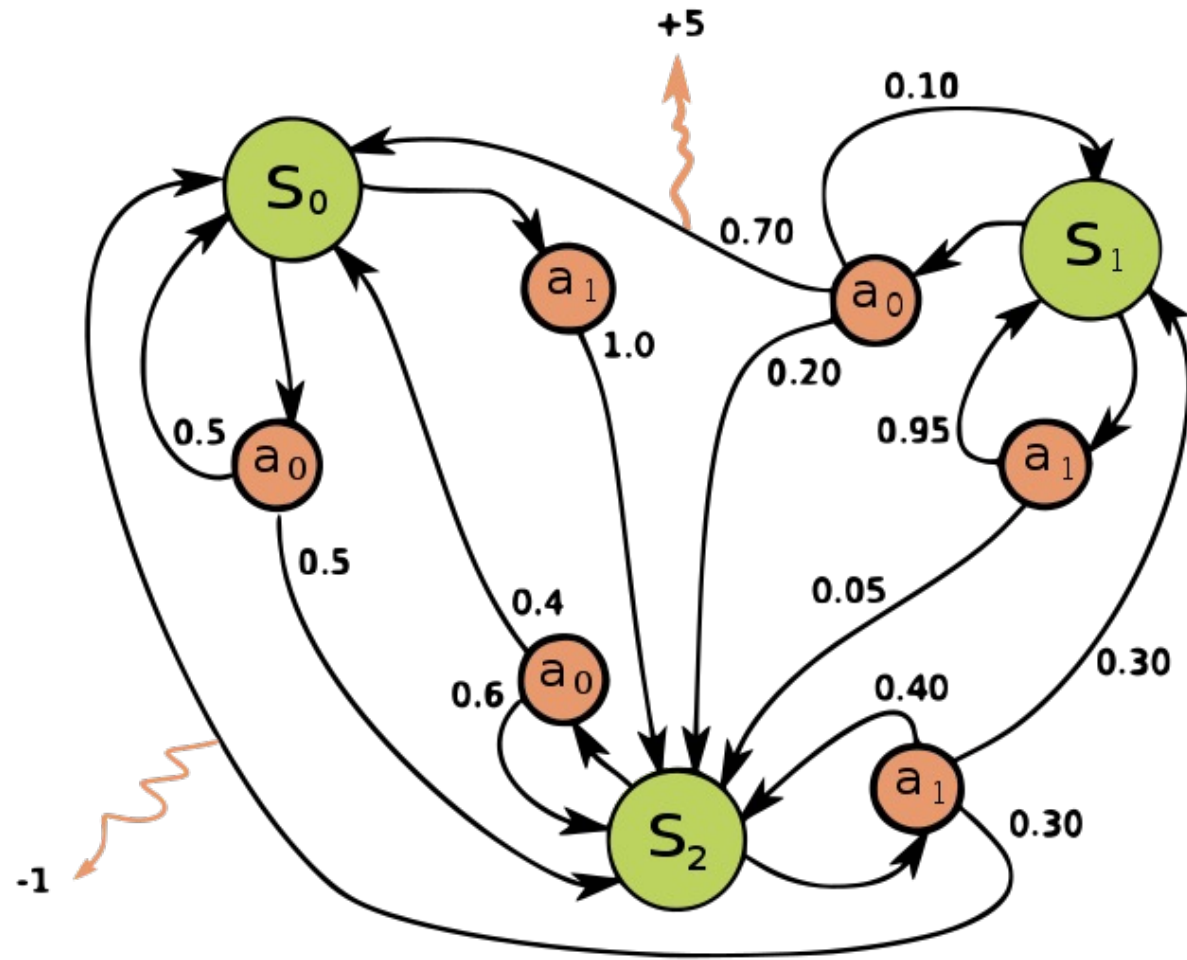
1. Tuple Representation: An MDP is typically represented as a tuple (S, A, P, R, γ) where:

1. S = Set of states.
2. A = Set of actions.
3. P = Transition probability matrix.
4. R = Reward function.
5. γ = Discount factor ($0 \leq \gamma < 1$), representing the importance of future rewards.

3. Example:

1. Grid World:

1. States: Each cell in a grid.
2. Actions: Move up, down, left, right.
3. Transition Model: Probabilities of moving to adjacent cells.
4. Rewards: Points for reaching specific cells.



Policy: Definition and Types (Deterministic vs. Stochastic)

1. A policy in reinforcement learning is a strategy or a mapping from states to actions that defines the behavior of an agent.

1.Types of Policies:

1. Deterministic Policy:

1. A deterministic policy directly maps each state to a specific action.

2. **Notation:** $\pi(s)=a$

3. Characteristics:

1. Given a state s , the action a is chosen with certainty.

2. **Example:** If an agent is in state s , it always takes action a (e.g., always move right in a grid world).

2. Stochastic Policy:

1. A stochastic policy provides a probability distribution over actions for each state.

2. **Notation:** $\pi(a|s)=P(a|s)$

3. Characteristics:

1. Given a state s , the action a is chosen according to a probability distribution.

2. **Example:** If an agent is in state s , it might move right with a probability of 0.7 and move left with a probability of 0.3.

1. Formal Definitions:

1. Deterministic Policy:

1. A function $\pi:S \rightarrow A$ where $\pi(s)=a$.

2. Stochastic Policy:

1. A function $\pi:S \times A \rightarrow [0,1]$ where $\pi(a|s)=P(a|s)$.

2. Examples:

1. Deterministic:

1. *Robot Navigation*: If the robot is at (2,3), it always moves to (2,4).

2. Stochastic:

1. *Game Strategy*: If in a certain game state, the agent might attack with a probability of 0.8 and defend with a probability of 0.2.

3. Advantages and Use Cases:

1. Deterministic Policies:

1. Simplicity and ease of implementation.
2. Useful when actions have predictable outcomes.

2. Stochastic Policies:

1. Useful in environments with uncertainty or noise.
2. Facilitate exploration by preventing the agent from being stuck in local optima.

Value Functions: State-Value and Action-Value Functions

1. Value functions estimate the expected return (total accumulated reward) from a given state or state-action pair.

1.State-Value Function $V(s)$:

1. Definition:

1. The state-value function $V(s)$ gives the expected return starting from state s and following a particular policy π .

2. Mathematical Formulation:

1. $V\pi(s)=E[Gt|St=s]$
2. Where Gt is the total return from time step t , and $St=s$ indicates the agent is in state s at time t .

3. Interpretation:

1. Measures the long-term value of being in a state s under policy π .

1. Action-Value Function $Q(s,a)$:

1. Definition:

1. The action-value function $Q(s,a)$ gives the expected return starting from state s , taking action a , and then following a particular policy π .

2. Mathematical Formulation:

1. $Q\pi(s,a) = E[G_t | S_t=s, A_t=a]$
2. Where G_t is the total return from time step t , and $S_t=s, A_t=a$ indicates the agent is in state s and takes action a at time t .

3. Interpretation:

1. Measures the long-term value of taking action a in state s under policy π .

1. Relationship Between $V(s)$ and $Q(s,a)$:

1. The state-value function $V(s)$ can be derived from the action-value function $Q(s,a)$:
 1. $V(s) = \sum_a \pi(a|s) Q(s,a)$
2. This equation shows that the value of a state is the expected value of the action-values, weighted by the policy.

2. Examples:

1. Grid World:

1. **State-Value $V(s)$:** The expected return from a specific cell in the grid.
2. **Action-Value $Q(s,a)$:** The expected return from a specific cell when taking a particular direction (e.g., moving right).

- https://www.youtube.com/watch?v=0ephTkEiCko&ab_channel=BenjaminSchumann
- https://www.youtube.com/watch?v=kopoLzvh5jY&ab_channel=OpenAI

Bellman Equation

1. Introduction to the Bellman Equation:

1. The Bellman Equation is a fundamental recursive formula used in reinforcement learning to calculate the value of a state or state-action pair.
2. Named after Richard Bellman, it provides a relationship between the value of a state and the values of its successor states.

2. Purpose of the Bellman Equation:

1. To break down the value function into the immediate reward and the discounted value of the next state.
2. Facilitates the computation of value functions by leveraging the principle of optimality.

3. State-Value Function Bellman Equation:

1. **Mathematical Formulation:** $V^\pi(s) = E[R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s]$

2. Explanation:

1. $V^\pi(s)$: Value of state s under policy π .
2. R_{t+1} : Reward received after taking an action in state s .
3. γ : Discount factor ($0 \leq \gamma < 1$), which accounts for the importance of future rewards.
4. $V^\pi(S_{t+1})$: Value of the next state S_{t+1} .

1. Action-Value Function Bellman Equation:

1. **Mathematical Formulation:** $Q\pi(s,a) = E[R_{t+1} + \gamma Q\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$

2. **Explanation:**

1. $Q\pi(s,a)$: Value of taking action a in state s under policy π .
2. R_{t+1} : Reward received after taking action a in state s .
3. γ : Discount factor.
4. $Q\pi(S_{t+1}, A_{t+1})$: Value of the next state-action pair (S_{t+1}, A_{t+1}) .

2. Principle of Optimality:

1. The Bellman Equation leverages the principle of optimality, which states that an optimal policy has the property that, whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

3. Recursive Nature:

1. The Bellman Equation's recursive nature simplifies the problem of finding the value function by breaking it down into smaller subproblems.

4. Example:

1. **Grid World:**

1. **State-Value:** For a state s , the value is calculated based on the immediate reward and the expected value of subsequent states.
2. **Action-Value:** For a state-action pair (s,a) , the value is calculated based on the immediate reward and the expected value of the subsequent state-action pairs.

Bellman Expectation Equation

1. The Bellman Expectation Equation extends the Bellman Equation to evaluate the expected value of a state or state-action pair under a specific policy π .

1.State-Value Function Bellman Expectation Equation:

1. **Mathematical Formulation:** $V\pi(s) = \sum \pi(a|s) \sum P(s'|s,a) [R(s,a,s') + \gamma V\pi(s')]$

2. Explanation:

1. $V\pi(s)$: Value of state s under policy π .
2. $\pi(a|s)$: Probability of taking action a in state s under policy π .
3. $P(s'|s,a)$: Probability of transitioning to state s' from state s after taking action a .
4. $R(s,a,s')$: Reward received after transitioning from state s to state s' due to action a .
5. γ : Discount factor.
6. $V\pi(s')$: Value of the next state s' .

1.Action-Value Function Bellman Expectation Equation:

1. **Mathematical Formulation:** $Q\pi(s,a)=\sum P(s'|s,a)[R(s,a,s')+\gamma \sum \pi(a'|s')Q\pi(s',a')]$

2. Explanation:

1. $Q\pi(s,a)$: Value of taking action a in state s under policy π .
2. $P(s'|s,a)$: Probability of transitioning to state s' from state s after taking action a .
3. $R(s,a,s')$: Reward received after transitioning from state s to state s' due to action a .
4. γ : Discount factor.
5. $Q\pi(s',a')$: Value of the next state-action pair (s',a') under policy π .

2.Process Flow:

1. Calculate the expected value of the immediate reward plus the discounted value of subsequent states, weighted by the policy and transition probabilities.

3.Example:

1. Grid World:

1. Evaluate the expected value of being in a cell considering all possible actions and subsequent states.

Bellman Optimality Equation

1. The Bellman Optimality Equation is used to find the optimal value function by maximizing the expected return over all possible actions.

1.State-Value Function Bellman Optimality Equation:

1. **Mathematical Formulation** $V^*(s) = \max_a (s' \sum P(s'|s,a)[R(s,a,s') + \gamma V^*(s')])$

2. Explanation:

1. $V^*(s)$: Optimal value of state s .
2. \max_a : Maximization over all possible actions a .
3. $P(s'|s,a)$: Probability of transitioning to state s' from state s after taking action a .
4. $R(s,a,s')$: Reward received after transitioning from state s to state s' due to action a .
5. γ : Discount factor.
6. $V^*(s')$: Optimal value of the next state s' .

1. Action-Value Function Bellman Optimality Equation:

1. **Mathematical Formulation:** $Q^*(s,a) = \sum P(s'|s,a) [R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$

2. Explanation:

1. $Q^*(s,a)$: Optimal value of taking action a in state s .
2. $\max_{a'}$: Maximization over all possible actions a' in the next state s' .
3. $P(s'|s,a)$: Probability of transitioning to state s' from state s after taking action a .
4. $R(s,a,s')$: Reward received after transitioning from state s to state s' due to action a .
5. γ : Discount factor.
6. $Q^*(s',a')$: Optimal value of the next state-action pair (s',a') .

2. Principle of Optimality:

1. The Bellman Optimality Equation relies on the principle that the optimal policy leads to the maximum expected return.

3. Example:

1. Grid World:

1. Compute the optimal value for a cell by considering the best possible action and subsequent states.

Trade-off

1. In reinforcement learning, the exploration vs. exploitation trade-off refers to the dilemma of choosing between exploring new actions to discover their effects (exploration) and using known actions that yield high rewards (exploitation).

1. Exploration:

1. **Purpose:** To gather information about the environment.
2. **Benefits:**
 1. Discovers potentially better actions.
 2. Helps avoid local optima.
3. **Example:** Trying a new route in a navigation problem, even if the current route is known to be good.

2. Exploitation:

1. **Purpose:** To maximize immediate reward using known information.
2. **Benefits:**
 1. Utilizes accumulated knowledge.
 2. Increases short-term gains.
3. **Example:** Continuously choosing the highest-rewarding action in a slot machine game.

1.Importance of Balancing the Trade-off:

- 1. Short-Term vs. Long-Term Rewards:** Balancing immediate gains with the potential of discovering better long-term strategies.
- 2. Learning Efficiency:** Efficient learning requires a balance to avoid excessive exploration or premature convergence on suboptimal policies.
- 3. Adaptation:** The balance allows the agent to adapt to changing environments and improve performance over time.

2.Challenges:

- 1. Dynamic Environments:** Adapting the balance as the environment changes.
- 2. Complexity:** Finding the optimal balance can be computationally challenging.

Strategies: ϵ -Greedy, Softmax, UCB

1. ϵ -Greedy Strategy:

1. Definition:

1. With probability ϵ , choose a random action (exploration).
2. With probability $1-\epsilon$, choose the best-known action (exploitation).

2. Mathematical Formulation:

$$a = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg \max_a Q(s, a) & \text{with probability } 1 - \epsilon \end{cases}$$

3. Benefits:

1. Simple to implement.
2. Provides a straightforward balance between exploration and exploitation.

4. Drawbacks:

1. Fixed ϵ may not adapt well to different stages of learning.

5. Example: In a multi-armed bandit problem, occasionally trying a different lever.

1. Softmax Strategy:

1. Definition:

1. Chooses actions probabilistically based on their estimated value, using a softmax distribution.

2. Mathematical Formulation:

$$P(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}}$$

3. Explanation:

1. τ (temperature parameter) controls exploration level.
2. Higher τ increases exploration, lower τ increases exploitation.

4. Benefits:

1. Provides a smooth balance between exploration and exploitation.
2. Adapts to varying action values.

5. Drawbacks:

1. Computationally more intensive than ϵ -Greedy.

6. Example: In a recommendation system, probabilistically selecting items to suggest based on their scores.

1. Upper Confidence Bound (UCB) Strategy:

1. Definition:

1. Chooses actions based on both the estimated value and the uncertainty or variance in the estimate.

2. Mathematical Formulation:

$$a = \arg \max_a \left[Q(s, a) + c \sqrt{\frac{\ln t}{N(s, a)}} \right]$$

3. Explanation:

1. $Q(s, a)$: Estimated value of action a .
2. c : Confidence level parameter.
3. t : Total number of trials.
4. $N(s, a)$: Number of times action a has been chosen.

4. Benefits:

1. Balances exploration and exploitation by considering the confidence interval.
2. Encourages actions with high uncertainty to be explored.

5. Drawbacks:

1. Requires maintaining counts and is more complex to implement.

6. Example: In a multi-armed bandit problem, choosing levers with high potential rewards and high uncertainty.

1. Comparison:

1. **ϵ -Greedy**: Simple and easy to implement, but may not adapt well.
2. **Softmax**: Provides a smooth transition between exploration and exploitation.
3. **UCB**: Balances exploration with confidence, suitable for problems with known variance.

Model-Based vs. Model-Free RL

1. In reinforcement learning, approaches can be broadly categorized into model-based and model-free methods, each with distinct characteristics and use cases.

1. Model-Based RL:

1. Definition:

1. Model-based RL methods use a model of the environment to make decisions. This model predicts the next state and reward given a state and action.

2. Components:

1. **Transition Model** $P(s'|s,a)$: Predicts the probability of transitioning to state s' from state s after action a .
2. **Reward Model** $R(s,a)$: Predicts the reward received after taking action a in state s .

3. Process:

1. **Planning**: Use the model to simulate the environment and plan actions.
2. **Example**: Using a known map for a robot to plan a path before execution.

4. Benefits:

1. **Efficient Learning**: Can leverage the model to plan and improve policies rapidly.
2. **Sample Efficiency**: Requires fewer real-world interactions by simulating experiences.

5. Drawbacks:

1. **Model Accuracy**: Performance depends on the accuracy of the model.
2. **Complexity**: Building and maintaining a model can be complex and computationally intensive.

1. Model-Free RL:

1. Definition:

1. Model-free RL methods learn policies or value functions directly from interactions with the environment without explicitly modeling the environment.

2. Approaches:

1. **Policy-Based:** Learn a policy directly, e.g., Policy Gradient methods.
2. **Value-Based:** Learn value functions, e.g., Q-Learning, SARSA.

3. Process:

1. **Trial and Error:** Learn by interacting with the environment and updating policies or value functions based on received rewards.
2. **Example:** A robot learns to navigate by exploring and adjusting based on feedback without a map.

4. Benefits:

1. **Simplicity:** No need for a model, reducing complexity.
2. **Flexibility:** Can handle complex and unknown environments.

5. Drawbacks:

1. **Sample Inefficiency:** Often requires a large number of interactions with the environment.
2. **Slower Learning:** Learning purely from experience can be slower compared to model-based methods.

1.Comparison:

1. Planning vs. Learning:

1. Model-Based: Emphasizes planning using a model.
2. Model-Free: Focuses on learning from direct interaction.

2. Sample Efficiency:

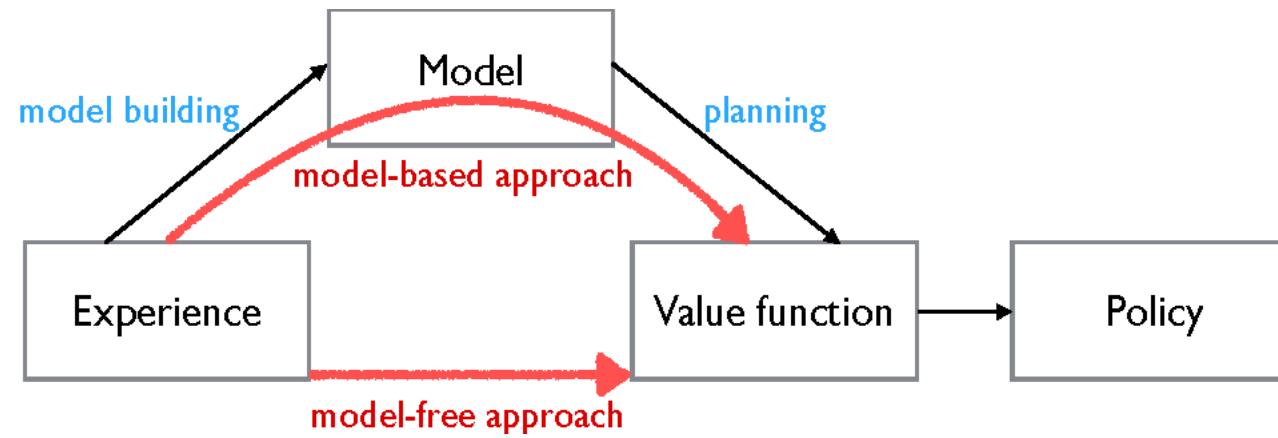
1. Model-Based: Typically more sample-efficient.
2. Model-Free: Generally less sample-efficient.

3. Computational Requirements:

1. Model-Based: Higher computational requirements for model maintenance and planning.
2. Model-Free: Lower computational requirements, focusing on policy or value updates.

2.Use Cases:

- 1. Model-Based:** Suitable for environments where a reliable model can be constructed, such as robotics with known dynamics.
- 2. Model-Free:** Preferred in complex or unknown environments where modeling is impractical, such as games or high-dimensional tasks.



Q-Learning

- Q-Learning is a model-free reinforcement learning algorithm that aims to find the best action to take given the current state.
- **Goal:** To learn a policy that tells an agent what action to take under what circumstances.
- **Key Concepts:**
 - **State (s):** The current situation or position in the environment.
 - **Action (a):** Possible moves or decisions the agent can make.
 - **Reward (r):** Feedback from the environment based on the action taken.
 - **Q-Value (Q(s, a)):** Represents the expected future rewards for an action taken in a given state.

- **The Q-Learning Algorithm:**

- 1. Initialize:** Q-values ($Q(s, a)$) arbitrarily.

- 2. For each episode:**

1. Initialize the starting state.

- 2. For each step in the episode:**

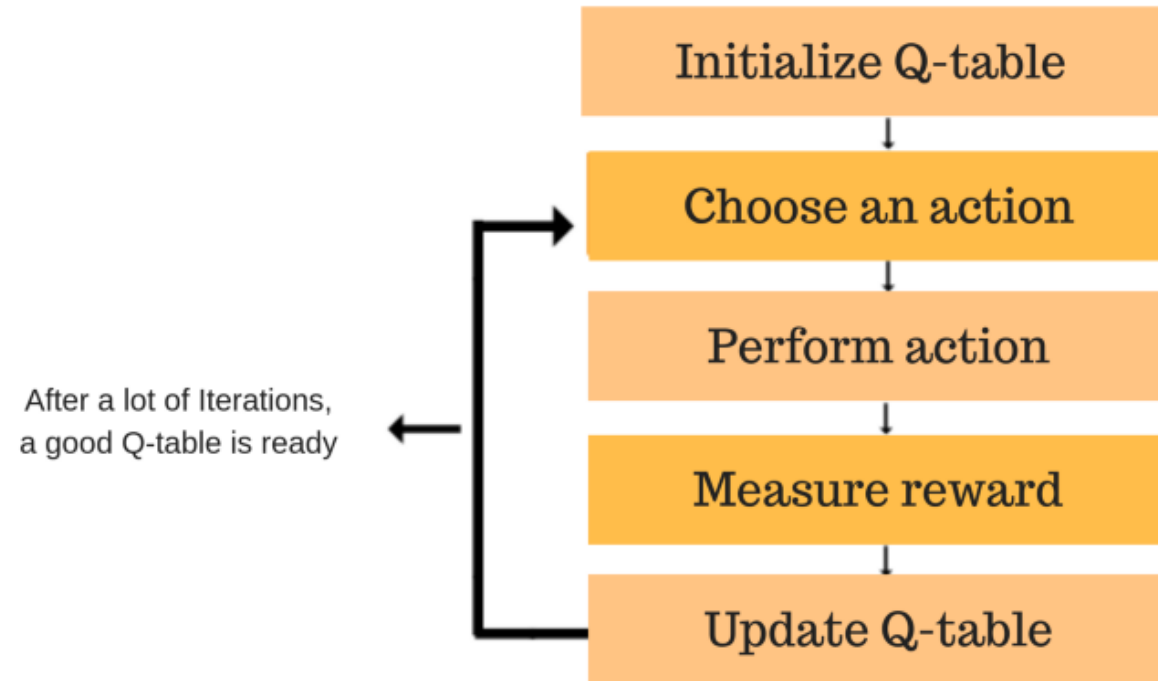
1. Choose an action (a) using an exploration strategy (e.g., ϵ -greedy).

2. Take the action, observe the reward (r) and next state (s').

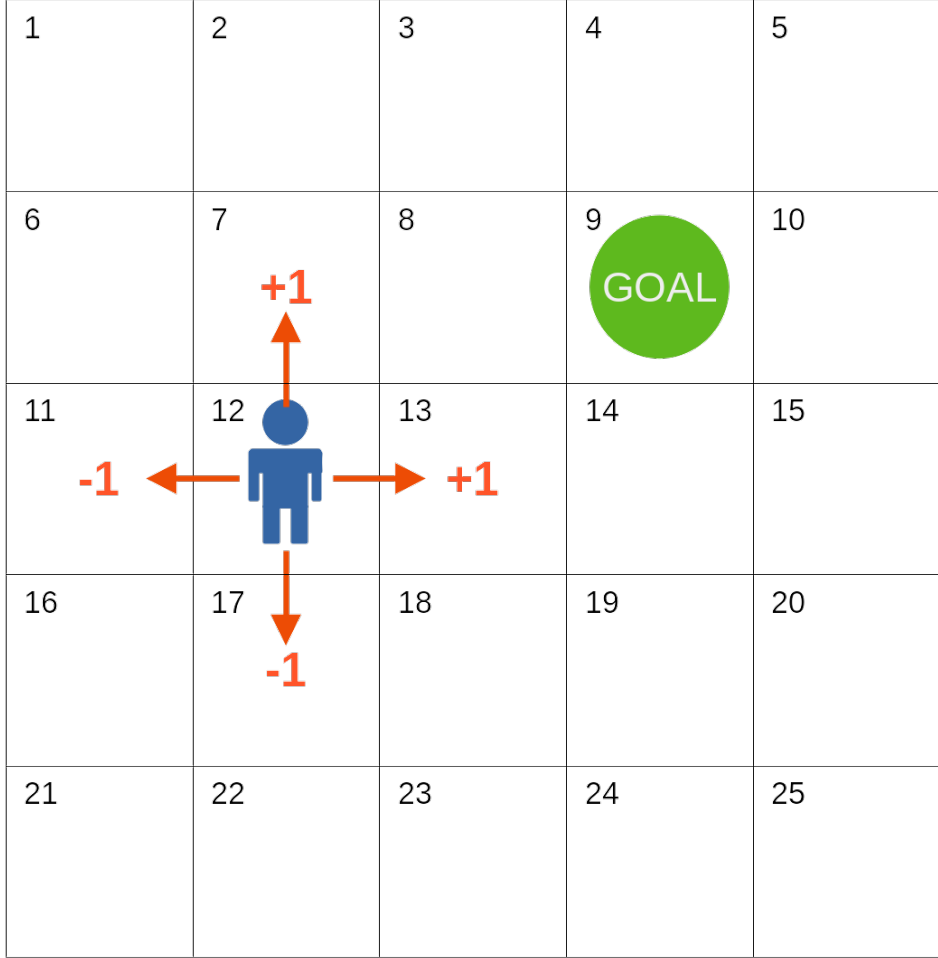
3. Update the Q-value using the Q-learning formula:





$$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

3. Repeat until the state is terminal.



1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



				
1	-	+1	-	+1
2	-	+1	-1	+1
3	-	+1	-1	+1
4	-	+1	-1	-1
5	-	+1	+1	-
...				
23	+1	-	-1	+1
24	+1	-	-1	-1
25	+1	-	+1	-