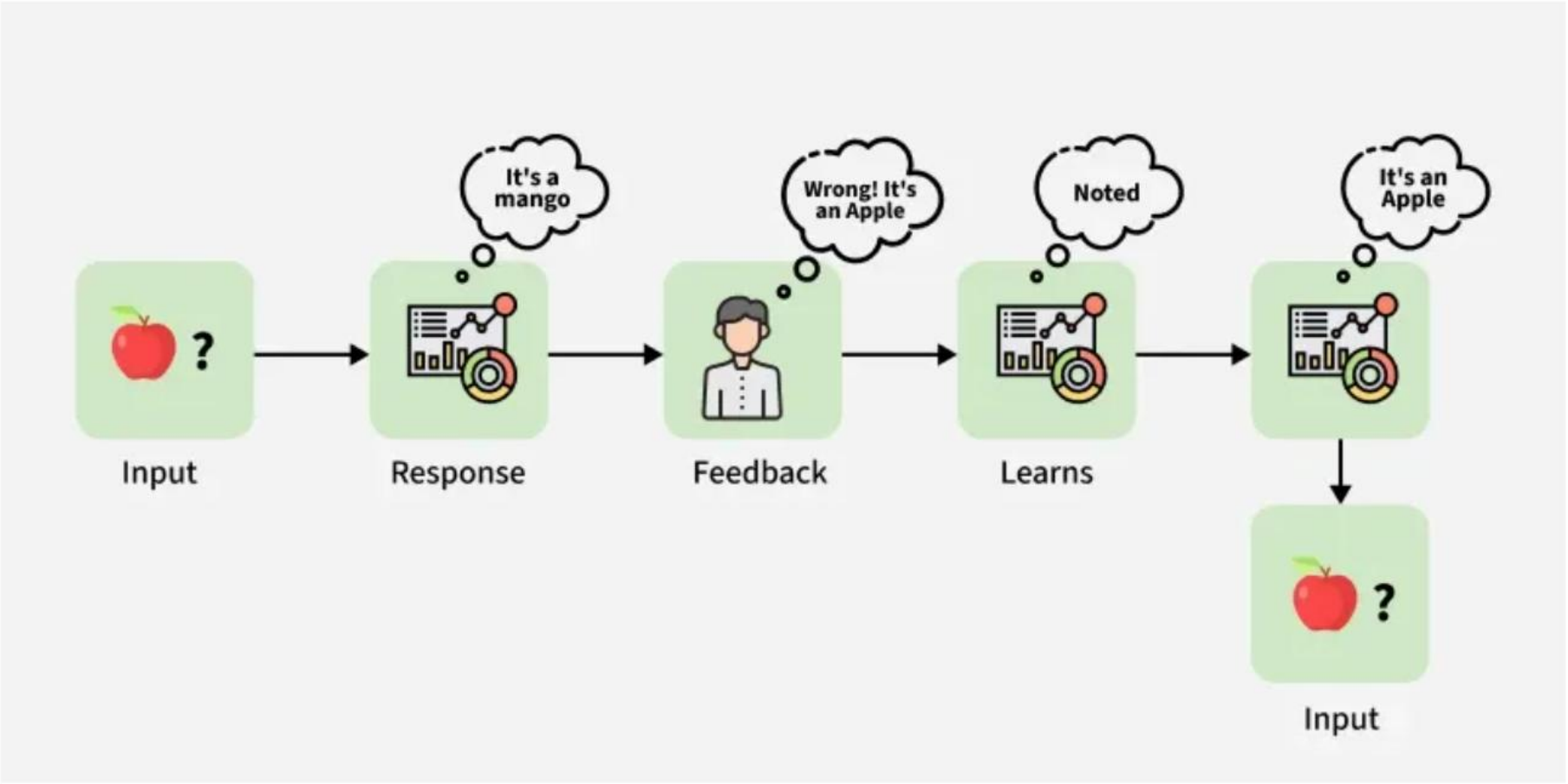
Q-Learning

Introduction

- Q-Learning is a popular model-free reinforcement learning algorithm that helps an agent learn how to make the best decisions by interacting with its environment.
- Instead of needing a model of the environment the agent learns purely from experience by trying different actions and seeing their results

- *Imagine a system that sees an apple but incorrectly says, "It's a mango." The system is told, "Wrong! It's an apple." It learns from this mistake. Next time, when shown the apple, it correctly says "It's an apple." This trial-and-error process, guided by feedback is like how Q-Learning works.*


It's a mango
Wrong! It's an Apple
Noted
It's an Apple
?
Input
Response
Feedback
Learns
?
Input

Q-Learning

Q-Learning in Reinforcement Learning

- The core idea is that the agent builds a Q-table which stores Q-values.
- Each Q-value estimates how good it is to take a specific action in a given state in terms of the expected future rewards.
- Over time the agent updates this table using the feedback it receives.

Key Components

- **Q-Values or Action-Values:** Q-values represent the expected rewards for taking an action in a specific state. These values are updated over time using the Temporal Difference (TD) update rule.

- **Rewards and Episodes:**The agent moves through different states by taking actions and receiving rewards. The process continues until the agent reaches a terminal state which ends the episode.

- **Temporal Difference or TD-Update :**The agent updates Q-values using the for $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$

- **S** is the current state and **S'** is the next state the agent moves to.
- **A** is the action taken by the agent and**A'** is the best next action in state S'.
- **R** is the reward received for taking action A in state S.
- **γ (Gamma)** is the discount factor which balances immediate rewards with future rewards.
- **α (Alpha)** is the learning rate determining how much new information affects the old Q-values.

- **ϵ-greedy Policy (Exploration vs. Exploitation)**

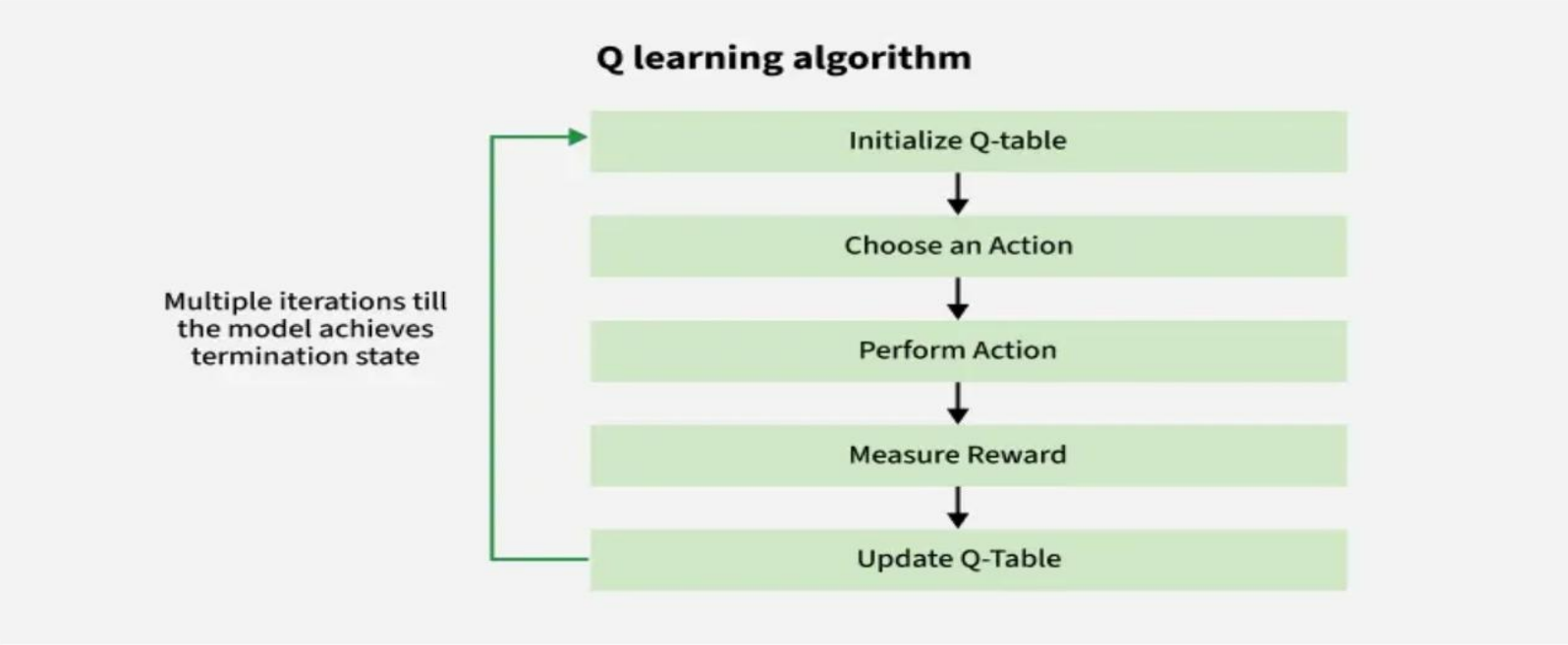
The ϵ-greedy policy helps the agent decide which action to take based on the current Q-value estimates:

- **Exploitation:** The agent picks the action with the highest Q-value with probability $1 - \epsilon$. This means the agent uses its current knowledge to maximize rewards.

- **Exploration:** With probability ϵ, the agent picks a random action, exploring new possibilities to learn if there are better ways to get rewards. This allows the agent to discover new strategies and improve its decision-making over time.

How does Q-Learning Works?

- Q-learning models follow an iterative process where different components work together to train the agent. Here's how it works step-by-step:



Q learning algorithm

❑ Start at a State (S)

- The environment provides the agent with a starting state which describes the current situation or condition.

❑ Agent Selects an Action (A)

- Based on the current state and the agent chooses an action using its policy. This decision is guided by a Q-table which estimates the potential rewards for different state-action pairs. The agent typically uses an ε-greedy strategy:
- It sometimes explores new actions (random choice).
- It mostly exploits known good actions (based on current Q-values).

- **Action is Executed and Environment Responds**

The agent performs the selected action. The environment then provides:

- A **new state (S′)** — the result of the action.
- A **reward (R)** — feedback on the action's effectiveness.

❑ 4. Learning Algorithm Updates the Q-Table

The agent updates the Q-table using the new experience:

- ❑ It adjusts the value for the state-action pair based on the received reward and the new state.
- ❑ This helps the agent better estimate which actions are more beneficial over time.

- **Policy is Refined and the Cycle Repeats**

With updated Q-values the agent:

- Improves its policy to make better future decisions.
- Continues this loop — observing states, taking actions, receiving rewards and updating Q-values across many episodes.

Over time the agent learns the optimal policy that consistently yields the highest possible reward in the environment.

Methods for Determining Q-values

- **Temporal Difference (TD):** Temporal Difference is calculated by comparing the current state and action values with the previous ones. It provides a way to learn directly from experience, without needing a model of the environment.

- **Bellman's Equation:**

Bellman's Equation is a recursive formula used to calculate the value of a given state and determine the optimal action.

It is fundamental in the context of Q-learning and is expressed as:

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

Where:

- **Q(s, a)** is the Q-value for a given state-action pair.
- **R(s, a)** is the immediate reward for taking action **a** in state **s**.
- **γ** is the discount factor, representing the importance of future rewards.
- $max_a Q(s', a)$ is the maximum Q-value for the next state **s'** and all possible actions.

What is a Q-table?

- The Q-table is essentially a memory structure where the agent stores information about which actions yield the best rewards in each state.
- It is a table of Q-values representing the agent's understanding of the environment.
- As the agent explores and learns from its interactions with the environment, it updates the Q-table.
- The Q-table helps the agent make informed decisions by showing which actions are likely to lead to better rewards.

Structure of a Q-table:

- Rows represent the states.
- Columns represent the possible actions.
- Each entry in the table corresponds to the Q-value for a state-action pair.
- Over time, as the agent learns and refines its Q-values through exploration and exploitation, the Q-table evolves to reflect the best actions for each state, leading to optimal decision-making.

Implementation

- Here, we implement basic Q-learning algorithm where agent learns the optimal action-selection strategy to reach a goal state in a grid-like environment.

- **Step 1: Define the Environment**

We first define the environment including the number of states, actions, goal state and the Q-table. Each state represents a position in a 4×4 grid and actions represent movements in four directions.

➢ Step 2: Set Hyperparameters

These parameters control the learning process:

- ❑ **learning_rate (α):** How much new info overrides old info.
- ❑ **discount_factor (γ):** How much future rewards are valued.
- ❑ **exploration_prob (ε):** Probability of taking a random action.
- ❑ **epochs:** Number of training episodes.

➢ Step 3: Define the State Transition Function

This function calculates the next state based on the current state and chosen action.
We assume:

- ❑ 0 → Left
- ❑ 1 → Right
- ❑ 2 → Up
- ❑ 3 → Down

- **Step 4: Implement the Q-Learning Algorithm**
- Now we train the agent to learn the optimal policy. At each step:
- Choose an action using epsilon-greedy policy.
- Move to the next state based on that action.
- Update the Q-value using the Bellman equation.
- End the episode if the goal is reached.

- **Step 5: Output the Learned Q-Table**

After training, we visualize the maximum Q-value of each state on a grid and print the learned Q-table. States closer to the goal should have higher Q-values, showing the agent's learned path.

```
Learned Q-table:
[[12.42059108 13.12828402  9.64649941  8.38685619]
 [12.4058649  13.70673929 12.40562452 13.81924681]
 [11.5182582  14.5457676   9.82738645 14.25394111]
 [13.61580358 12.78553816 14.32320837 15.3121845 ]
 [11.74093567  8.06524516  3.44580865 13.81376551]
 [12.28626772 12.91524409 13.09937537 14.67923529]
 [ 8.82651103 15.31206808 13.10192887 14.87638963]
 [14.02849773 15.23929949 14.38487249 16.21016289]
 [13.36775052 14.65052003 12.31653404 13.79142056]
 [13.91633622 14.62101887 13.83538555 15.50782123]
 [13.86577562 16.33806019 14.29643719 13.45597691]
 [15.29003162 16.2028466  15.22371997 17.20405657]
 [14.03732145 15.42069045 12.79412622 14.11852529]
 [14.60128903 16.34328743 14.66744352 15.39940312]
 [15.27700317 17.20420599 15.3053891  16.17748193]
 [14.9542415  17.05709817  8.78760795 13.84864754]]
```

Learned Q table

- Each row index (0–15) represents a state in the 4×4 grid.
- Each column value shows the Q-value for the actions [Left, Right, Up, Down].
- Higher Q-values indicate better actions that move the agent closer to the goal.

- For example, In the last few rows (states 12–15), the Q-values are noticeably higher (around 15–17). This means the agent has learned that these states are near the goal state (15) and should move toward it.
- The top rows (states 0–3) have lower Q-values (around 8–13), showing that those positions are far from the goal and require more steps to reach it.
- The agent's policy i.e. best action from each state would be the action corresponding to the maximum Q-value in that row.

Example Illustration

State	Q-values (A_1, A_2, A_3)	Best Action (Strategy)
S_1	(3.2, 1.8, 0.5)	A_1
S_2	(0.7, 4.0, 2.3)	A_2
S_3	(2.5, 2.1, 3.8)	A_3

So, the **strategy/policy** derived from this Q-table is:

$$\pi(S_1) = A_1,\ \pi(S_2) = A_2,\ \pi(S_3) = A_3$$

Advantages

- **Trial and Error Learning**: Q-learning improves over time by trying different actions and learning from experience.
- **Self-Improvement**: Mistakes lead to learning, helping the agent avoid repeating them.
- **Better Decision-Making**: Stores successful actions to avoid bad choices in future situations.
- **Autonomous Learning**: It learns without external supervision, purely through exploration.

Disadvantages

- **Slow Learning**: Requires many examples, making it time-consuming for complex problems.
- **Expensive in Some Environments**: In robotics, testing actions can be costly due to physical limitations.
- **Curse of Dimensionality**: Large state and action spaces make the Q-table too large to handle efficiently.
- **Limited to Discrete Actions**: It struggles with continuous actions like adjusting speed, making it less suitable for real-world applications involving continuous decisions.