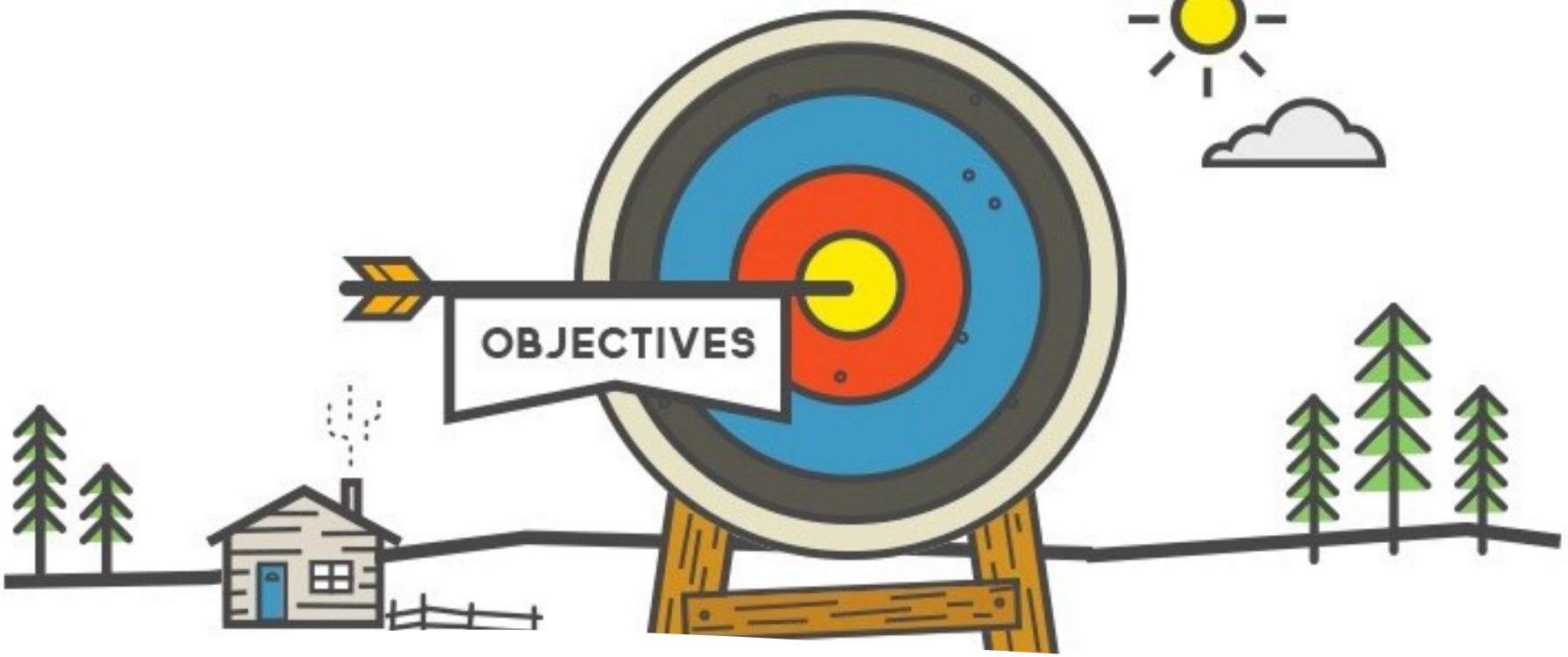
Lisbon School of Economics & Management
Universidade de Lisboa

Reinforcement Learning

Carlos J. Costa (2024)



Learning Goals

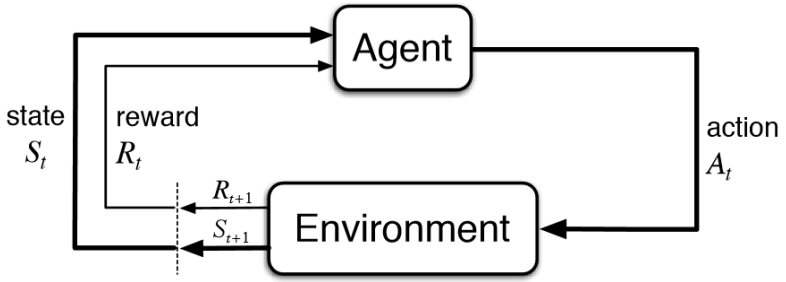
- Understand main Concepts of Reinforcement Learning
- Understand how to implement

Summary

- Reinforcement learning
- Key components
- Example

Reinforcement learning

- is a type of machine learning
- where
  - an **agent learns to make decisions** by interacting with an environment,
  - receiving **rewards**, and
  - improving its behavior over time.



Key components

- Agent - The decision-maker (e.g., a trading bot).
- Environment - The system the agent interacts with (e.g., the stock market).
- State - The current situation of the agent (e.g., time or price point).
- Action - A choice the agent makes (e.g., Buy, Sell, Hold).
- Reward - Feedback received after an action (e.g., profit/loss).
- Policy - A strategy that maps states to actions.

Environment

- The system the agent interacts with
- The stock prices (the world):
  - The robot sees stock prices moving up and down over 100 days.
  - Each "state" is just a day — Day 0, Day 1, ..., Day 99.

Action

- A choice the agent makes
- The robot can make 3 moves (actions):
  - Buy — decide to buy the stock.
  - Sell — decide to sell it.
  - Hold — do nothing.

Reward (score)

- If the robot buys and the price goes up → good job → reward!
- If it buys and the price drops → bad job → penalty.
- Same idea for selling.
- Holding gets zero reward.

What is Q-learning

- Q-learning is an algorithm
- Used in **reinforcement learning**
- Find the **best action to take in each state**
- Maximize **future rewards**

- It works by learning a table of values called a **Q-table**

What is Q-learning

- Each entry in the table is Q(state, action)
- It estimates how good it is to take that action when it is in that state.
- The goal is to learn the best action in every state
- **interacting with the environment** and **updating the Q-values** over time

What is Q-learning

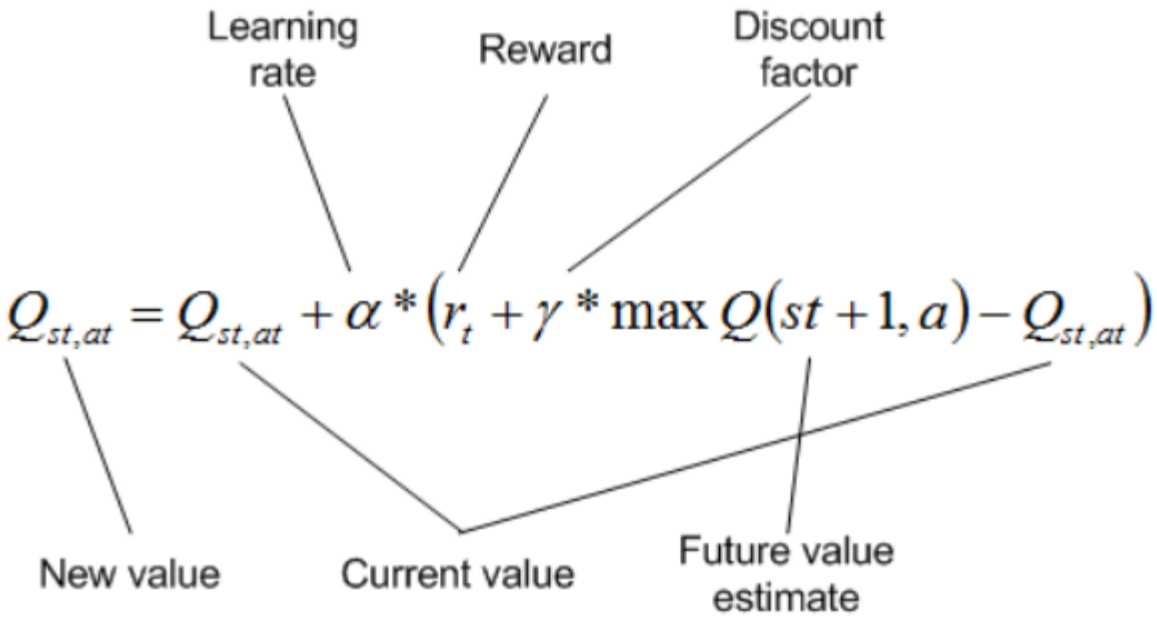
- Q-learning is an algorithm used in **reinforcement learning** to find the **best action to take in each state** to maximize **future rewards**.
- It works by learning a table of values called a Q-table, where:
  - Q stands for "Quality" of a certain action in a certain state.
  - Each entry in the table is Q(state, action) — it estimates how good it is to take that action when you're in that state.

What is Q-learning

The goal is to learn the best action in every state by **interacting with the environment** and **updating the Q-values** over time using this formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \cdot \max' Q(s',a') - Q(s,a)] Q(s, a)$$

Q-learning formula



What is a **strategy** in Q-learning

- The strategy, also called the policy, is the rule the agent uses to decide which action to take in each state.
- In Q-learning, the strategy is:
  - For any given state, pick the action that has the highest Q-value.
  - This is how it's done in code:

    ```
    action = actions[np.argmax(q_table.iloc[state])]
    ```

  - That line says: “Look at the current state, find the row in the Q-table, and pick the action with the highest number.”

How the code uses Q-learning and learns the strategy

- Initialize the Q-table
- Pick an action
- Get the reward
- Update the Q-table (learn)
- Repeat for many episodes
- After training, the strategy is: For any given state, choose the action with the highest Q-value.

Example Python (1/3)

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Generate synthetic stock price data
np.random.seed(0)
dates = pd.date_range(start='2024-01-01', periods=100)
prices = np.cumsum(np.random.randn(100)) + 100  # Synthetic stock prices
data = pd.DataFrame({'Date': dates, 'Price': prices})

# Q-Learning parameters
learning_rate = 0.1
discount_factor = 0.9
exploration_rate = 1.0
max_exploration_rate = 1.0
min_exploration_rate = 0.01
exploration_decay_rate = 0.01

# Initialize Q-Table
states = np.arange(0, len(prices))
actions = ['Buy', 'Sell', 'Hold']
q_table = pd.DataFrame(np.zeros((len(states), len(actions))), columns=actions)
```

Example Python (2/3)

```python
# Training the agent
rewards = []
for episode in range(1000):
    state = np.random.randint(0, len(states))
    done = False
    total_reward = 0

    while not done:
        # Exploration-exploitation trade-off
        if np.random.uniform(0, 1) < exploration_rate:
            action = np.random.choice(actions)
        else:
            action = actions[np.argmax(q_table.iloc[state])]

        # Simulate the next state and reward
        next_state = state + 1 if state < len(states) - 1 else state
        if action == 'Buy':
            reward = prices[next_state] - prices[state]
        elif action == 'Sell':
            reward = prices[state] - prices[next_state]
        else:
            reward = 0

        # Update Q-Table
        q_table.loc[state, action] = (1 - learning_rate) * q_table.loc[state, action] + \
                                     learning_rate * (reward + discount_factor * np.max(q_table.iloc[next_state]))

        state = next_state
        total_reward += reward

        if state == len(states) - 1:
            done = True

    # Decay exploration rate
    exploration_rate = min_exploration_rate + \
                       (max_exploration_rate - min_exploration_rate) * np.exp(-exploration_decay_rate * episode)

    rewards.append(total_reward)
```

Example Python (3/3)

```
# Plot rewards
plt.plot(rewards)
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.title('Total Reward per Episode')
plt.show()

# Display Q-Table
q_table
```

Conclusion

- Concept
- Key components
- Example