JC3504 Robot Technology

Lecture 13: Reinforcement Learning

Dr Xiao Li xiao.li@abdn.ac.uk

Dr Junfeng Gao junfeng.gao@abdn.ac.uk

An Example of Reinforcement Learning

<Video : 13 - Multi-Agent Hide and Seek.mp4>

Outline

Basic Concepts of Reinforcement Learning

Markov Decision Process (MSP)

Bellman Equation

Basic Concepts of Reinforcement Learning

UNIVERSITY OF ABERDEEN

Reinforcement Learning

Reinforcement learning is a branch of machine learning that enables algorithms (or "agents") to learn how to achieve goals through trial and error in an environment.

In reinforcement learning, an agent learns by interacting with its environment: it takes actions in a specific state and is rewarded or punished depending on the results of its actions.

The agent's goal is to learn how to choose actions that maximise the sum of rewards it obtains in the long run.

agent

Reinforcement Learning

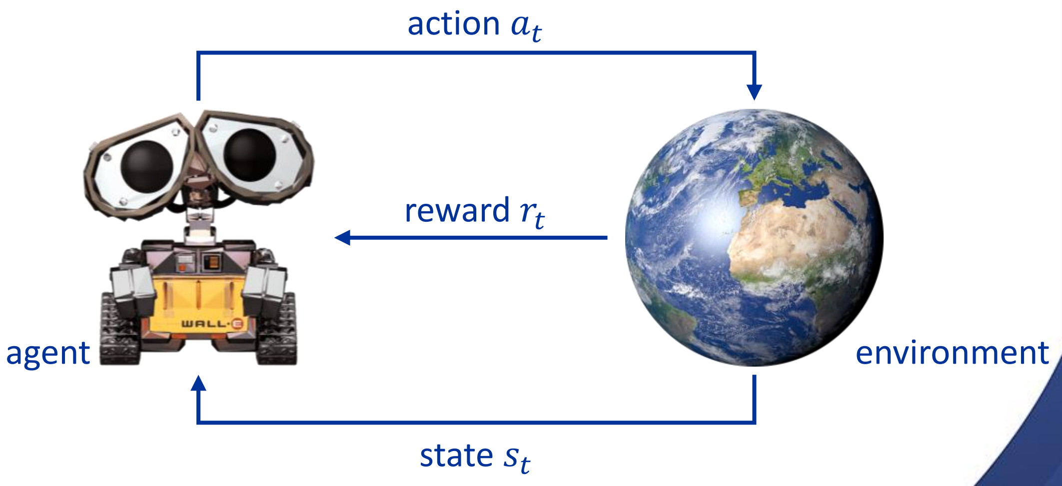
At each step t the agent:

- Executes action a_t
- Receives state/observation s_t
- Receives scalar reward r_t

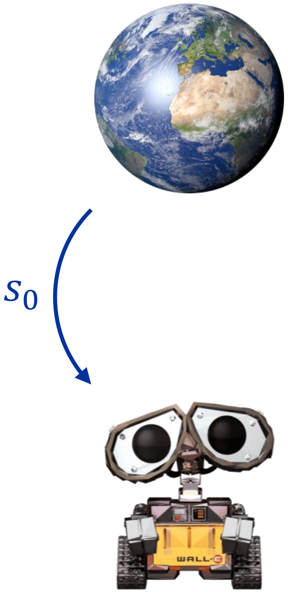
The environment:

- Receives action a_t
- Emits observation o_{t+1}
- Emits scalar reward r_{t+1}

t increments at env. step



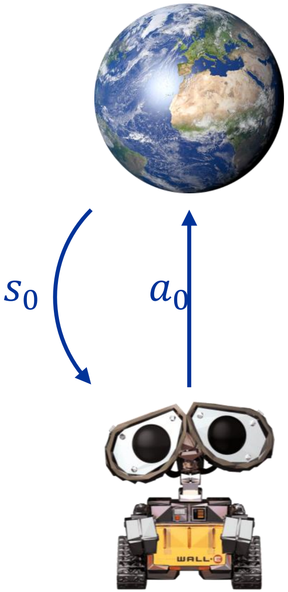
Reinforcement Learning



The agent observes a state which may be noisy or incomplete. This implies that the information acquired could contain errors or may not fully represent the environment.

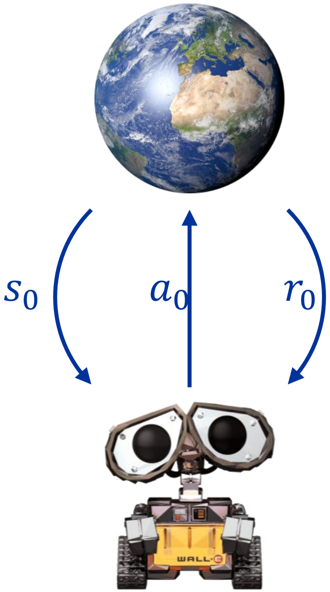
1495 UNIVERSITY OF ABERDEEN

Reinforcement Learning



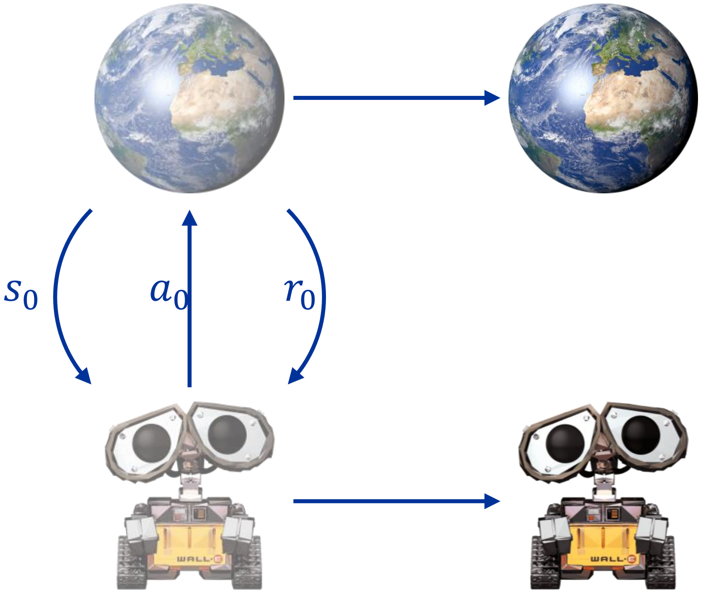
The agent takes an action based upon its observations.
It decides its course of action by interpreting the current state of the environment.

Reinforcement Learning



The reward indicates to the agent the effectiveness of its actions. It provides a measure of success, guiding the agent on its performance.

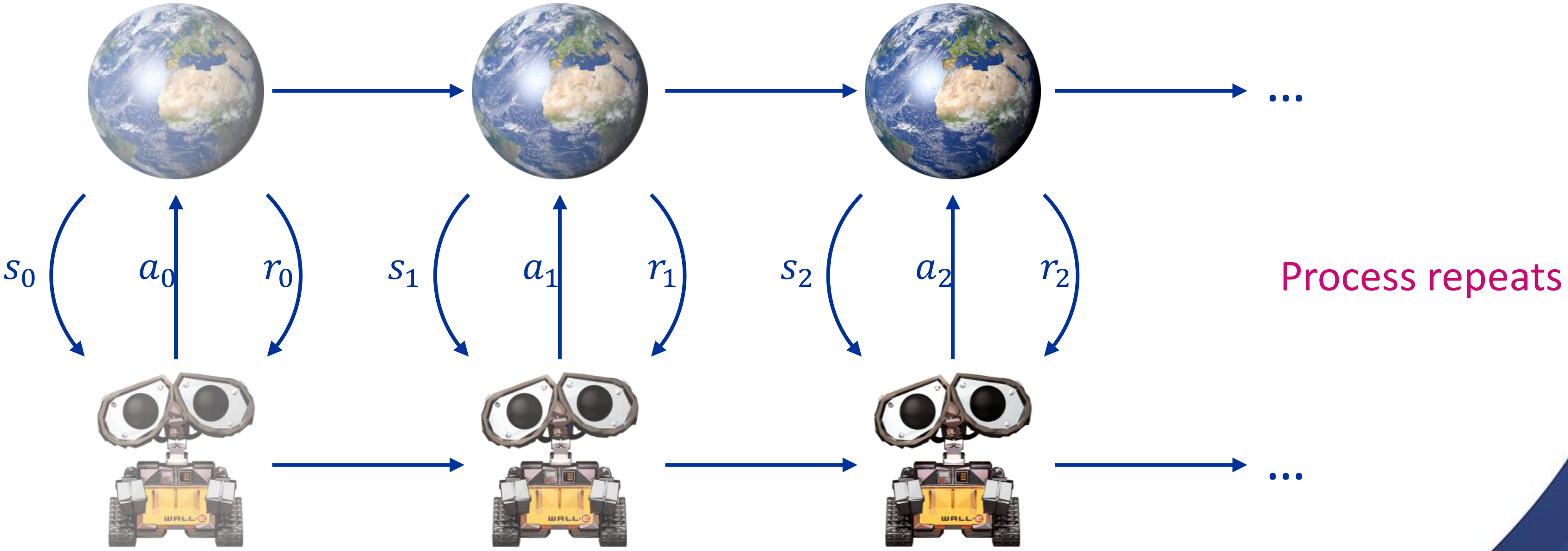
Reinforcement Learning



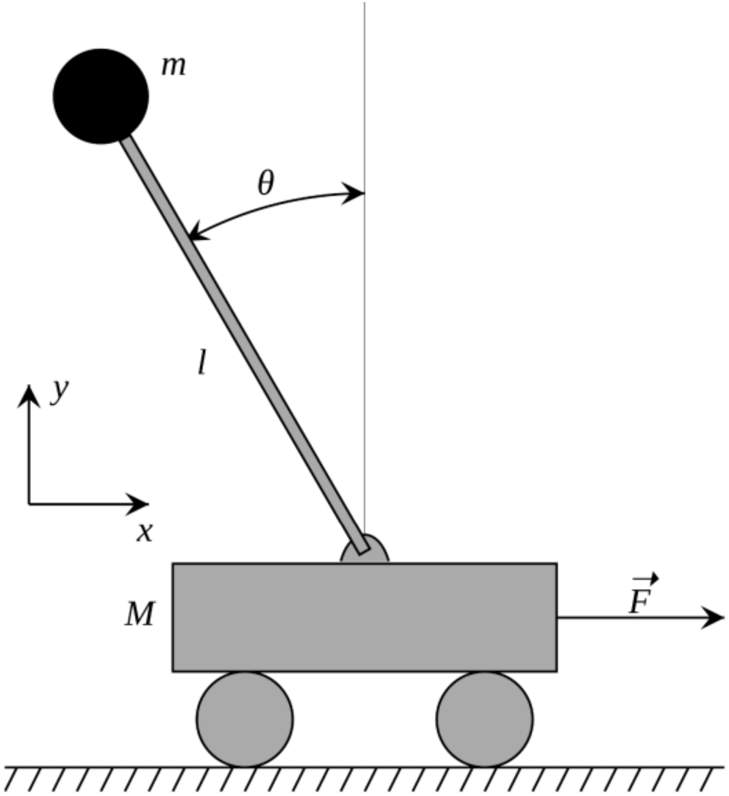
Action causes change to environment

Agent learns

Reinforcement Learning



Example: Cart-Pole Problem



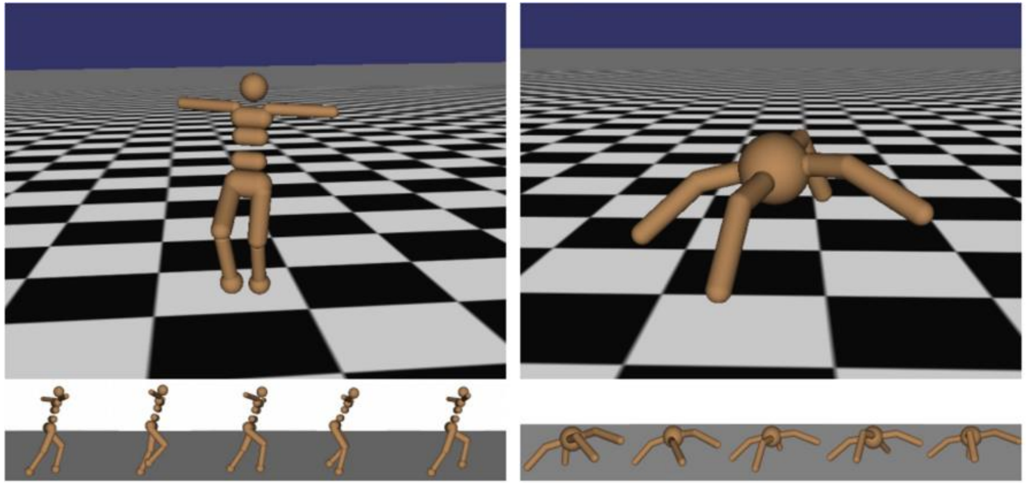
Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity

Action: horizontal force applied on the cart

Reward: *1* at each time step if the pole is upright

Example: Robot Locomotion



Objective: Make the robot move forward

State: Angle, position, velocity of all joints

Action: Torques applied to joints

Reward: *1* at each time step upright + forward movement

UNIVERSITY OF ABERDEEN

Example: Atari Games

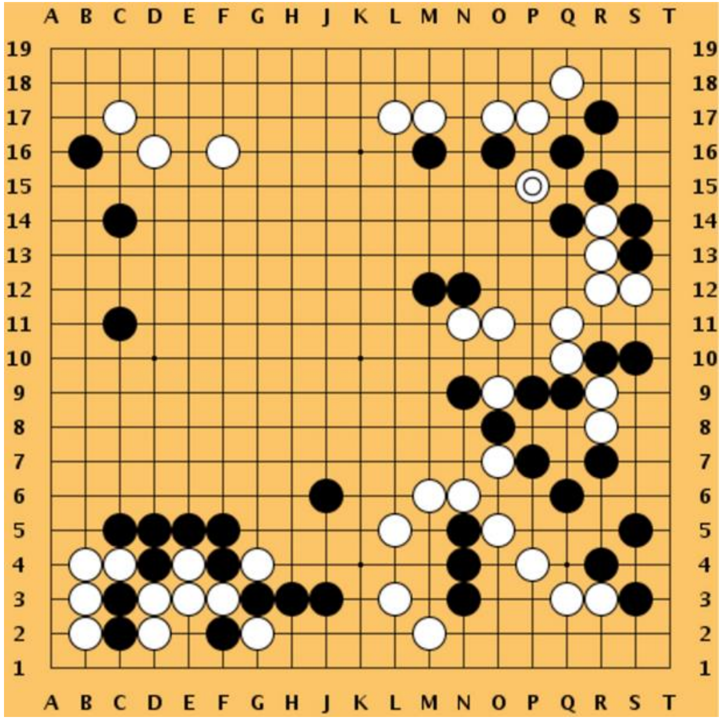
Objective: Complete the game with the highest score

State: ?

Action: ?

Reward: ?

Example: Go



Objective: ?

State: ?

Action: ?

Reward: ?

UNIVERSITY OF ABERDEEN

Reinforcement Learning vs Supervised Learning

In reinforcement learning, the agent must navigate a stochastic environment and learn from delayed rewards without clear guidance on which actions are best.

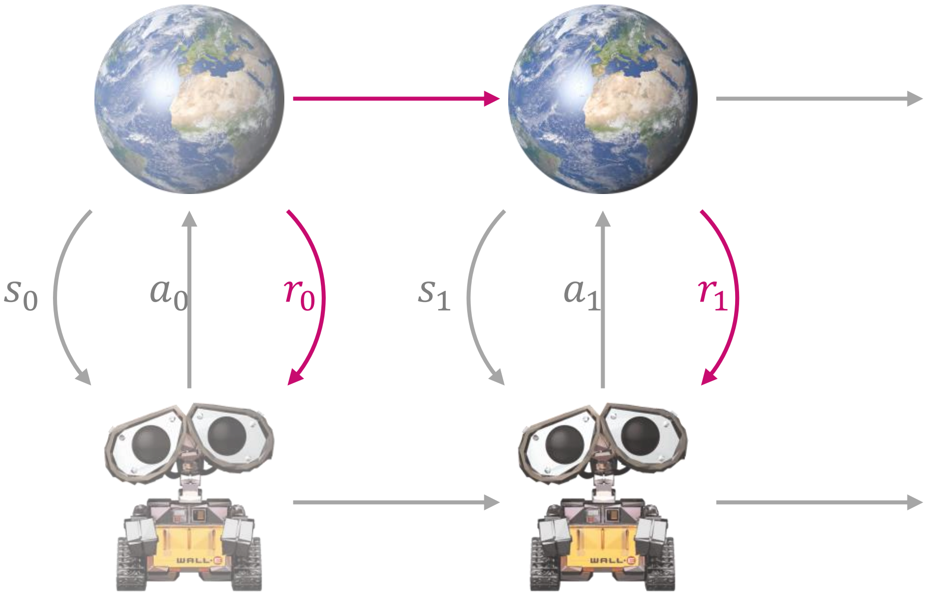
Conversely, in supervised learning, the model is trained on a fixed dataset with explicit examples, optimising for prediction accuracy through gradient descent.

The main differences lie in

- Stochasticity
- Credit assignment
- Non-differentiability
- Non-stationarity

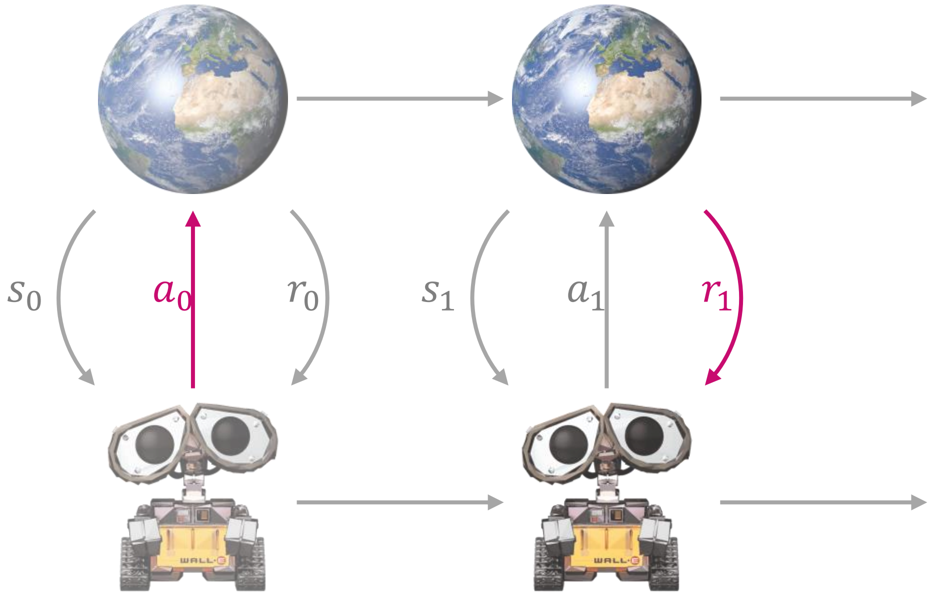
Reinforcement Learning vs Supervised Learning

Stochasticity: Rewards and state transitions may be random



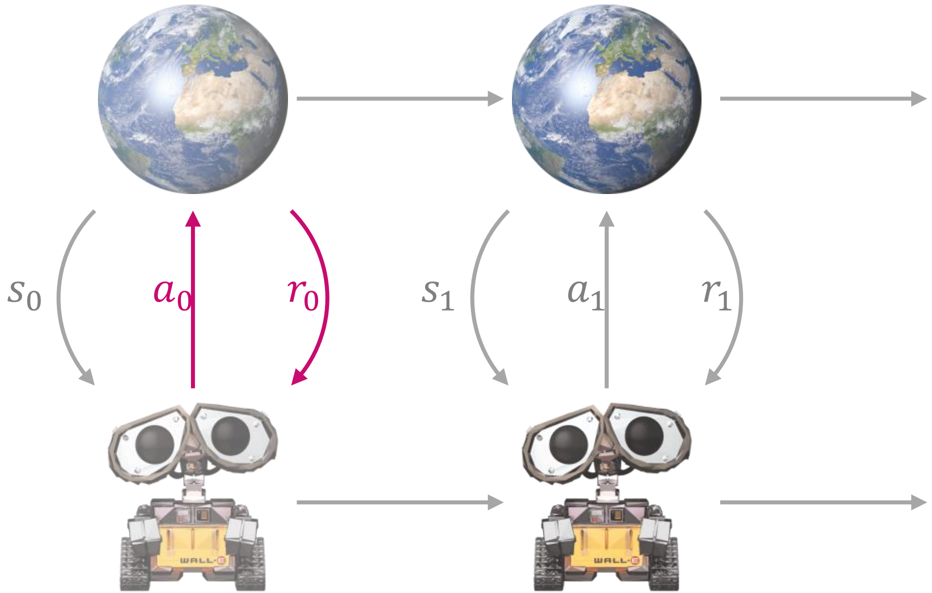
Reinforcement Learning vs Supervised Learning

Credit assignment: Reward r_t may not directly depend on action a_t



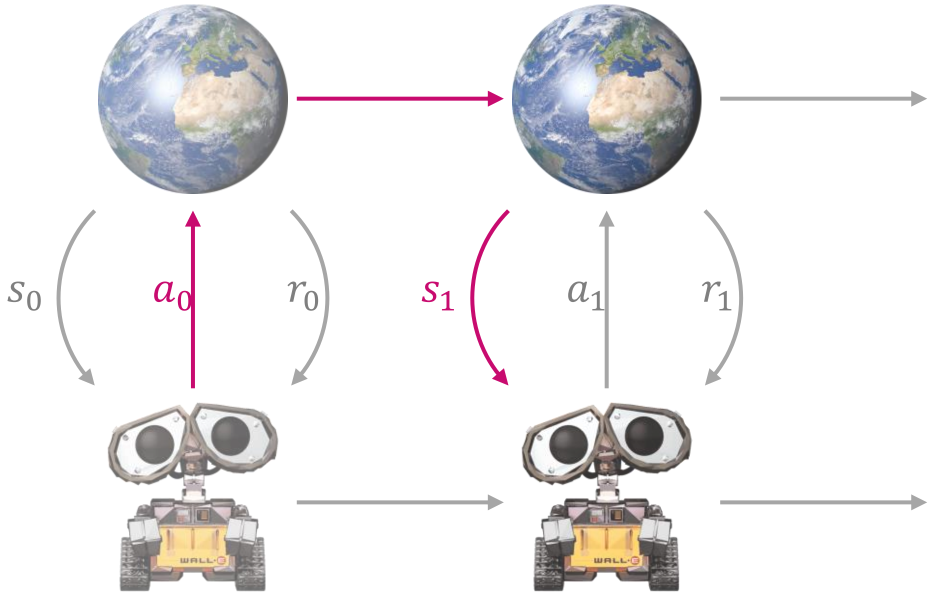
Reinforcement Learning vs Supervised Learning

Non-differentiable: Cannot backpropagate through world i.e. cannot compute $\frac{dr_t}{da_t}$



Reinforcement Learning vs Supervised Learning

Nonstationary: What the agent experiences depends on how it actions



Markov Decision Process (MSP)

Markov Decision Process (MDP)

MDP is a mathematical formalisation of the reinforcement learning problem, characterised by a tuple (S, A, R, P, γ)

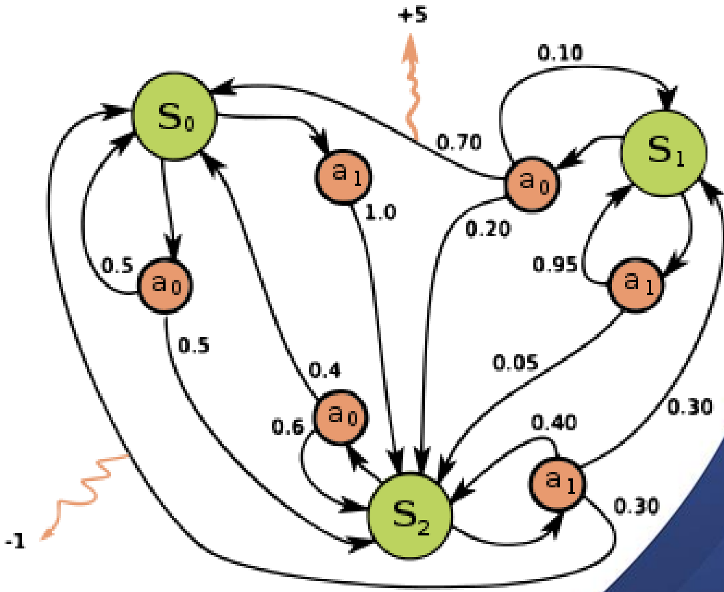
S : Set of possible states (We assume a state completely characterises a state of the world)

A : Set of possible actions

R : Distribution of reward given (state, action) pair

P : A transition probability matrix offering the distribution over subsequent states given the current state and action.

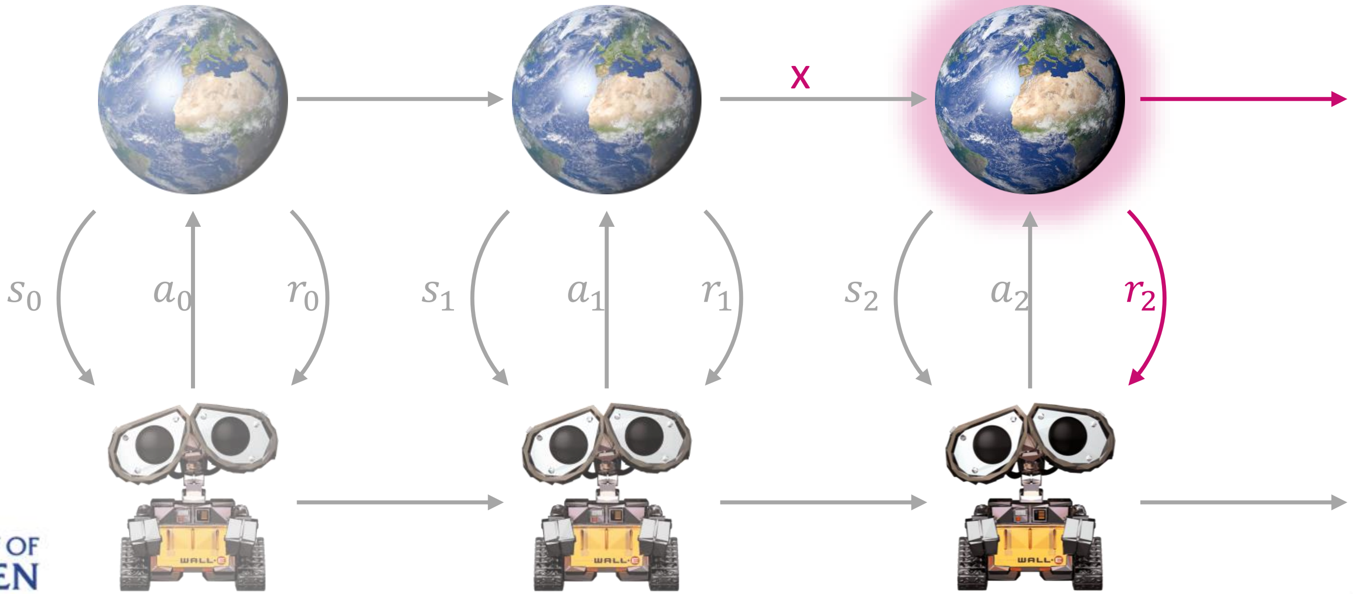
γ : Discount factor (trade-off between future and present rewards)



UNIVERSITY OF ABERDEEN

Markov Hypothesis in MDF

Markov hypothesis: Rewards and next states depend only on current state, not history.

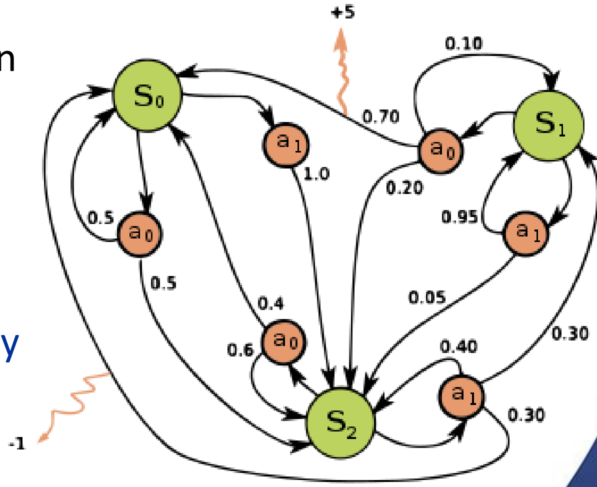


Policy of MDP

In MDP, policy π refers to the mapping from states to actions. Specifically, it is a decision-making policy that defines the actions an agent should take when given a state. The policy π can be deterministic or stochastic:

- Deterministic policy: In this case, for each state s, the policy π specifies an action a, that is, $\pi(s) = a$
- Randomness policy: For each state s, π will provide a probability distribution of actions, that is, for all possible actions a, $\pi(a|s)$ represents the probability of choosing action a in state s.

The goal of RL is to find policy $\hat{\pi}$ that maximises cumulative discounted reward: $\sum_t \gamma^t r_t$



UNIVERSITY OF ABERDEEN

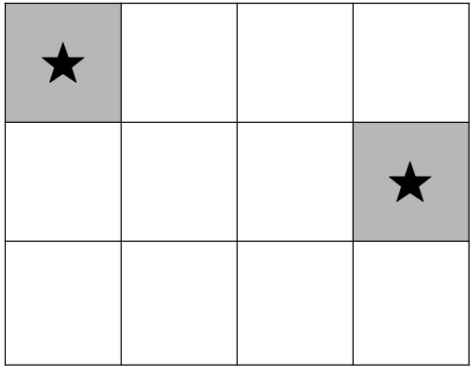
Markov Decision Process Example

Objective: Reach one of the terminal (★) states in as few moves as possible

Actions:

1. Right
2. Left
3. Up
4. Down

States:



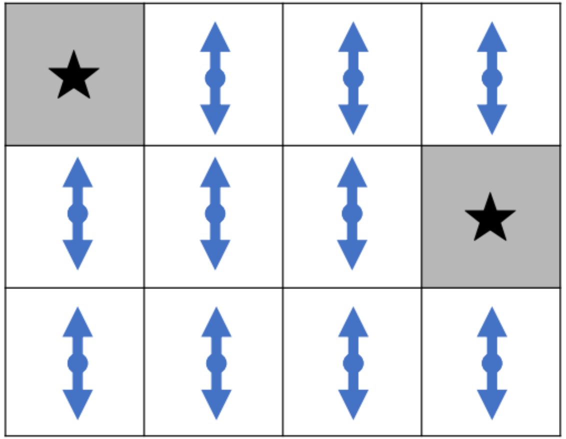
Rewards:

?

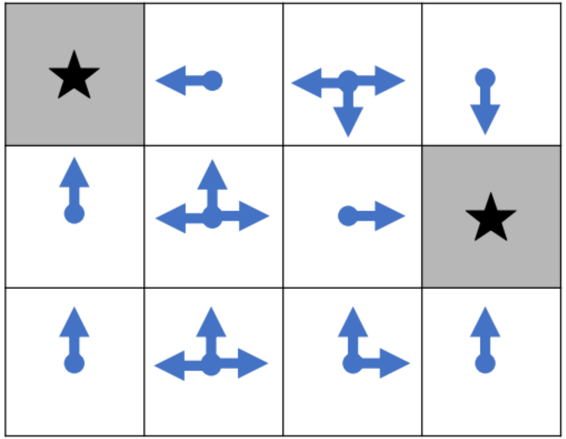
Markov Decision Process Example

Policy $\hat{\pi}$:

Bad policy



Optimal Policy



UNIVERSITY OF ABERDEEN

Finding Optimal Policies

Goal: Find the optimal policy $\hat{\pi}$ that maximises sum of rewards.

Problem: Lots of randomness! Initial state, transition probabilities, rewards

Solution: Maximize the expected sum of rewards

$$\hat{\pi} = arg \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi\right], \qquad \begin{gathered} s_0 \sim p(s_0) \\ a_t \sim \pi(a|s_t) \\ s_{t+1} \sim P(s|s_t, a_t) \end{gathered}$$

Bellman Equation

Value Function

Following a specific policy π, we generate a sequence of sample trajectories (or paths), which comprise a series of states, actions, and rewards: $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

The value function $V^\pi(s)$ at a given state s represents the expected cumulative discounted reward when starting from state s and adhering to policy π. Mathematically, this is expressed as:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, \pi\right]$$

Value function represents how good the state s is.

Q Function

Following a specific policy π, we generate a sequence of sample trajectories (or paths), which comprise a series of states, actions, and rewards: $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

Q Function, also known as the action-value function, is used in reinforcement learning to assess the quality of state-action pairs.

The Q Function, $Q^{\pi}(s, a)$ for a particular state s and action a, represents the expected cumulative discounted reward when one commits to action a in state s and thereafter adheres to policy π. It is defined as follows:

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

UNIVERSITY OF ABERDEEN

Bellman Equation

The Bellman Equation is a fundamental concept in reinforcement learning and dynamic programming, introduced by Richard Bellman. It provides a recursive solution for determining the **optimal policy**.

In reinforcement learning, the Bellman Equation describes the value function of a state (or the action-value function) which represents the maximum expected return achievable by following an optimal policy from that state. The Bellman Equation takes advantage of the Markov Decision Process property, asserting that the value of the current state is the immediate reward plus the value of the subsequent state, discounted by a factor.

Bellman Equation

Optimal Q-function: $\hat{Q}(s,a)$ is the Q-function for the optimal policy $\hat{\pi}$ It gives the max possible future reward when taking action a in state s:

$$\hat{Q}(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

$\hat{Q}$ encodes the optimal policy:

$$\hat{\pi}(s) = arg \max_{a\prime} Q(s,a')$$

UNIVERSITY OF ABERDEEN

Bellman Equation

Optimal Q-function: $\hat{Q}(s,a)$ is the Q-function for the optimal policy $\hat{\pi}$ It gives the max possible future reward when taking action a in state s:

$$\hat{Q}(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

$\hat{Q}$ encodes the optimal policy:

$$\hat{\pi}(s) = arg\max_{a'} Q(s,a')$$

Bellman Equation: $\hat{Q}$ satisfies the following recurrence relation:

$$\hat{Q}(s,a) = \mathbb{E}_{\mathrm{r,s'}}\left[r + \gamma \max_{a'} \hat{Q}(s',a')\right], r{\sim}R(s,a), s'{\sim}P(s,a)$$

Intuition: After taking action a in state s, we get reward r and move to a new state s'. After that, the max possible reward we can get is $\max_{a'} \hat{Q}(s',a')$

Solving for the Optimal Policy: Value Iteration

Bellman Equation: $\hat{Q}$ satisfies the following recurrence relation:

$$\hat{Q}(s,a) = \mathbb{E}_{\mathrm{r,s'}}\left[r + \gamma \max_{a'} \hat{Q}(s',a')\right], r \sim R(s,a), s' \sim P(s,a)$$

Idea: If we find a function $Q(s,a)$ that satisfies the Bellman Equation, then it must be $\hat{Q}$.
Start with a random Q, and use the Bellman Equation as an update rule:

$$Q_{i+1}(s,a) = \mathbb{E}_{\mathrm{r,s'}}\left[r + \gamma \max_{a'} Q_i(s',a')\right], r \sim R(s,a), s' \sim P(s,a)$$

Amazing fact: Q_i converges to $\hat{Q}$ as $i \rightarrow \infty$

Example: to Find $\hat{Q}$

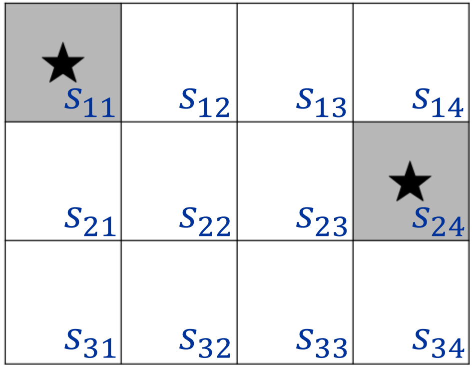
Reconsider the example.

Objective: Reach one of the terminal (★) states in as few moves as possible

Actions:

1. Right $a_{\uparrow}$
2. Left $a_{\leftarrow}$
3. Up $a_{\downarrow}$
4. Down $a_{\rightarrow}$

States:



Q_0:

$$Q(s_{12}, a_{\uparrow}) = 0$$
$$Q(s_{12}, a_{\leftarrow}) = 0$$
$$Q(s_{12}, a_{\downarrow}) = 0$$
$$Q(s_{12}, a_{\rightarrow}) = 0$$
$$Q(s_{13}, a_{\uparrow}) = 0$$
$$Q(s_{13}, a_{\leftarrow}) = 0$$
$$Q(s_{13}, a_{\downarrow}) = 0$$
$$Q(s_{13}, a_{\rightarrow}) = 0$$

…

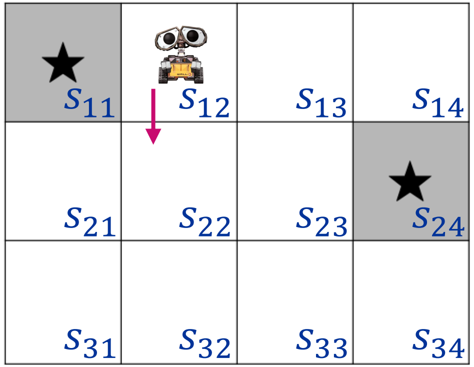
We use a large value to represent $Q(\bigstar, \cdot)$ e.g. 10.

Example: to Find $\hat{Q}$

Suppose the robot is in s_{12}, and takes action $a_{\downarrow}$,

$$Q(s_{12}, a_{\downarrow}) = r_{s_{12},a_{\downarrow}} + \gamma \max\big(Q(s_{22}, a_{\uparrow}), Q(s_{22}, a_{\leftarrow}), Q(s_{22}, a_{\downarrow}), Q(s_{22}, a_{\rightarrow})\big)$$

$$Q(s_{12}, a_{\downarrow}) = -1 + \gamma \max(0,0,0,0) = -1$$



$$Q(s_{12}, a_{\uparrow}) = 0$$
$$Q(s_{12}, a_{\leftarrow}) = 0$$
$$Q(s_{12}, a_{\downarrow}) = 0$$
$$Q(s_{12}, a_{\rightarrow}) = 0$$
$$Q(s_{13}, a_{\uparrow}) = 0$$
$$Q(s_{13}, a_{\leftarrow}) = 0$$
$$Q(s_{13}, a_{\downarrow}) = 0$$
$$Q(s_{13}, a_{\rightarrow}) = 0$$

…

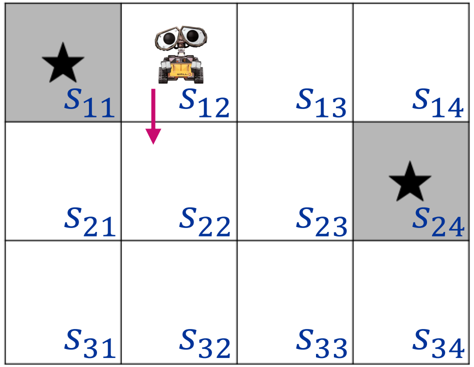
Example: to Find $\hat{Q}$

Suppose the robot is in s_{12}, and takes action $a_{\downarrow}$,

$$Q(s_{12}, a_{\downarrow}) = r_{s_{12},a_{\downarrow}} + \gamma \max\big(Q(s_{22}, a_{\uparrow}), Q(s_{22}, a_{\leftarrow}), Q(s_{22}, a_{\downarrow}), Q(s_{22}, a_{\rightarrow})\big)$$

$$Q(s_{12}, a_{\downarrow}) = -1 + \gamma \max(0,0,0,0) = -1$$

So, we update $Q(s_{12}, a_{\downarrow}) := -1$



$$Q(s_{12}, a_{\uparrow}) = 0$$
$$Q(s_{12}, a_{\leftarrow}) = 0$$
$$Q(s_{12}, a_{\downarrow}) = -1$$
$$Q(s_{12}, a_{\rightarrow}) = 0$$
$$Q(s_{13}, a_{\uparrow}) = 0$$
$$Q(s_{13}, a_{\leftarrow}) = 0$$
$$Q(s_{13}, a_{\downarrow}) = 0$$
$$Q(s_{13}, a_{\rightarrow}) = 0$$

...

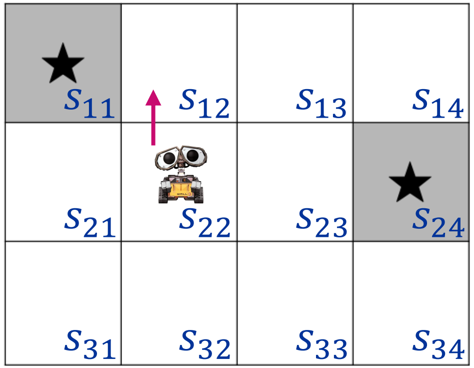
Example: to Find $\hat{Q}$

Then, suppose the robot takes action $a_{\uparrow}$,

$$Q(s_{22}, a_{\uparrow}) = r_{s_{22},a_{\uparrow}} + \gamma \max\big(Q(s_{12}, a_{\uparrow}), Q(s_{12}, a_{\leftarrow}), Q(s_{12}, a_{\downarrow}), Q(s_{12}, a_{\rightarrow})\big)$$

$$Q(s_{22}, a_{\uparrow}) = -1 + \gamma \max(0,0,-1,0) = -1$$

So, we update $Q(s_{22}, a_{\uparrow}) := -1$



$Q(s_{12}, a_{\uparrow}) = 0$
$Q(s_{12}, a_{\leftarrow}) = 0$
$Q(s_{12}, a_{\downarrow}) = -1$
$Q(s_{12}, a_{\rightarrow}) = 0$
$Q(s_{13}, a_{\uparrow}) = 0$
$Q(s_{13}, a_{\leftarrow}) = 0$
$Q(s_{13}, a_{\downarrow}) = 0$
$Q(s_{13}, a_{\rightarrow}) = 0$
…

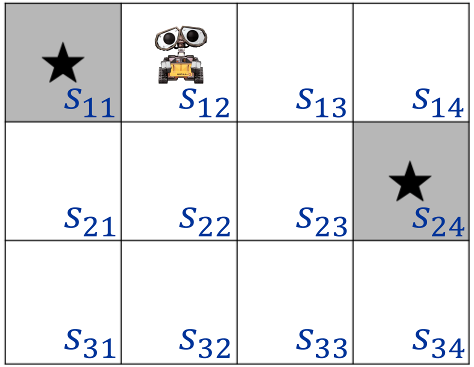
UNIVERSITY OF ABERDEEN

Example: to Find $\hat{Q}$

Now, the robot back to s_{12} again. When it check which action it can take, it will find that:

$$Q(s_{12}, a_{\uparrow}) = 0, Q(s_{12}, a_{\leftarrow}) = 0, Q(s_{12}, a_{\downarrow}) = -1, Q(s_{12}, a_{\rightarrow}) = 0$$

So, probably, it will choose an action from $a_{\uparrow}$, $a_{\leftarrow}$, and $a_{\rightarrow}$.



$$Q(s_{12}, a_{\uparrow}) = 0$$
$$Q(s_{12}, a_{\leftarrow}) = 0$$
$$Q(s_{12}, a_{\downarrow}) = 0$$
$$Q(s_{12}, a_{\rightarrow}) = 0$$
$$Q(s_{13}, a_{\uparrow}) = 0$$
$$Q(s_{13}, a_{\leftarrow}) = 0$$
$$Q(s_{13}, a_{\downarrow}) = 0$$
$$Q(s_{13}, a_{\rightarrow}) = 0$$
$$\dots$$

Example: to Find $\hat{Q}$

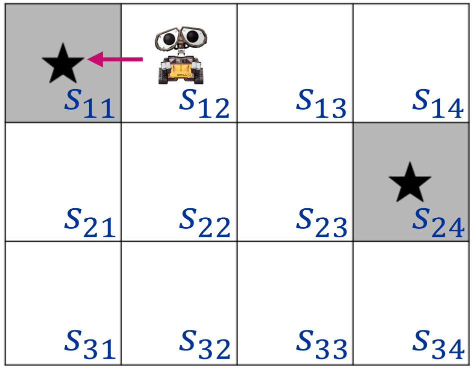
Suppose the robot takes action $a_{\leftarrow}$,

$Q(s_{12}, a_{\leftarrow}) = r_{s_{12},a_{\leftarrow}} + \gamma \max\big(Q(s_{11}, a_{\uparrow}), Q(s_{11}, a_{\leftarrow}), Q(s_{11}, a_{\downarrow}), Q(s_{11}, a_{\rightarrow})\big)$ B

Because the s_{11} is a terminal, we use a large value to represent $Q(s_{11},\cdot)$ e.g. 10.
Suppose $\gamma = 0.9$, they give:

$Q(s_{12}, a_{\leftarrow}) = -1 + 0.9 \times 10 = 8$

Thus, we repeat the process,
the Q will finally approach to $\hat{Q}$.



$Q(s_{12}, a_{\uparrow}) = 0$
$Q(s_{12}, a_{\leftarrow}) = 8$
$Q(s_{12}, a_{\downarrow}) = 0$
$Q(s_{12}, a_{\rightarrow}) = 0$
$Q(s_{13}, a_{\uparrow}) = 0$
$Q(s_{13}, a_{\leftarrow}) = 0$
$Q(s_{13}, a_{\downarrow}) = 0$
$Q(s_{13}, a_{\rightarrow}) = 0$
…

An Iterative Process for Walking Robot Training

<Video: 13 - AI Learns to Walk (deep reinforcement learning).mp4 >

Conclusion

Reinforcement Learning (RL) introduces a framework for decision-making in uncertain environments, focusing on learning optimal behaviours through interactions.

- The Markov Decision Process (MDP) is a core mathematical concept in RL, providing a formalism for modeling decision making where outcomes are partly random and partly under the control of a decision-maker.
- Understanding the Value Function and Q Function is essential for evaluating and optimising the expected long-term rewards in RL scenarios.
- The Bellman Equation plays a pivotal role in RL, serving as the foundation for many algorithms by recursively breaking down the value functions. It links current actions to future rewards.