Learning to Act - Part 2

Robotic Vision Summer School 2024

Pamela Carreno-Medrano and Dana Kulić
Monash University
dana.kulic@monash.edu

[1]The material covered in this lecture is based on David Silver's RL lectures at UCL, Mario Martin's RL lectures at UPC, and Sutton and Barton's Introduction to RL book

Activity 0: Notebook Setup

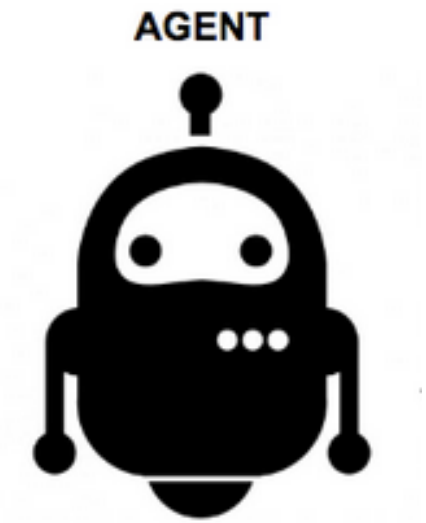
- Please open your Jupyter notebook environment and open the `LearnintToAct Session2` notebook in the `Reinforcement_Learning` folder
- We will be making use of the Gym toolkit: https://gym.openai.com/

What is Reinforcement Learning?

Area of machine learning inspired by behavioural psychology, concerned with how **agents** *ought to take* **actions** *in an (uncertain)* **environment** *so as to maximize some notion of* **reward** (Wikipedia)

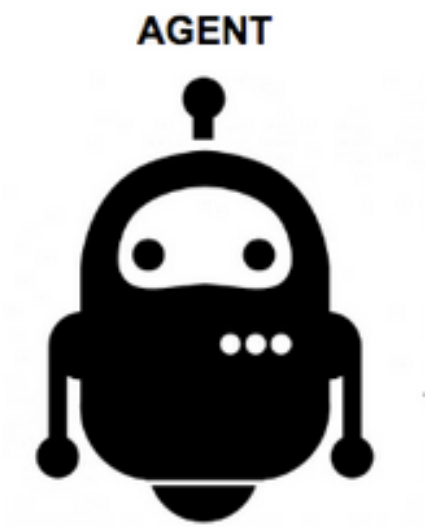
Reinforcement Learning is learning what to do so as to maximize a numerical reward signal (Sutton and Barto)

RL Concept (1)



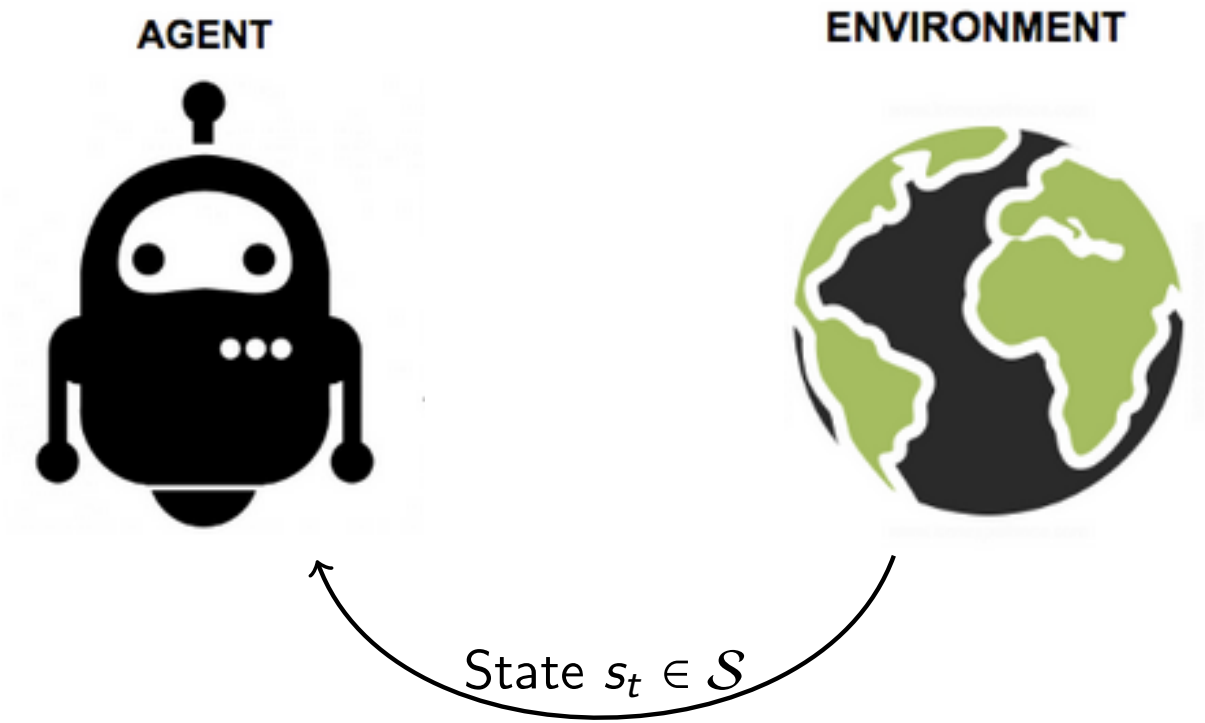
[2]Images courtesy of Lilian Weng

RL Concept (1)



[2]Images courtesy of Lilian Weng

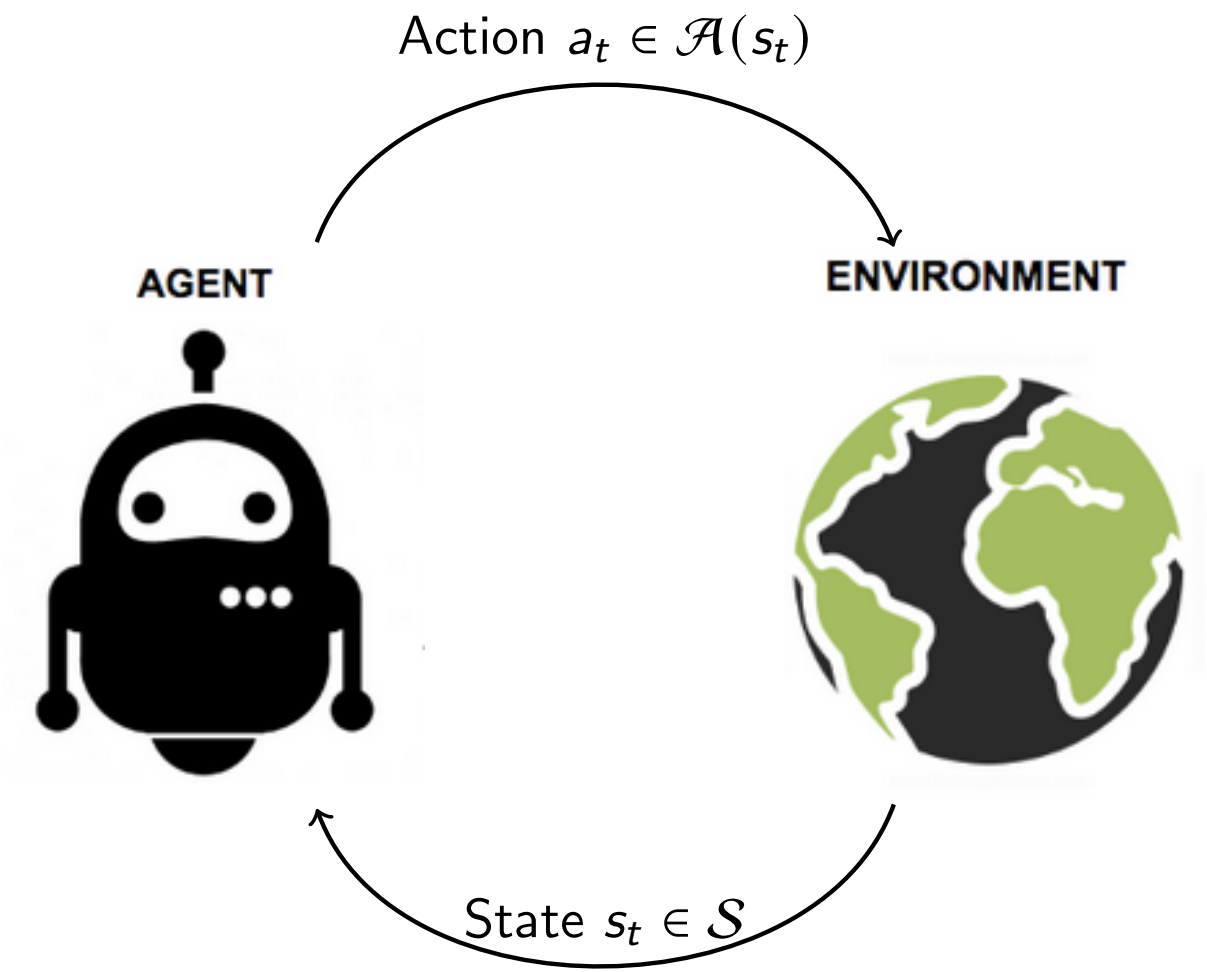
RL Concept (1)



In this lecture we assume full state observability, i.e., the agent can directly and accurately observe the state of the environment.

[2]Images courtesy of Lilian Weng

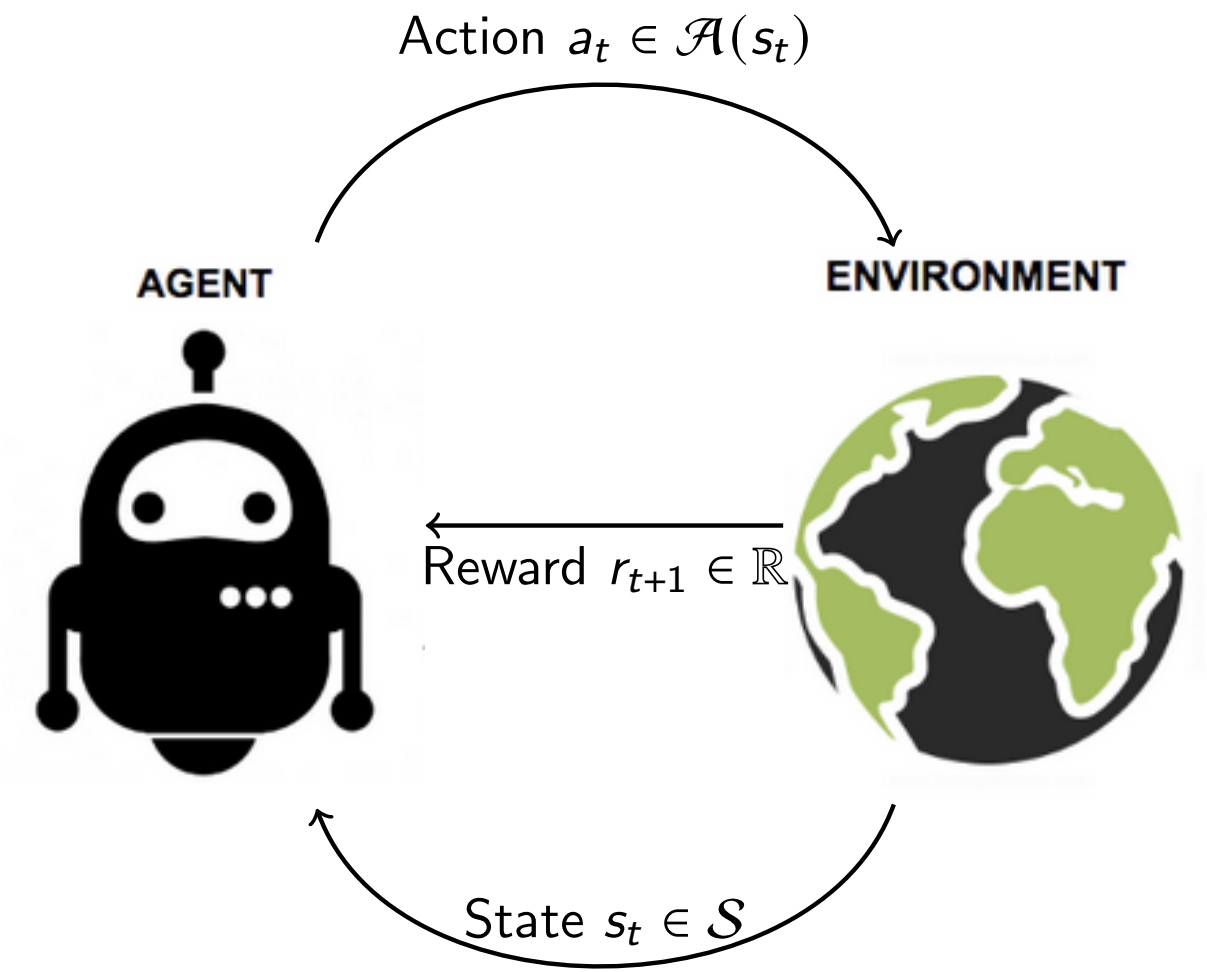
RL Concept (1)



In this lecture we assume full state observability, i.e., the agent can directly and accurately observe the state of the environment.

[2]Images courtesy of Lilian Weng

RL Concept (1)



In this lecture we assume full state observability, i.e., the agent can directly and accurately observe the state of the environment.

[2]Images courtesy of Lilian Weng

RL Concept (2)

The agent and the environment interact at discrete time steps $t = \{1, 2, \dots\}$. At each step t,

the agent:

- Observes state s_t
- Executes action a_t
- Receives scalar reward r_{t+1}

the environment:

- Receives action a_t
- Emits state s_{t+1}
- Emits scalar reward r_{t+1}

RL Concept (2)

The agent and the environment interact at discrete time steps $t = \{1, 2, \dots\}$. At each step t,

the agent:

- Observes state s_t
- Executes action a_t
- Receives scalar reward r_{t+1}

the environment:

- Receives action a_t
- Emits state s_{t+1}
- Emits scalar reward r_{t+1}

The RL problem: what sequence of actions should the agent take to maximise reward?

Examples of Reinforcement Learning

Autonomous Driving

Figure: Wayve.ai - A car learns to follow a lane from scratch

Examples of Reinforcement Learning

Robot Manipulation

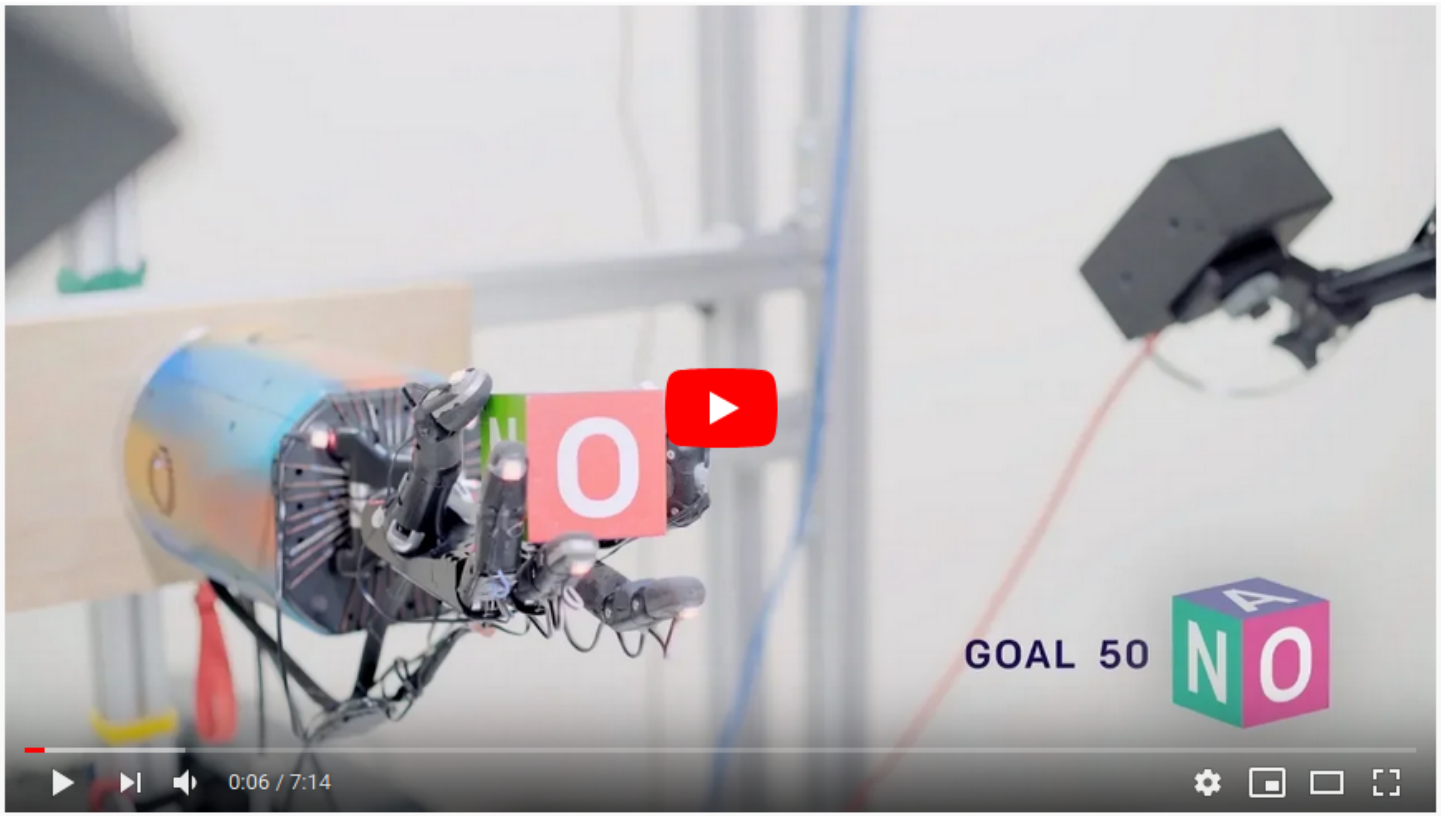


Figure: OpenAI - Learning Dexterity

Recall our Mathematical Formulation: MDP

- Markov Decision Processes (MDP) provide a general formalism for modeling and describing decision-making under uncertainty (i.e., RL and planning problems).
- MDPs are tuples $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ where
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - $\mathcal{T}$ is a transition function that defines the dynamics of the environment

$$\mathcal{T}(s_t, a_t, s_{t+1}) = \mathbb{P}(s_{t+1}|s_t, a_t)$$

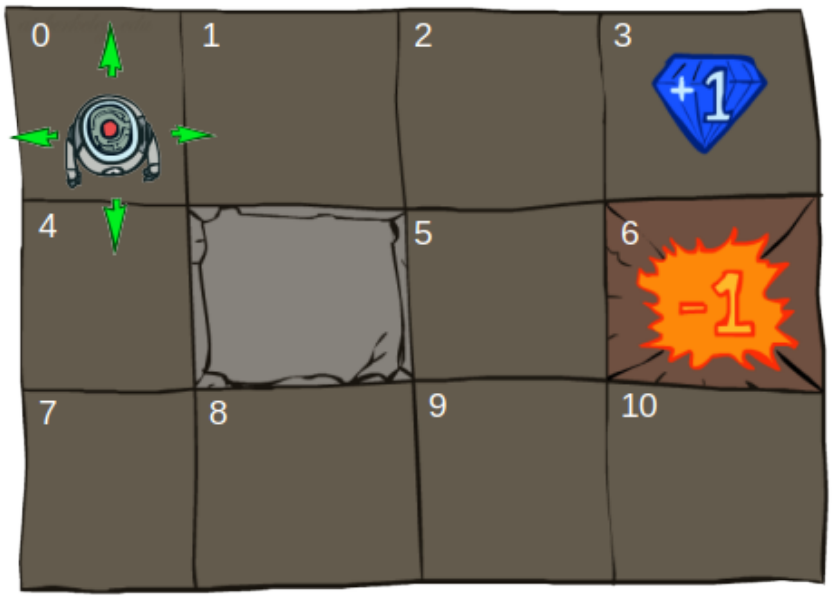
  - $\mathcal{R}$ is a reward function

$$\mathcal{R}(s_t, a_t, s_{t+1}) = r_{t+1}, \text{ or}$$
$$\mathcal{R}(s_t, a_t) = \mathbb{E}[r_{t+1}|s_t, a_t] = \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s_t, a_t)\mathcal{R}(s_t, a_t, s')$$

  - γ is a discount factor $\gamma \in [0, 1]$ that defines the present value of future rewards

MDP Example (1)

Grid World

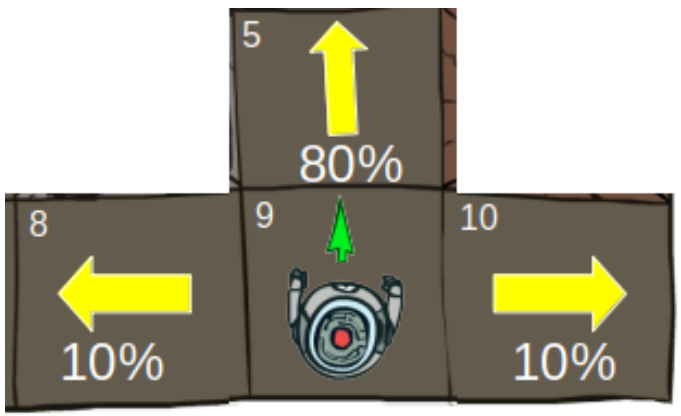


- $\mathcal{S}$ = Cells of the grid with indexes $\{0, 1, \ldots, 10\}$
- $\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\}$

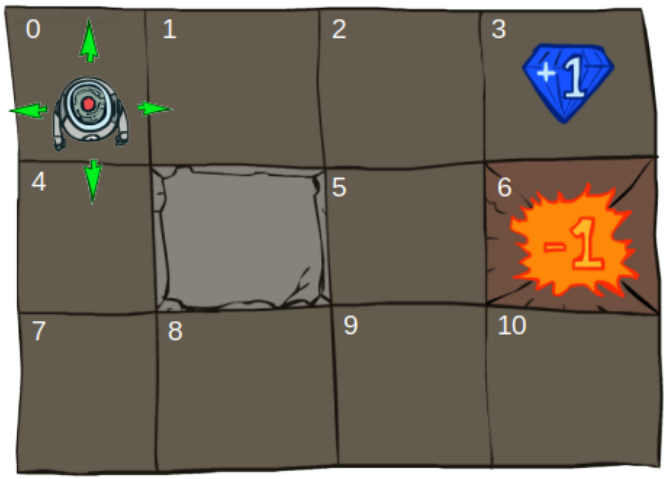
MDP Example (2)

Grid World

- State transitions are noisy



- Walls block the agent's path



- Big rewards at terminal states
- Small running cost at each step, e.g., $\mathcal{R}(4, \text{up}, 1) = -0.04$

Activity 1: Rewards of an MDP

- Please go back to your Jupyter notebook environment and the `LearningToAct-Session2` notebook in the `Reinforcement_Learning` folder
- Take a look at Section 1. Rewards for an MDP

Policies in Finite MDPs

- Formally, a policy π is a distribution of actions given states,

$$\pi(a|s) = \mathbb{P}(a_t = a|s_t = s)$$

- Policies fully define the behaviour of an agent because they specify how to act in any state
- We consider stationary (time-independent) policies,

$$a_t \sim \pi(\cdot|s_t), \forall\, t > 0$$

- Given a finite MDP, we want to find the optimal policy π^*, i.e., **the policy that maximises the expected return if followed**.

Value Functions in Finite MDPs

A value function defines the amount of reward an agent can expect to accumulate over the future under a particular policy π

The *state-value* function $v_\pi(s)$ is the expected return when starting in state s and following π thereafter,

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right], \forall s \in \mathcal{S}$$

The *action-value* function $q_\pi(s, a)$ is the expected return starting from state s, taking action a and following π thereafter,

$$q_\pi(s, a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right]$$

Value Functions and Bellman Expectation Equations

Both the *state-value* and *action-value* functions can be decomposed into the immediate reward plus the discounted value of successor state.

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \ldots) | s_t = s\right] \\
&= \mathbb{E}_\pi\left[\underbrace{r_{t+1}}_{\text{Immediate reward}} + \overbrace{\gamma v_\pi(s_{t+1})}^{\text{Discounted value}} | s_t = s\right]
\end{aligned}
$$

The same decomposition applies to the *action-value* function $q_\pi(s, a)$

$$
q_\pi(s, a) = \mathbb{E}_\pi\left[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\right]
$$

The relationship between the value functions

Value functions in terms of rewards:

$$v_\pi(s) = \mathbb{E}_\pi\Big[r_{t+1} + \gamma v_\pi(s_{t+1})|s_t = s\Big]$$

$$q_\pi(s,a) = \mathbb{E}_\pi\Big[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})|s_t = s, a_t = a\Big]$$

Substituting in for when the model is known:

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\Big(\mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a,s')v_\pi(s')\Big)$$

$$q_\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a,s') \sum_{a'\in\mathcal{A}} \pi(a'|s')q_\pi(s',a')$$

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)q_\pi(s,a)$$

$$q_\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a,s')v_\pi(s')$$

Solving the RL Problem with the Bellman equation

We use the fundamental insight of the Bellman equation to solve 3 related problems:

- Policy Evaluation: what is the *value* of a given policy?
- Policy Iteration: Can we find a policy that maximises the value?
- Value Iteration: Can we estimate the value and optimise the policy at the same time?

Iterative Policy Evaluation (1)

Given an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, we want to compute the *state-value* function for an arbitrary policy π.

Solution (intuition):

- Start with an arbitrary guess $v_0(s)$ that is an estimate of $v_\pi(s)$
- Improve estimate $v_i(s)$ by iteratively applying Bellman equation for all states until convergence

$$\underbrace{v_{i+1}(s)}_{\text{iteration i+1}} \leftarrow \underbrace{\sum_{a \in \mathcal{A}} \pi(a|s) \Big(\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') v_i(s') \Big)}_{\text{iteration i}}$$

- To figure out when to stop, we can compute the error (Δ) using

$$\Delta \leftarrow \max_{s \in \mathcal{S}} \Big| v_{i+1}(s) - v_i(s) \Big|$$

Iterative Policy Evaluation (2)

Algorithm 1: Iterative Policy Evaluation for estimating $v \approx v_\pi$

Input : MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, policy π, threshold $\theta > 0$
Output: $v_\pi(s)$
Initialize $v(s) = 0 \, \forall s \in \mathcal{S}$
repeat
  $\Delta \leftarrow 0$
  foreach $s \in \mathcal{S}$ **do**
    $V \leftarrow v(s)$
    $v(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \Big(\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') v(s') \Big)$
    $\Delta \leftarrow \max(\Delta, |V - v(s)|)$
until $\Delta < \theta$

Activity 2: Policy Evaluation

- Please go back to your Jupyter notebook environment and the `Session 1 IntroRL` notebook in the `Reinforcement_Learning` folder
- Take a look at Section 2, Iterative Policy Evaluation
- What happens to the value when you change the policy?

Finding the Optimal Policy

Optimal Policy Through Policy Iteration (1)

A policy can be improved iif

$$\exists s \in \mathcal{S}, a \in \mathcal{A} \text{ such that } q_\pi(s, a) > q_\pi(s, \pi(a))$$

If this condition is met, how do we improve π?

Solution (intuition):

- Evaluate the policy π using *policy evaluation*
- Improve the policy by acting *greedily* with respect to $v_\pi(s)$

Optimal Policy Through Policy Iteration (2)

Using policy iteration, we can obtain a sequence of monotonically improving value policies and value functions

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \ldots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} v^*$$

This process of policy iteration always converges to π^*

Optimal Policy Through Policy Iteration (3)

Algorithm 2: Policy Iteration for estimating $\pi \approx \pi^*$

Input : MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$

Output: $\pi \approx \pi^*$

Initialize $\pi(s)\, \forall s \in \mathcal{S}$ to a random action $a \in \mathcal{A}$, arbitrarily

repeat

  $\pi' \leftarrow \pi$

  Compute $v_\pi(s)$ for all states using *policy evaluation*

  foreach $s \in \mathcal{S}$ **do**

    $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') v_\pi(s')$

until $\pi(s) == \pi'(s)\, \forall s \in \mathcal{S}$

Activity 3: Policy Iteration

- Please go back to your Jupyter notebook environment and the `Session 1 IntroRL` notebook in the `Reinforcement_Learning` folder
- Take a look at Section 4, Policy Iteration

Value Iteration

Policy iteration requires a complete execution of policy evaluation before improvement

Can we do better?

Solution (intuition):

- Stop *policy evaluation* after one update of each state
- Improve the policy by acting *greedily* with respect to $v(s)$

Value Iteration - Algorithm

Algorithm 3: Value Iteration for estimating $\pi \approx \pi^*$

Input : MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, error threshold $\theta > 0$

Output: $\pi \approx \pi^*$

Initialize $v(s) = 0 \, \forall s \in \mathcal{S}$

repeat

  $\Delta \leftarrow 0$

  foreach $s \in \mathcal{S}$ **do**

    $V_{old} \leftarrow v(s)$

    $v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') v(s') \right)$

    $\Delta \leftarrow \max(\Delta, |V_{old} - v(s)|)$

until $\Delta < \theta$

Return deterministic policy π, such that:

$\pi(s) = \arg\max_{a \in \mathcal{A}} \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') v_\pi(s')$

Summary

- Reinforcement learning is about learning how to behave in an environment to achieve a specified goal
- The key elements of a RL problem are
  - **Reward**: what are the good and bad states/actions in the environment
  - **Policy**: how the agent chooses its actions
  - **Model**: how the environment changes as a result of the agent's actions (and randomness)
  - **Value function**: what is good and bad in the long run (i.e., prediction of future reward)
- The optimal value functions $v^*(s)$ and $q^*(s, a)$ describe the largest expected return achievable by any policy π
- If we know $q^*(s, a)$ and/or $v^*(s)$, we immediately have the optimal policy

Looking ahead to tomorrow's lecture

- Up until now, we have assumed that we know how the environment responds to agent actions (i.e., we know $\mathcal{T}(s, a, s')$) and where the rewards are in the environment (i.e., we know $\mathcal{R}(s, a)$)
- This allows us to figure out the best policy without having to interact with the environment at all
- Tomorrow's lecture: How can we learn if we don't know the environment and the rewards ahead of time?