LEARNING TO ACT

Session 2: Imitation Learning (+RL)

Dana Kulić
Monash University
dana.kulic@monash.edu

RVSS 2026

Session 2 Overview

- Imitation Learning
  - Inverse Reinforcement Learning
  - Behavioural Cloning
- A brief peek at reinforcement learning

Review of policy methods

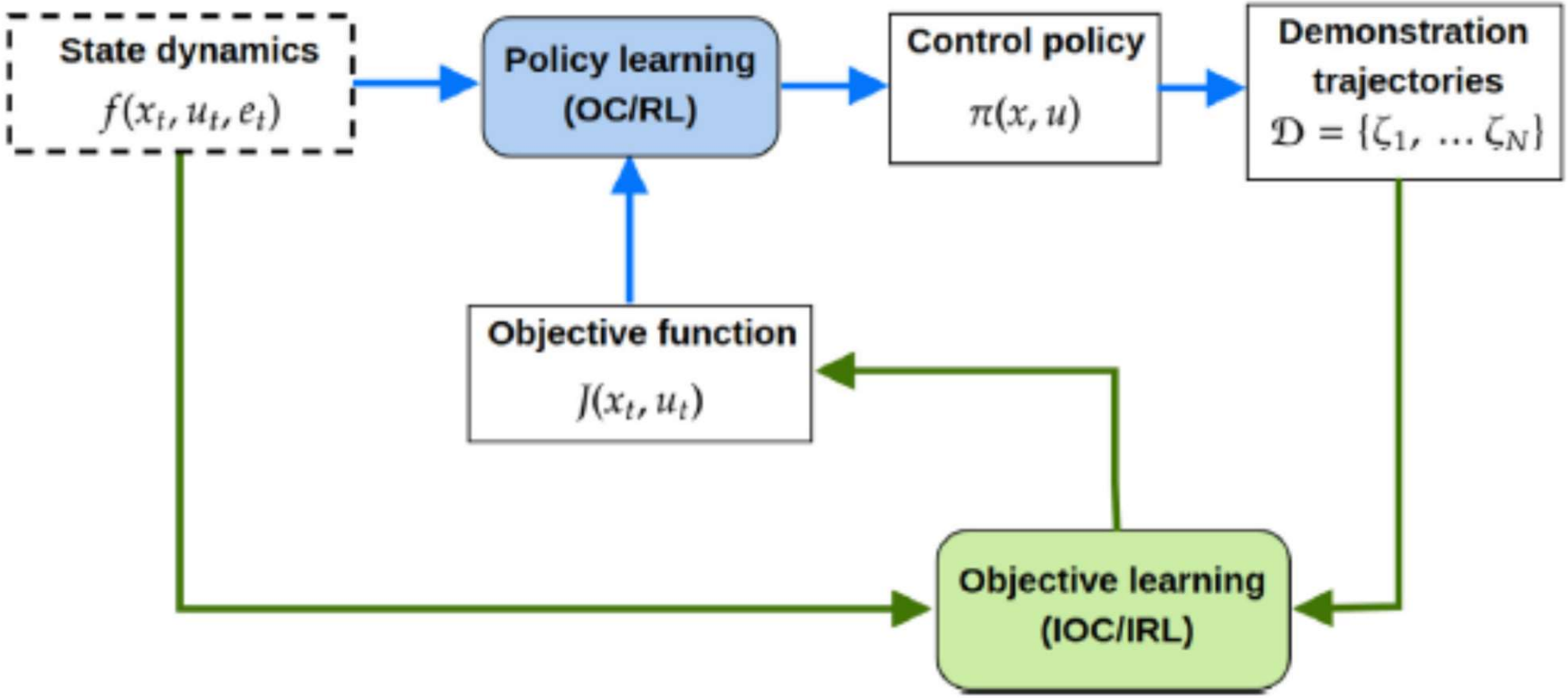
Method	Transition Function	Reward Function	Expert Demos
Optimal Control	Known	Known	No
Reinforcement Learning	Known or Unknown	Known	No
Imitation Learning – IRL	Known or Unknown	Unknown	Yes
Imitation Learning – BC	Known or Unknown	Implicit	Yes

Imitation Learning

- What if the reward is hard to formulate?
- And I have an expert who can show me the best way to do the task?

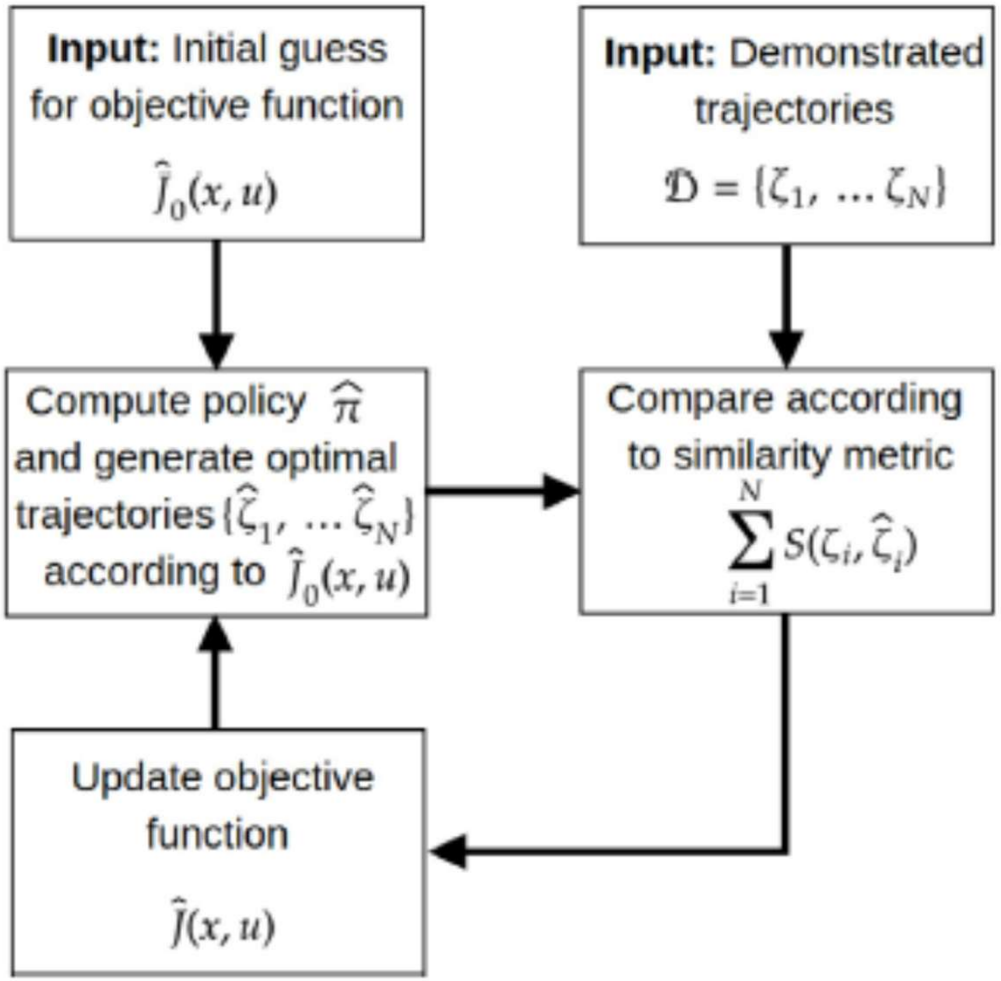
- Two approaches:
  - Inverse Reinforcement Learning (IRL): Use the human demonstrations to learn the reward function, then use RL or OC to learn the policy
  - Behaviour Cloning (BC): Learn the policy directly

Learning the Objective (Cost) Function



J. F. S. Lin, P. Carreno-Medrano, M. Parsapour, M. Sakr, D. Kulić, Objective learning from human demonstrations, Annual Reviews in Control, 2021.

Bi-level objective learning



J. F. S. Lin, P. Carreno-Medrano, M. Parsapour, M. Sakr, D. Kulić, Objective learning from human demonstrations, Annual Reviews in Control, 2021.

Behavioural Cloning

- Collect data from demonstration episodes $\mathcal{D}(e_{1:N})$
- Each episode is a sequence of states and actions $e_i = (s_0, a_1, s_1, a_2, \ldots s_T)$
- Learn a policy $\phi(s)$ using supervised learning:

$$L = (a_{\mathcal{D}}(s) - \phi(s))^2$$

  - The state s corresponds to the input data
  - The action a corresponds to the label
  - Behavioural cloning learns the policy function $\phi(s)$ to minimise the difference between the estimated action and the observed expert action from each state

Behavioural Cloning

Algorithm 1 Abstract of behavioral cloning

Collect a set of trajectories demonstrated by the expert $\mathcal{D}$
Select a policy representation π_θ
Select an objective function $\mathcal{L}$
Optimize $\mathcal{L}$ w.r.t. the policy parameter θ using $\mathcal{D}$
return optimized policy parameters θ

Key Design Choices:

- What loss function should be used to represent the difference between demonstrated and produced behaviour?
- How should the state be represented?
- How should the policy be represented?
- Output a single action or a trajectory?

T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel and J. Peters, An Algorithmic Perspective on Imitation Learning, Foundations and Trends in Robotics, 2018.

Behavioural Cloning

- What might be some problems we encounter when we apply behavioural cloning?

Behavioural Cloning – potential problems

- How to collect the expert data?
- What is the right state representation? Does the robot see the same things as the expert does?
- Expert demonstrations may cover only a very small region of the state-space
  - For large state/action spaces, may require a huge data collection effort
- What should the robot do when it encounters a situation that wasn't seen in the dataset?
- How to handle variations in strategy?

Behavioural Cloning – potential problems

- **How to collect the expert data?**
- What is the right state representation? Does the robot see the same things as the expert does?
- Expert demonstrations may cover only a very small region of the state-space
  - For large state/action spaces, may require a huge data collection effort
- What should the robot do when it encounters a situation that wasn't seen in the dataset?
- How to handle variations in strategy?

How to collect expert data?

From human demonstrations directly

Via teleoperation

How to collect demonstration data?

Properties of demonstration data

- Do I get the state only or both the state and action?
- How valid is the assumption that the expert data is optimal?
  - Intuitive interface
  - Sensor noise
  - Demonstrator expertise
- How much of the state-space/task variation is covered?
- Are there variations in strategy?

Are there other ways to collect expert input?

Are there other ways to collect expert input?

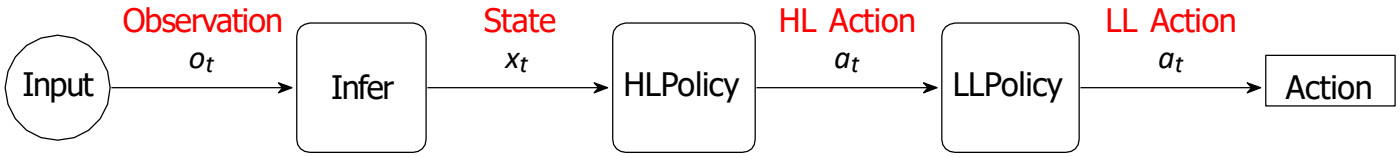
- Preference Learning:
  - Use preferences to learn the human reward function - Reinforcement Learning from Human Feedback (RLHF, Christiano et al. 2017)
  - Use preferences to learn the policy directly – Direct Preference Optimisation (DPO – Rafailov et al. 2023)
  - Make the process more efficient by encoding prior on likely human biases – Kahneman-Tversky Optimisation (KTO – Ethayarajh et al. 2024)
- Unifying multiple human signals for learning (Jeon et al., 2020)
- Use (simulated) optimisation solutions as demonstrations

Behavioural Cloning – potential problems

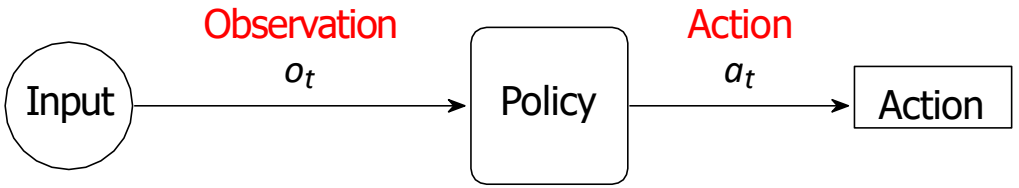
- How to collect the expert data?
- **What is the right state representation? Does the robot see the same things as the expert does?**
- Expert demonstrations may cover only a very small region of the state-space
  - For large state/action spaces, may require a huge data collection effort
- What should the robot do when it encounters a situation that wasn't seen in the dataset?
- How to handle variations in strategy?

Choices for Problem formulation

Modular



End-to-end



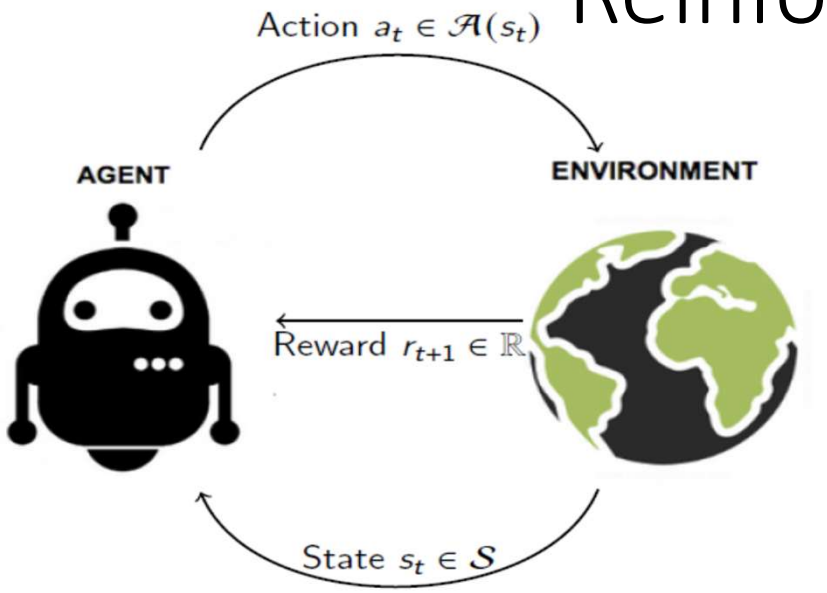
Behavioural Cloning – potential problems

- How to collect the expert data?
- What is the right state representation? Does the robot see the same things as the expert does?
- Expert demonstrations may cover only a very small region of the state-space
  - For large state/action spaces, may require a huge data collection effort
- What should the robot do when it encounters a situation that wasn't seen in the dataset?
- How to handle variations in strategy?

Let’s try it out!

- Please open `IntroBC.ipynb` from the `Reinforcement_Learning` folder in the school repo

Reinforcement Learning



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}

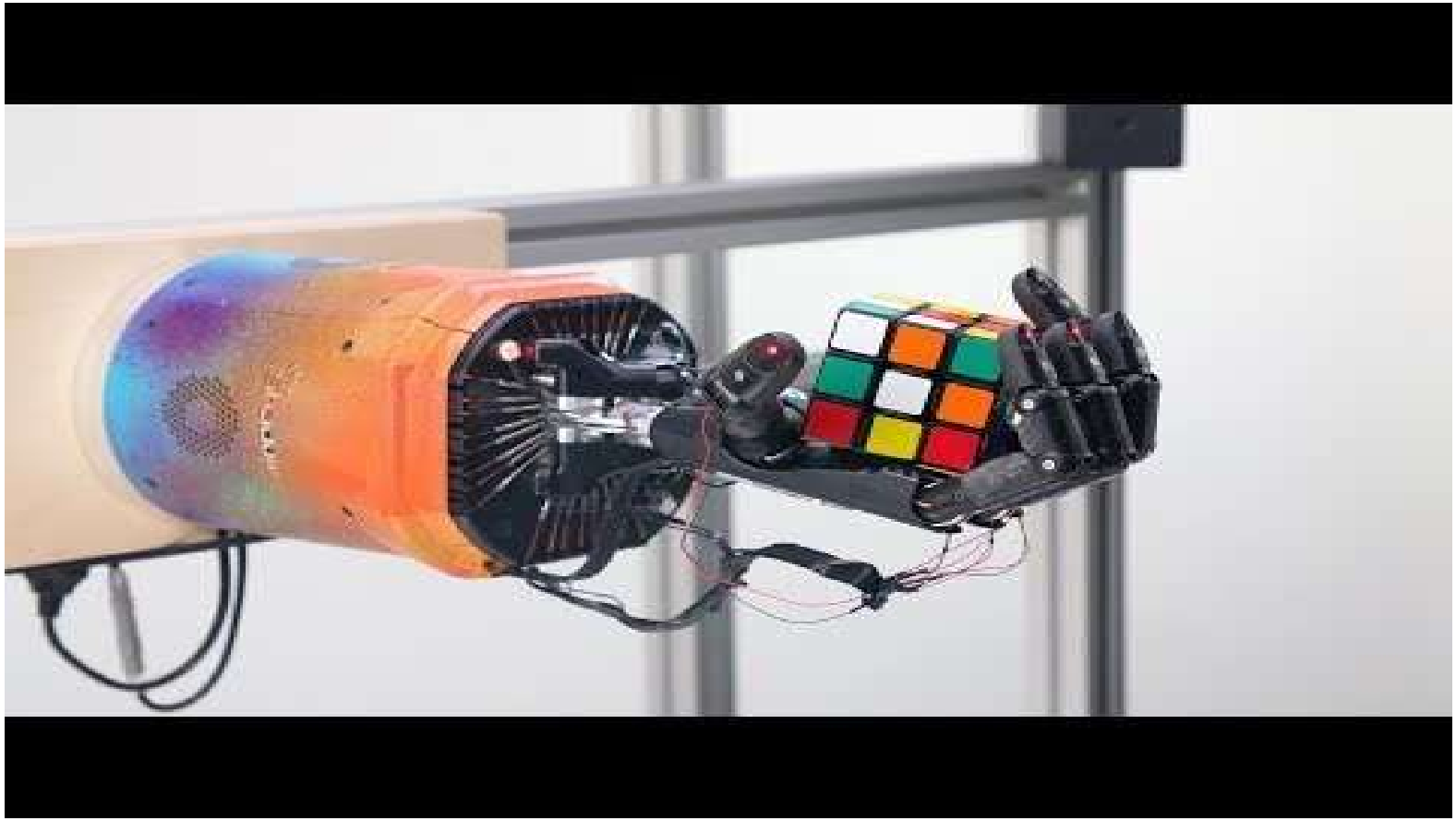
the environment:

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

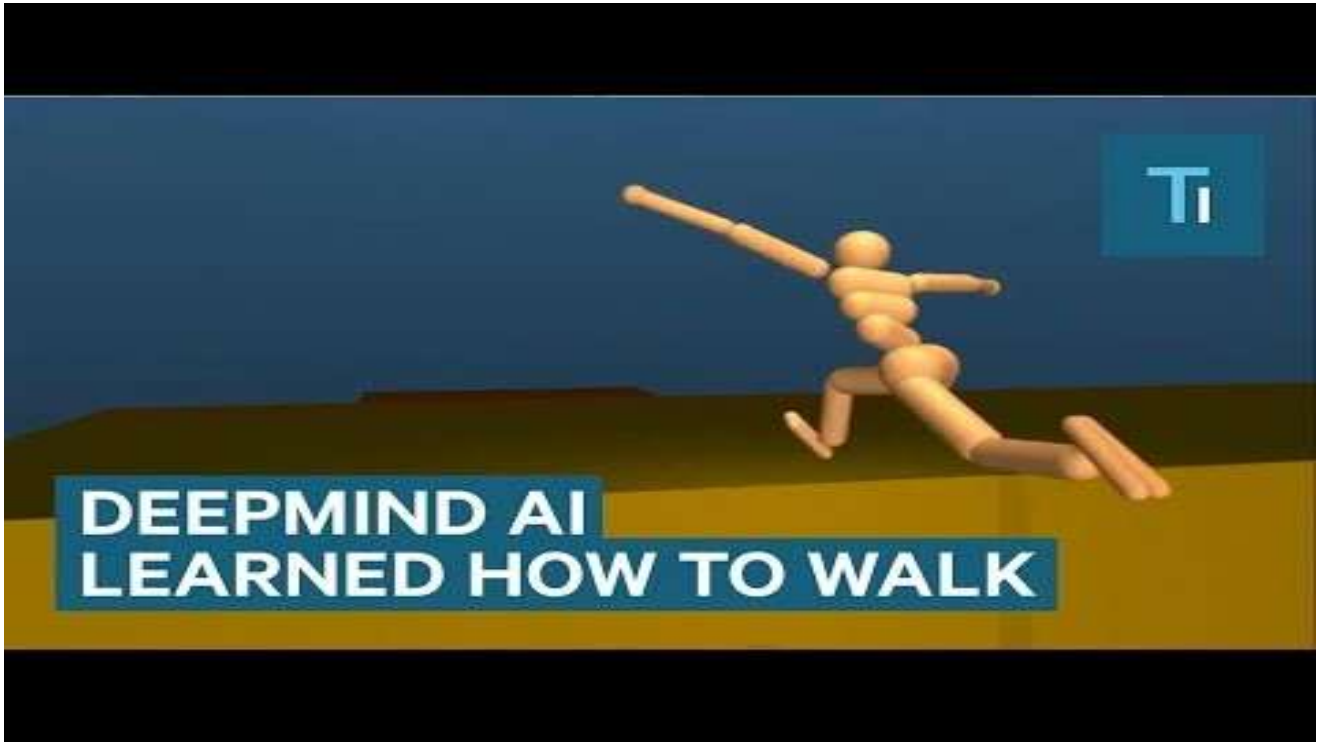
Image Courtesy of Lilian Weng

RL Examples

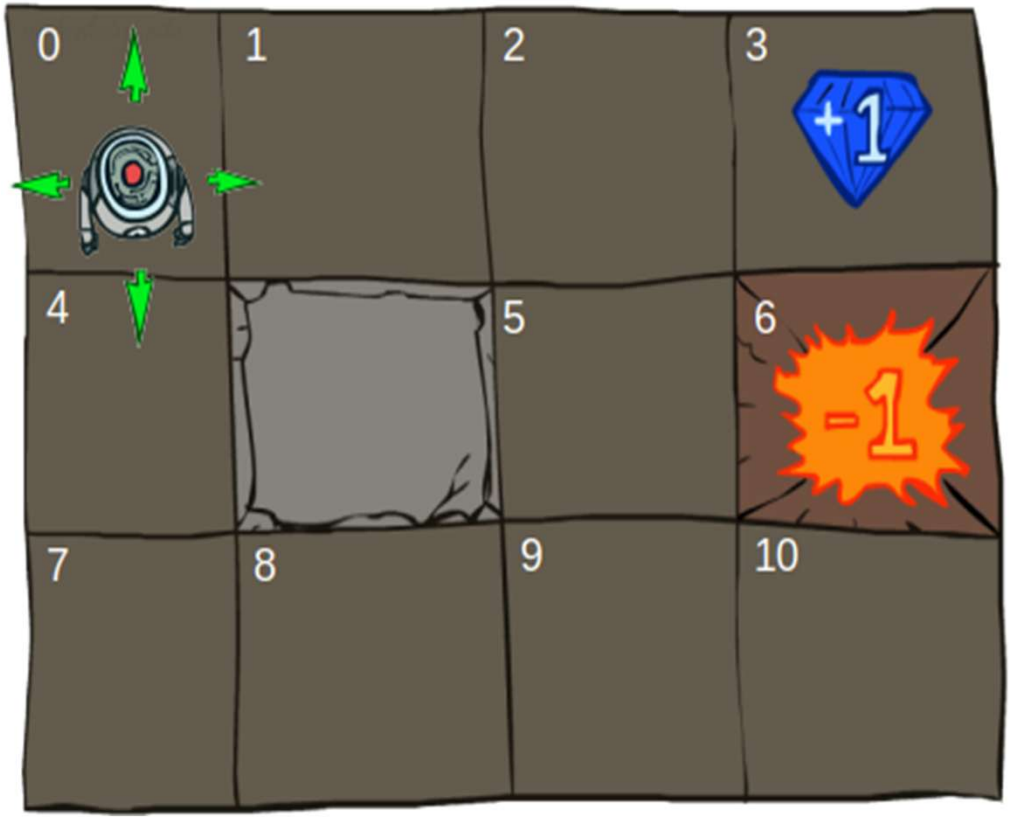
RL Examples



RL Examples



The problem with maximising reward from experience



Value Functions

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

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

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

Value Functions and Bellman Equations

- The state-value function can be decomposed into the immediate reward plus the discounted value of the successor state

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

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

Optimal Value and Policy

The optimal value function produces the maximum return:

$$V_*(s) = \max_\pi V_\pi(s),$$

The optimal policy achieves optimal value functions:

$$\pi_* = \arg\max_\pi V_\pi(s).$$

How does this help us to learn the best policy?

- Start with some policy guess
- Estimate how good that policy is (by estimating its value function)
- Improve the policy
- Iterate -> Policy Iteration

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

Algorithm for Policy Evaluation

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**

    $V \leftarrow v(s)$

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

    $\Delta \leftarrow \max(\Delta, |V - v(s)|)$

until $\Delta < \theta$

Algorithm for Policy Improvement

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

Let’s try it out!

- Please open `IntroRL.ipynb` from the `Reinforcement_Learning` folder in the school repo

What if we don't have the model? – Temporal Difference Learning

Goal: Given a policy π, learn $\hat{v}_\pi(s)$ from experience episodes

$$\{s_0, a_0, r_1, \ldots, s_T\} \sim \pi$$

Recall: State-value Bellman Equation

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

Approximation: At each time step t use *observed immediate* reward r_{t+1} and the estimated return $\hat{v}(s_{t+1})$ to update $\hat{v}(s_t)$

$$\hat{v}(s_t) \leftarrow \hat{v}(s_t) + \alpha \overbrace{[\underbrace{r_{t+1} + \gamma \hat{v}(s_{t+1})}_{\text{TD target}} - \hat{v}(s_t)]}^{\text{TD error}},$$

where α is a step-size paratemer.

We update our sample estimate $\hat{v}(s_t)$ in the direction of the TD error.

Q-learning

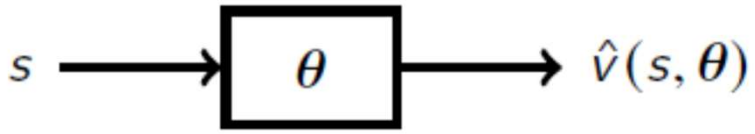
Algorithm 2: Q-learning algorithm

Input : Step-size $\alpha \in (0, 1]$, small $\epsilon > 0$
Output: $\hat{q}^*(s, a)$
Initialize $\hat{q}(s, a)$ arbitrarily $\forall s \in \mathcal{S}\ a \in \mathcal{A}$, $q(\text{terminal state}, \cdot) = 0$
Loop *for each episode*
    Initialize s
    repeat
        Choose action a from s using ϵ-greedy policy derived from $\hat{q}(s, a)$
        Take action a, observe r, s'
        $\hat{q}(s, a) \leftarrow \hat{q}(s, a) + \alpha[r + \gamma \max_{a'} \hat{q}(s', a') - \hat{q}(s, a)]$
        $s \leftarrow s'$
    until s *is terminal*

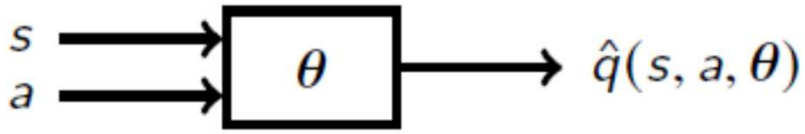
Let’s try it out!

- Please open `TD-RL.ipynb` from the `Reinforcement_Learning` folder in the school repo

Function Approximation



$$\hat{v}(s, \theta) \approx v_\pi(s)$$



$$\hat{q}(s, a, \theta) \approx q_\pi(s, a)$$

Function Approximation

- If we knew the true action-value function, this would be a standard supervised learning problem, we could find the best approximation by minimising:

$$J(\theta) = \mathbb{E}_\pi[(q_\pi(s,a) - \hat{q}_\pi(s,a,\theta))^2]$$

- But RL only gives access to rewards. Use Bellman equation

$$\Delta\theta = \alpha[\underbrace{r_{t+1} + \gamma\hat{q}_\pi(s_{t+1}, a_{t+1}, \theta)}_{\text{Target}} - \hat{q}_\pi(s_t, a_t, \theta)]\nabla_\theta\hat{q}_\pi(s_t, a_t, \theta)$$

DQN Algorithm

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N

Initialize action-value function Q with random weights

Implemented in main loop

for episode $= 1, M$ **do**

  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

  for $t = 1, T$ **do**

    With probability ϵ select a random action a_t

    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$

    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

  end for

end for

Implemented in optimize(.) method of the Agent class

Let’s try it out!

- Please open `DeepRL_BasicDQN.ipynb` from the `Reinforcement_Learning` folder in the school repo

- Further exploration: check out `DeepRL_TargetDQN.ipynb`

Types of RL Algorithms

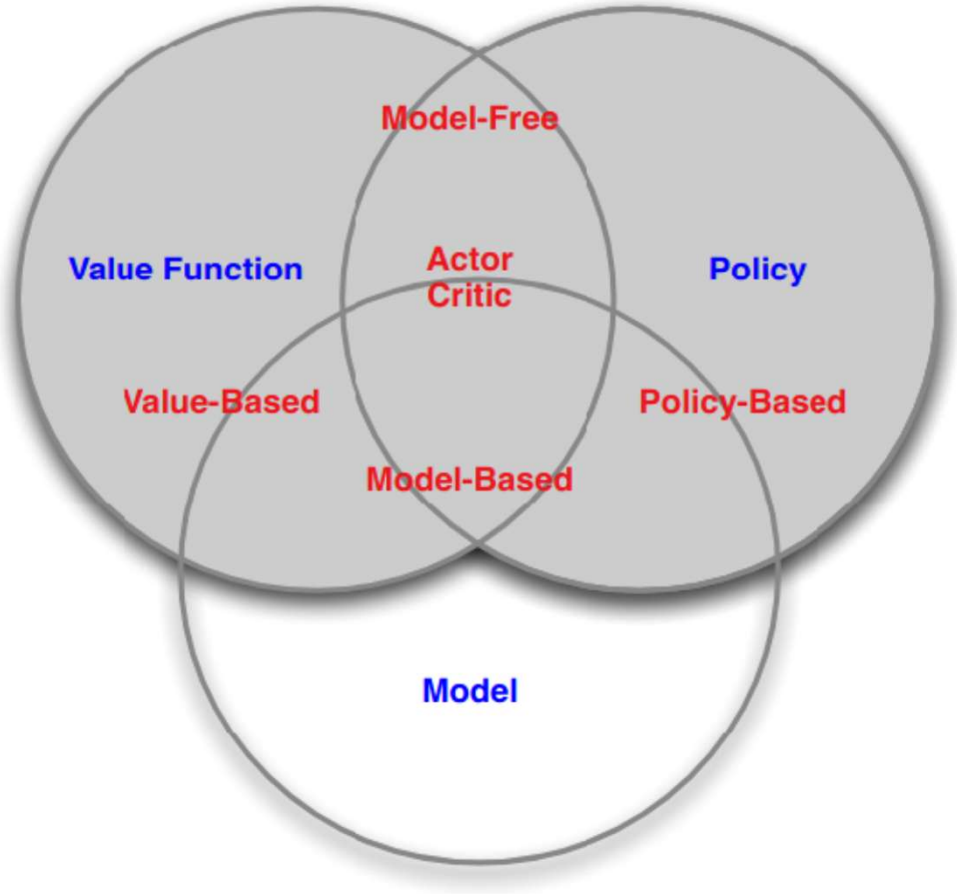


Image Courtesy of David Silver

Function approximation for value/policy learning

- Collect data from executions in the replay buffer $e_t = (S_t, A_t, R_t, S_{t+1})$
- Sample a minibatch of data from the replay buffer
- Update Value function with Bellman loss

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

- Update policy parameters to maximise the value

$$\mathcal{J}(\theta) = \sum_{s\in\mathcal{S}} d_{\pi_\theta}(s) V_{\pi_\theta}(s) = \sum_{s\in\mathcal{S}} \left(d_{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi(a|s,\theta) Q_\pi(s,a)\right)$$

Challenges of RL

- Exploitation vs. exploration
- Reward Formulation
- “The Deadly Triad”: bootstrapping+off-policy+approximation
- Sim2real gap

Summary of This Session

- Imitation Learning:
  - Don’t need to know the reward
  - Must collect “expert” demonstrations
  - Many design choices about problem formulation
- Reinforcement Learning:
  - Must specify the reward
  - From reward to value
  - From value to policy