Learning to Act - Part 1

Robotic Vision Summer School 2024

Dana Kulić
Monash University
dana.kulic@monash.edu

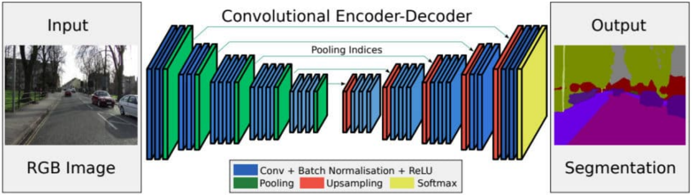
[1]The material covered in this lecture is based on David Silver's RL lectures at UCL, Mario Martin's RL lectures at UPC, and Sutton and Barton's Introduction to RL book

Activity 0: Notebook Setup

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

From perception to action

- Robot vision allows the robot to perceive its environment



- Perception is a mapping from sensory data (e.g. pixels) to percepts (labels)
- This lecture: how do we map from sensory data to action

[2]Images courtesy of Derrick Mwiti

Characteristics of robot action

Why take action?

- To accomplish (hopefully useful) tasks
- To improve perception
- To improve the robot's knowledge about the environment

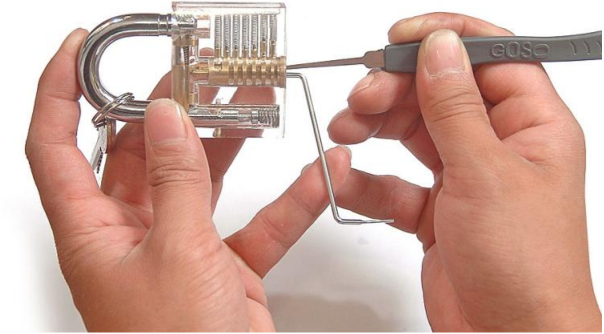
Consequences of action

- Actions modify the world
- Actions can be costly

https://www.youtube.com/watch?v=Mt-1smlom_M

Figure: A robot takes actions with consequences (Image courtesy of theverge.com)

Action Selection



- Action Selection is a sequential decision-making problem

Formalizing Sequential Decision-Making Problems

Mathematical Formulation: MDP

- Markov Decision Processes (MDP) provide a general formalism for modeling and describing decision-making under uncertainty.

Mathematical Formulation: MDP

- Markov Decision Processes (MDP) provide a general formalism for modeling and describing decision-making under uncertainty.
- 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
  - γ is a discount factor $\gamma \in [0, 1]$ that defines the present value of future rewards

Mathematical Formulation: MDP

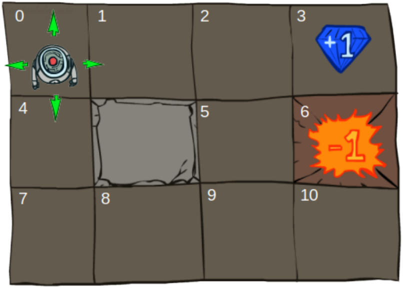
- Markov Decision Processes (MDP) provide a general formalism for modeling and describing decision-making under uncertainty.
- MDPs are tuples $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ where
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set 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
  - γ is a discount factor $\gamma \in [0, 1]$ that defines the present value of future rewards
- Why do we call this a *Markov* decision process?

MDP Example (1)

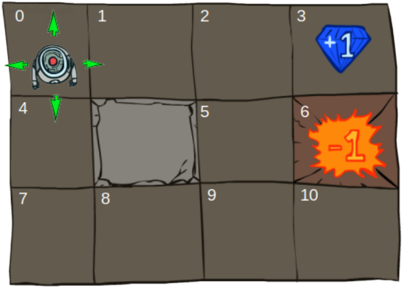
Grid World



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

MDP Example (1)

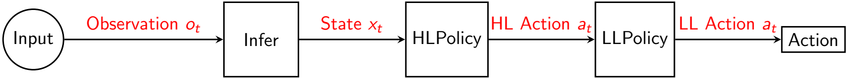
Grid World



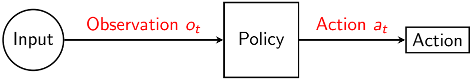
- $\mathcal{S}$ = Cells of the grid with indexes $\{0, 1, \ldots, 10\}$
  - Is this the only choice for states?
- $\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\}$
  - Is this the only choice for actions?

Choices for Problem formulation

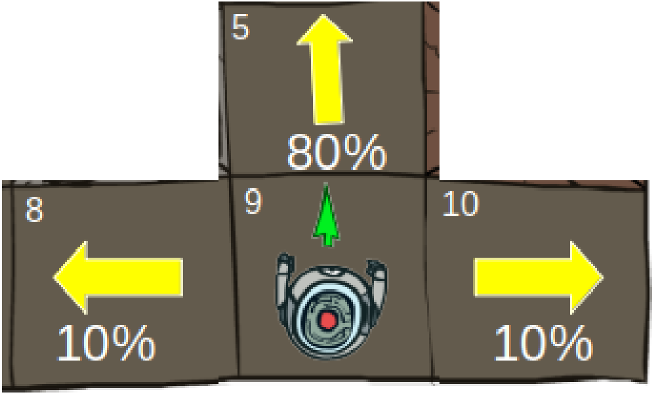
Modular



End-to-end



- State transitions are noisy



- Walls block the agent's path

Activity 1: Elements of an MDP

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

Policies in Finite MDPs

- A policy is a mapping from states (or observations) to actions
- Types of policies:
  - Deterministic policy: a function that takes state as input and outputs an action

    $$a_t = \pi(s_t)$$

  - Stochastic policy: a distribution over actions given states)

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

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

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

Policy Learning

How to get the best policy?

- Replicate an expert's policy - imitation learning
- Learn from trial and error - reinforcement learning

Imitation Learning

(specifically, Behavioural Cloning)

- Collect data from demonstration episodes $\mathcal{D}(e_{1:N})$
- Each episode is a sequence of states and actions $e_i = (s_0, a_1, s_1, a_2, ...s_T)$
- 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

Activity 2: Behavioural cloning in our toy grid world

- Please go back to your Jupyter notebook environment and the `Session 1 IntroRL` notebook in the `Reinforcement_Learning` folder
- Take a look at Section 3. Behavioural cloning

Behavioural Cloning

Potential Problems:

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

Often combine BC+RL to get best of both worlds, e.g., Lu et al. 2023

Lu et al., Imitation Is Not Enough: Robustifying Imitation with Reinforcement Learning for Challenging Driving Scenarios. https://waymo.com/research/imitation-is-not-enough-robustifying-imitation-with-reinforcement-learning/

Some considerations for using Behavioural Cloning in the Workshop

- What will make a good dataset?
  - Should you use your best driver? Your worst driver? Or a combination of drivers?
  - What should the driver be doing?
- How should you formulate the problem?
  - Modular: estimate the robot state first, then learn a policy that maps *states* to actions
  - End-to-end: learn a policy that maps directly from *observations* to actions
- How should you represent your policy?