LEARNING TO ACT

Session 1: Intro

Dana Kulić
Monash University
dana.kulic@monash.edu

RVSS 2025

Overview of the Series

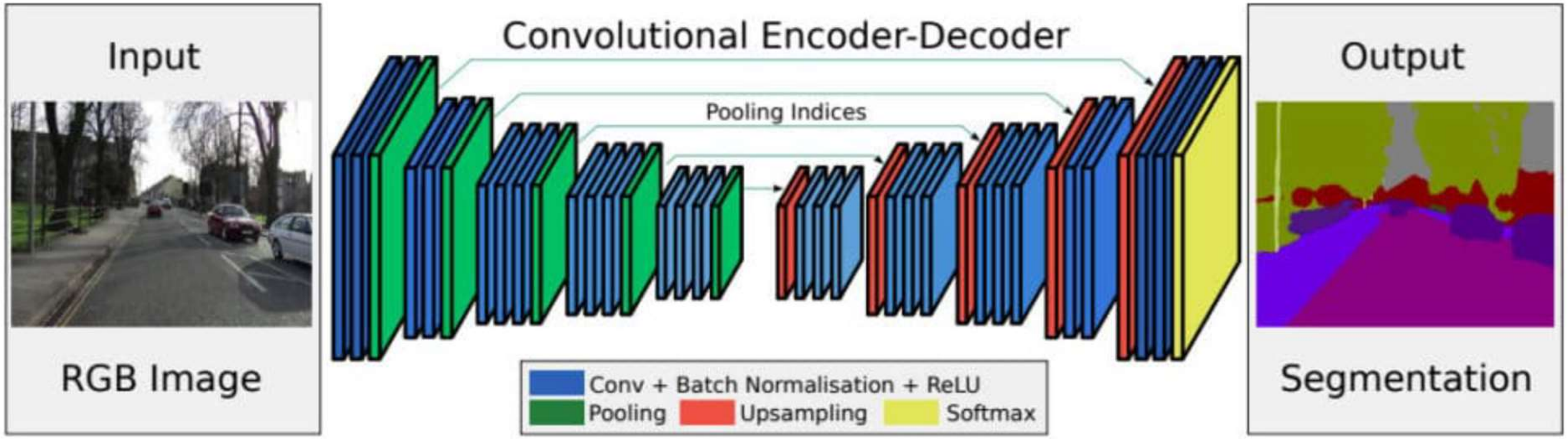
- Session 1
  - Introduction and Problem Formulation
  - Overview of solution methodologies
  - Introduction to Reinforcement Learning and Imitation Learning
- Session 2
  - Behaviour Cloning
  - Generative/Foundation Models for Policy Generation
- Session 3
  - From cool research to something practical?

Session 1

- Introduction to policy learning
- Problem formulation as a Markov Decision Process
- Solution approaches
    - Optimisation / Optimal Control
    - Reinforcement Learning
    - Imitation Learning
        - Inverse Optimal Control
        - Behavioural Cloning

From perception to action

- Robot vision allows the robot to perceive its environment

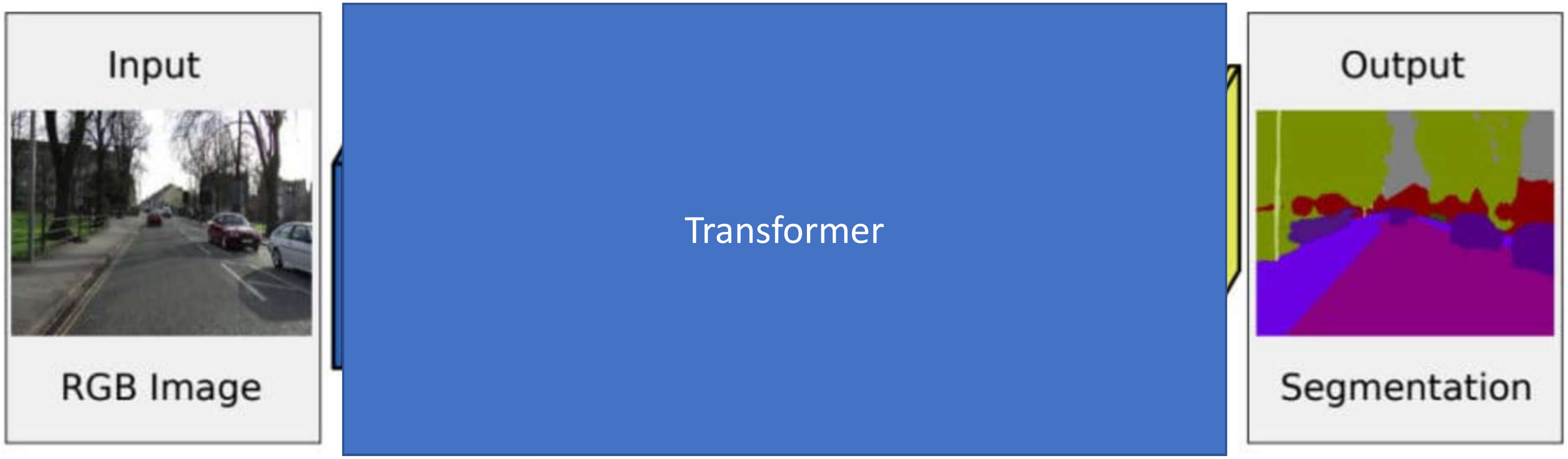


- Perception is a mapping from sensory data (e.g., pixels) to percepts (labels)
- This lecture: how do we map from sensory data to **action**

Image Courtesy of Derrick Mwiti

From perception to action

- Robot vision allows the robot to perceive its environment

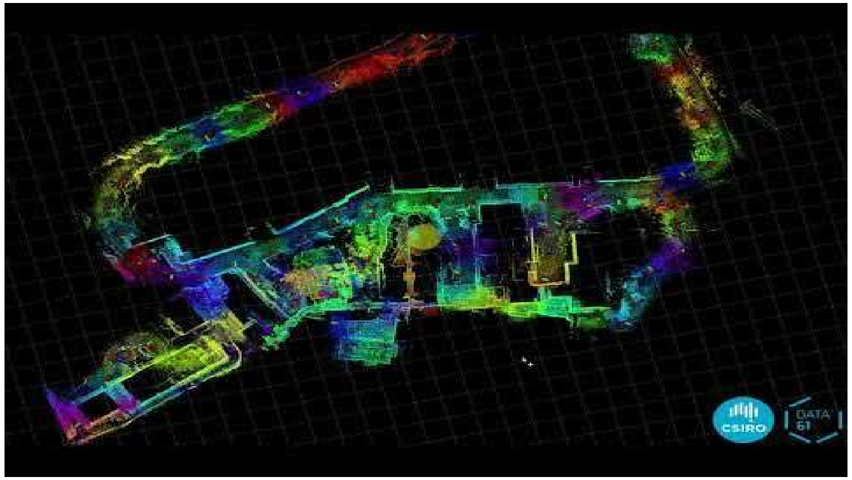


- Perception is a mapping from sensory data (e.g., pixels) to percepts (labels)
- This lecture: how do we map from sensory data to **action**

Image Courtesy of Derrick Mwiti

Characteristics of robot action

- Why take action?



Characteristics of Action

- Actions have consequences

Characteristics of Action

- Consequences may not be immediately apparent -> sequential decision making problem

Formalising Sequential Decision-Making Problems

Markov Decision Process (MDP)

- A general formalism for modelling 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

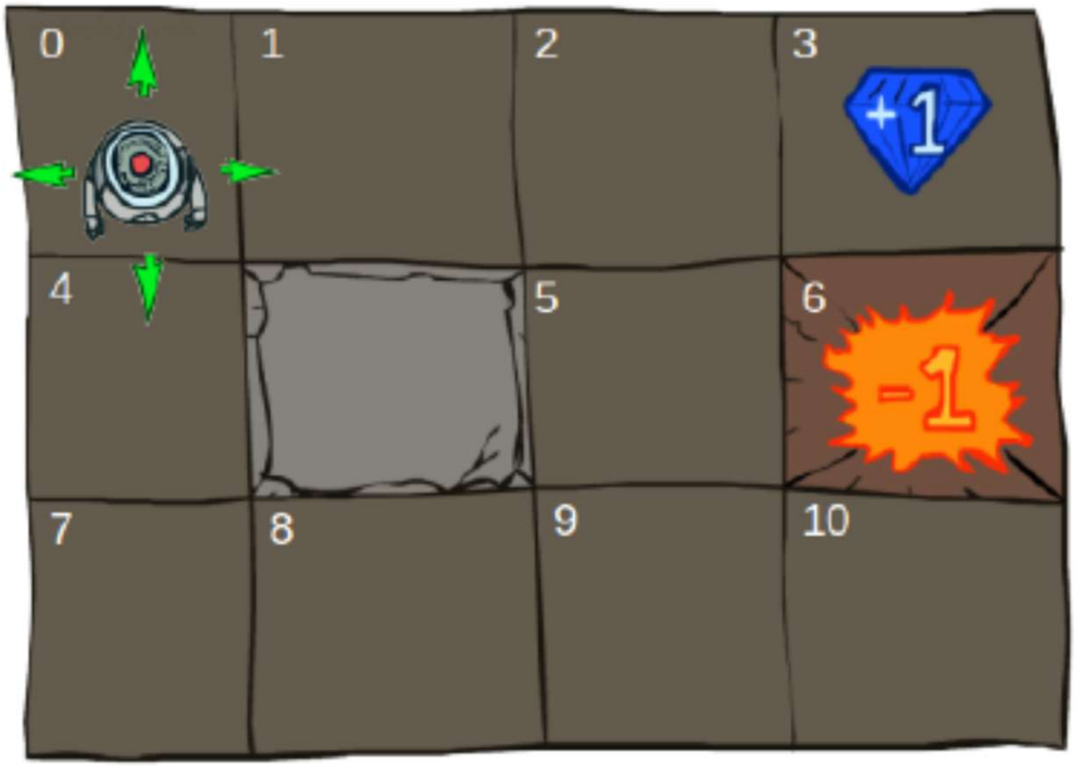
$$\mathcal{R}(s_t, a_t, s_{t+1}) = r_{t+1}, \text{ or}$$

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

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

MDP Example

- State set
- Action set
- Transition function
- Reward
- Discount



Policies in MDPs

- To solve an MDP problem, we want to find a **policy** that maximises the reward
- A **policy** is a mapping from states to actions

Types of policies:

- Deterministic policy: a function that takes the state as input and outputs an action

$$a_t = \pi(s_t)$$

- Stochastic policy: a distribution over actions given states

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

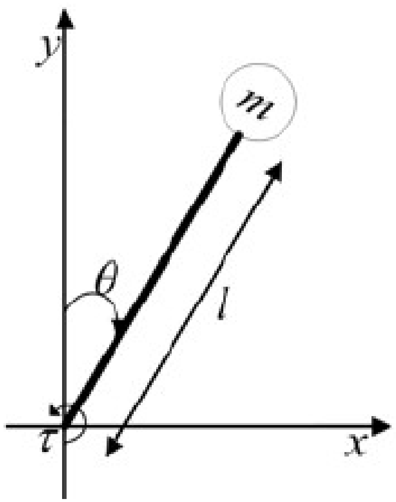
Policy Learning

How to figure out the best policy?

How to figure out the best policy?

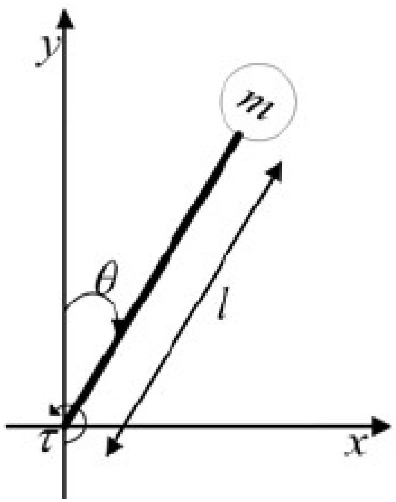
- Solve analytically or numerically via optimisation/optimal control
  - Assumes we know the reward and the transition function
- Learn from trial and error – reinforcement learning
  - Assumes we know (or can formulate) the reward
- Learn from an expert teacher – imitation learning

Optimisation / Optimal Control



- A single link (limb)
  - Link length l
  - Link mass (concentrated at the end) m
  - Link position θ
  - Link torque τ
- How to model this system?
  - What is the state space of this system?
  - What is the action space?
  - What is the transition function?

Single Link Model



- Link length l
- Link mass m, inertia I
- Link position θ
- Link torque τ

- Apply Euler's equation (rotational motion equivalent of Newton's 2nd law)

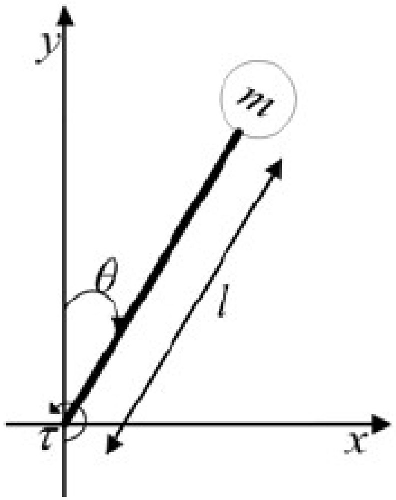
$$\Sigma\tau_e = I\alpha$$

- Equation of motion for the single link

$$\tau - mgl\sin\theta = I\frac{d^2\theta}{dt^2} = ml^2\frac{d^2\theta}{dt^2}$$

- Is this equation linear?
- Is my model complete?

Linearised optimal control



- Link length l
- Link mass m, inertia I
- Link position θ
- Link torque τ

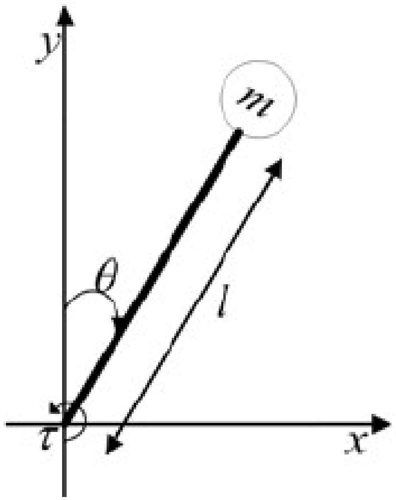
- First, let's try simplifying by linearising the equations

$$\tau - mgl\sin\theta = I\frac{d^2\theta}{dt^2} = ml^2\frac{d^2\theta}{dt^2}$$

$$\tau \approx I\frac{d^2\theta}{dt^2}$$

Linearised optimal control

$$\tau \approx I\frac{d^2\theta}{dt^2}$$



- Link length l
- Link mass m, inertia I
- Link position θ $\quad \mathbf{x} = [\theta \quad \dot{\theta}]^T$
- Link torque τ $\quad u = \tau$

- Re-write the equations into state-space form

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

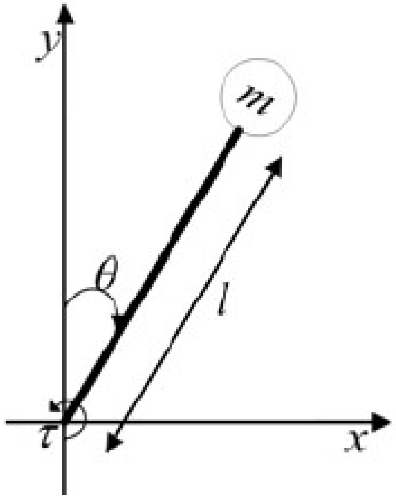
- How do I convert my system equation into this form?

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}, u = \tau$$

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{I} \end{bmatrix} u$$

Linearised optimal control

$$\tau \approx I \frac{d^2\theta}{dt^2}$$



- Link length l
- Link mass m, inertia I
- Link position θ $\quad \mathbf{x} = [\theta \quad \dot{\theta}]^T$
- Link torque τ $\quad u = \tau$

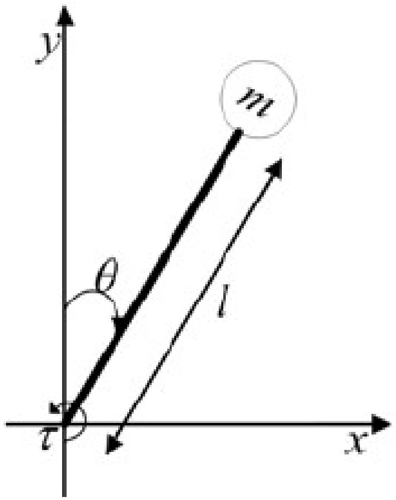
- Given our system equations and some objective function, find the control input u

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

- Want to keep the link upright, i.e. $\theta_d = 0$
- What should be the objective (cost) function?

Linearised optimal control

$$\tau \approx I \frac{d^2\theta}{dt^2}$$



- Link length l
- Link mass m, inertia I
- Link position θ $\quad \mathbf{x} = [\theta \quad \dot{\theta}]^T$
- Link torque τ $\quad u = \tau$

- Given our system equations and some objective function, find the control input u

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

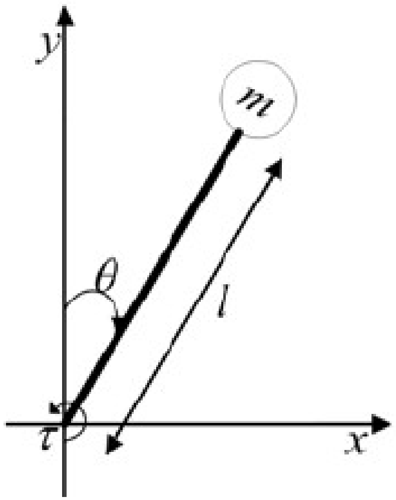
- What should be the cost function?

$$J = \int_0^\infty (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u}) dt$$

Infinite horizon

Linearised optimal control

$$\tau \approx I \frac{d^2\theta}{dt^2}$$



- Link length l
- Link mass m, inertia I
- Link position θ $\quad \mathbf{x} = [\theta \quad \dot{\theta}]^T$
- Link torque τ $\quad u = \tau$

- We can also formulate a discrete version of the problem

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t$$

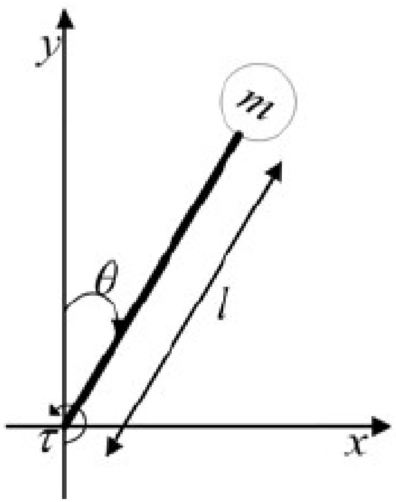
- What should be the cost function?

$$J = \sum_{k=0}^{\infty} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u})$$

infinite horizon

Linearised optimal control

$$\tau \approx I \frac{d^2\theta}{dt^2}$$



- Link length l
- Link mass m, inertia I
- Link position θ $\mathbf{x} = [\theta \quad \dot{\theta}]^T$
- Link torque τ $u = \tau$

- For the system $\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t$ and cost

$$J = \sum_{k=0}^{\infty} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u})$$

- The optimal policy has the form

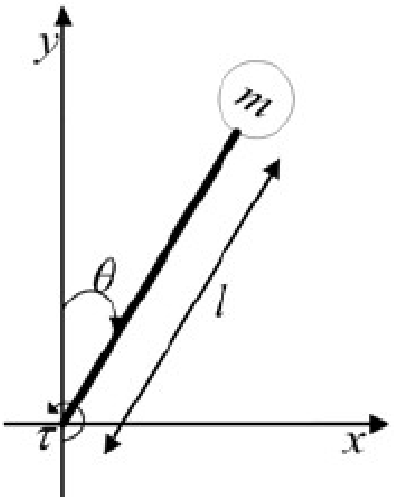
$$\mathbf{u}_t = -K\mathbf{x}_t$$

- Where K is the solution to the discrete time algebraic Ricatti equation

$$K = (R + B^T P B)^{-1}(B^T P A)$$
$$P = A^T P A - (A^T P B)(R + B^T P B)^{-1}(B^T P A) + Q$$

Linearised optimal control – finite horizon

$$\tau \approx I\frac{d^2\theta}{dt^2}$$



- Link length l
- Link mass $m,$ inertia I
- Link position θ $\quad \mathbf{x} = [\theta \quad \dot{\theta}]^T$
- Link torque τ $\quad u = \tau$

- Given the system:

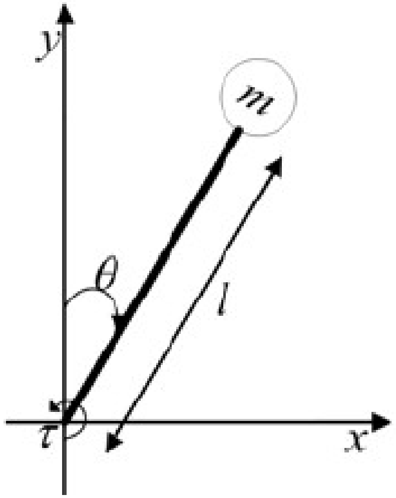
$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t$$

- And cost function

$$J = \mathbf{x}_N^T Q_N \mathbf{x}_N + \sum_{k=0}^{N-1} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u})$$

Linearised optimal control – finite horizon

$$\tau \approx I\frac{d^2\theta}{dt^2}$$



- Link length l
- Link mass m, inertia I
- Link position θ $\mathbf{x} = [\theta \quad \dot{\theta}]^T$
- Link torque τ $u = \tau$

- For the system $\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t$ and cost

$$J = \mathbf{x}_N^T Q_N \mathbf{x}_N + \sum_{k=0}^{N-1} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u})$$

- The optimal controller has the form

$$\mathbf{u}_t = -K_t \mathbf{x}_t$$

- Where K_t needs to be computed iteratively over the horizon

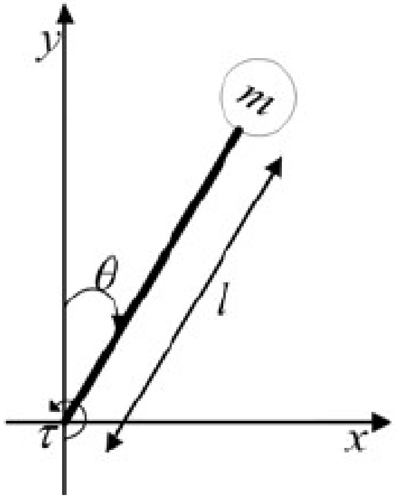
$$P_N := Q_f$$

$$P_{t-1} := Q + A^T P_t A - A^T P_t B(R + B^T P_t B)^{-1} B^T P_t A$$

$$K_t := -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$$

Linearised optimal control – finite horizon

$\tau \approx I\frac{d^2\theta}{dt^2}$



- Link length l
- Link mass m, inertia I
- Link position θ $\mathbf{x} = [\theta \quad \dot{\theta}]^T$
- Link torque τ $u = \tau$

- For the system $\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t$ and cost

$$J = \mathbf{x}_N^T Q_N \mathbf{x}_N + \sum_{k=0}^{N-1} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u})$$

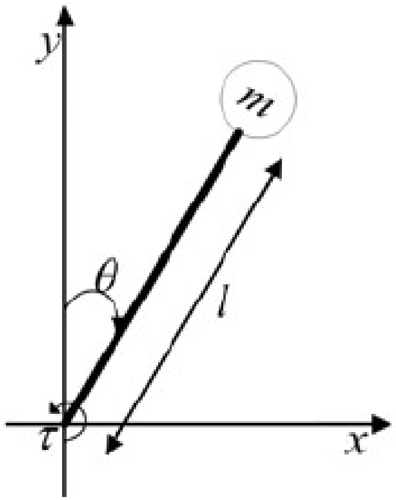
- The optimal controller has the form

$$\mathbf{u}_t = -K_t \mathbf{x}_t$$

- A note about implementation:
  - Open-loop: pre-compute all ***u*** and apply in sequence
  - Closed-loop: pre-compute all ***u***, apply ***u*₀**, re-compute in next timestep – Model Predictive Control (MPC)

Linearised optimal control – finite horizon

$$\tau \approx I \frac{d^2\theta}{dt^2}$$



- Link length l
- Link mass m, inertia I
- Link position θ $\mathbf{x} = [\theta \quad \dot{\theta}]^T$
- Link torque τ $u = \tau$

- For the system $\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t$ and cost

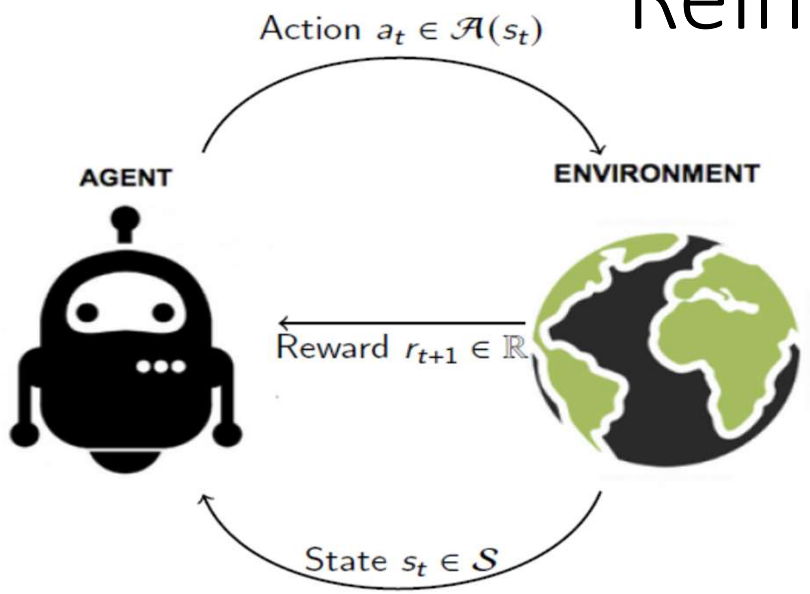
$$J = \mathbf{x}_N^T Q_N \mathbf{x}_N + \sum_{k=0}^{N-1} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u})$$

- The optimal controller has the form

$$\mathbf{u}_t = -K_t \mathbf{x}_t$$

- What assumptions/simplifications did I need to make to get this policy?

Reinforcement Learning



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

the agent:

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

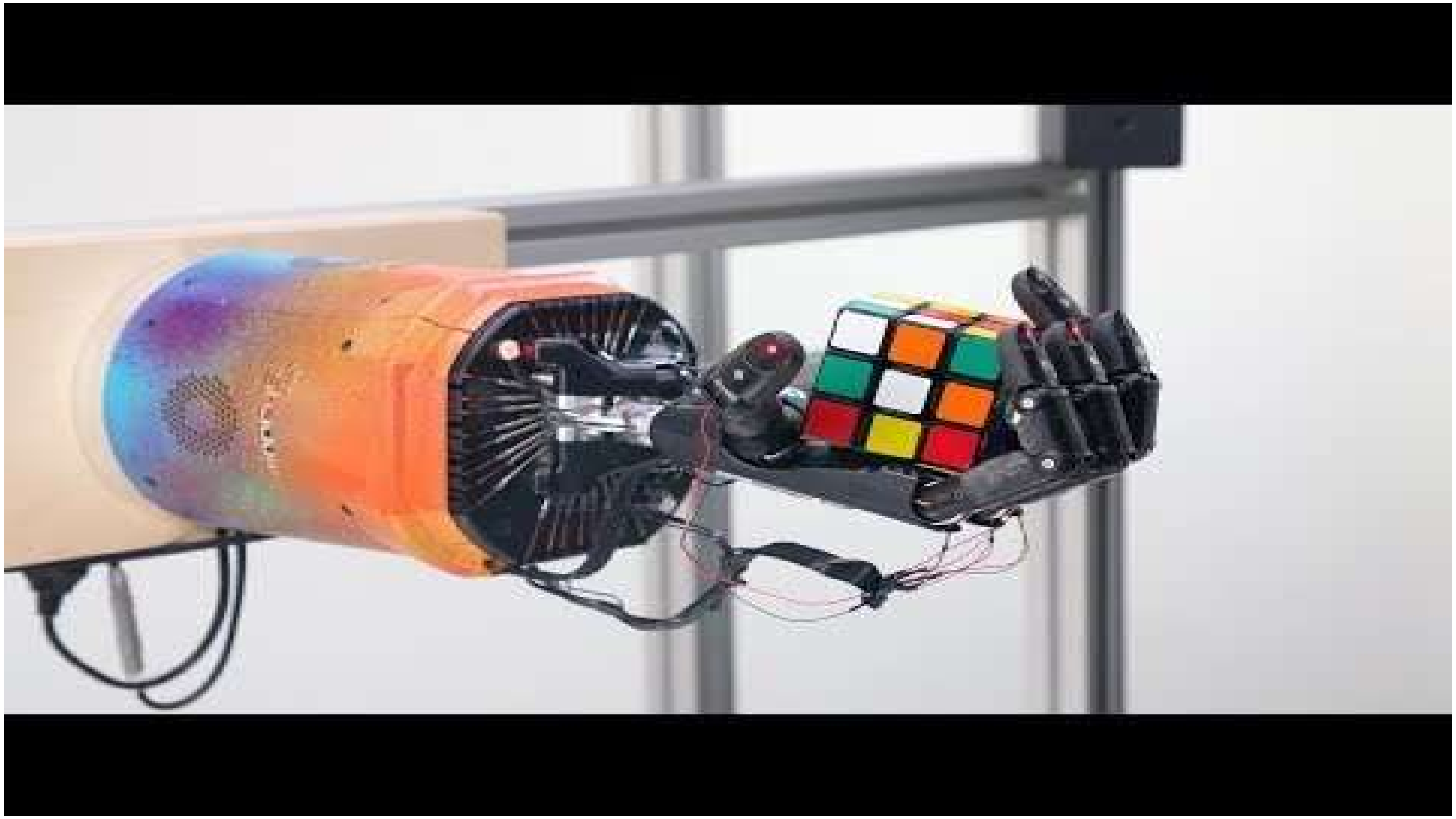
the environment:

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

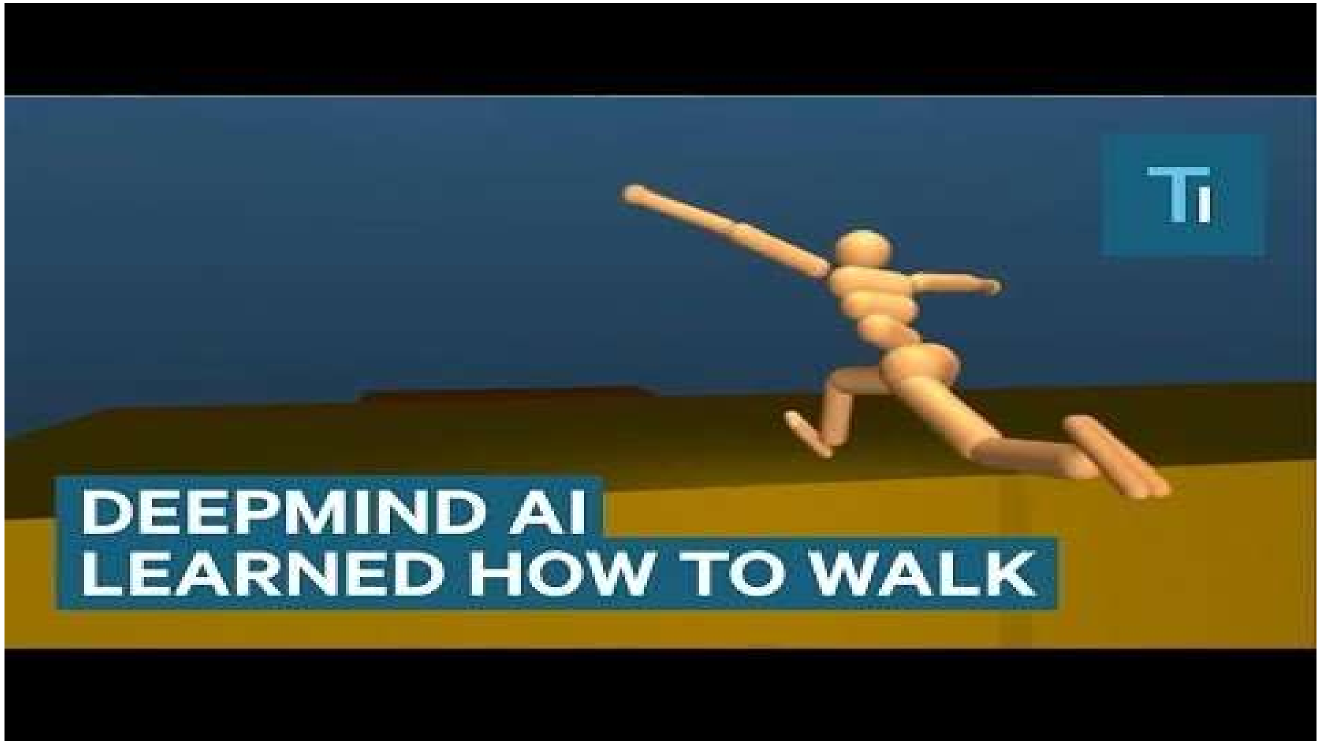
Image Courtesy of Lilian Weng

RL Examples

RL Examples



RL Examples



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,

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

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 Equations

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

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

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

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

Optimal Value and Policy

The optimal value function produces the maximum return:

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

The optimal policy achieves optimal value functions:

$$\pi_* = \arg\max_\pi V_\pi(s), \pi_* = \arg\max_\pi Q_\pi(s,a)$$

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

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

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: $\mathrm{v}_\pi(s)$

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 $\Delta < \theta$

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

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

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

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 $\hat{q}(s, a)$

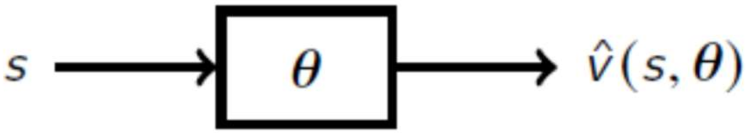
    Take action a, observe r, s'

    $\hat{q}(s, a) \leftarrow \hat{q}(s, a) + \alpha[r + \gamma \max_{a'} \hat{q}(s', a') - \hat{q}(s, a)]$

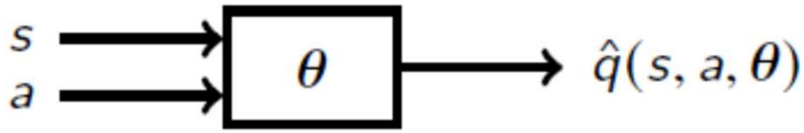
    $s \leftarrow s'$

  until *s is terminal*

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

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

Types of RL Algorithms

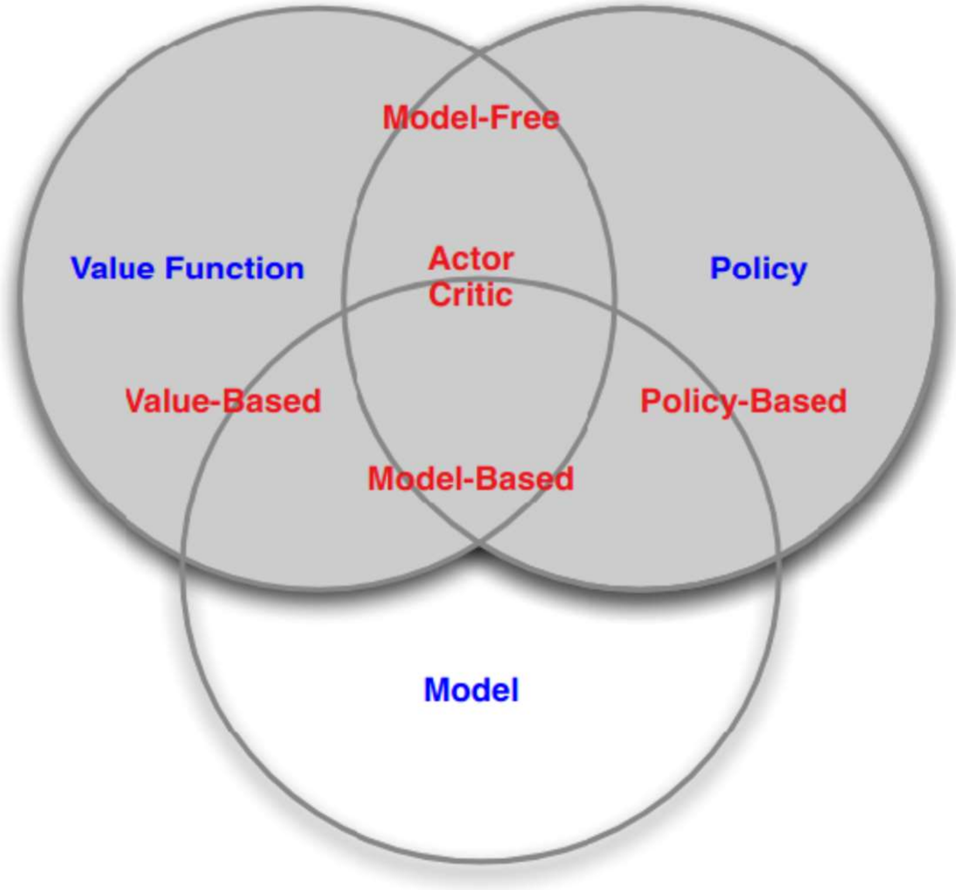


Image Courtesy of David Silver

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

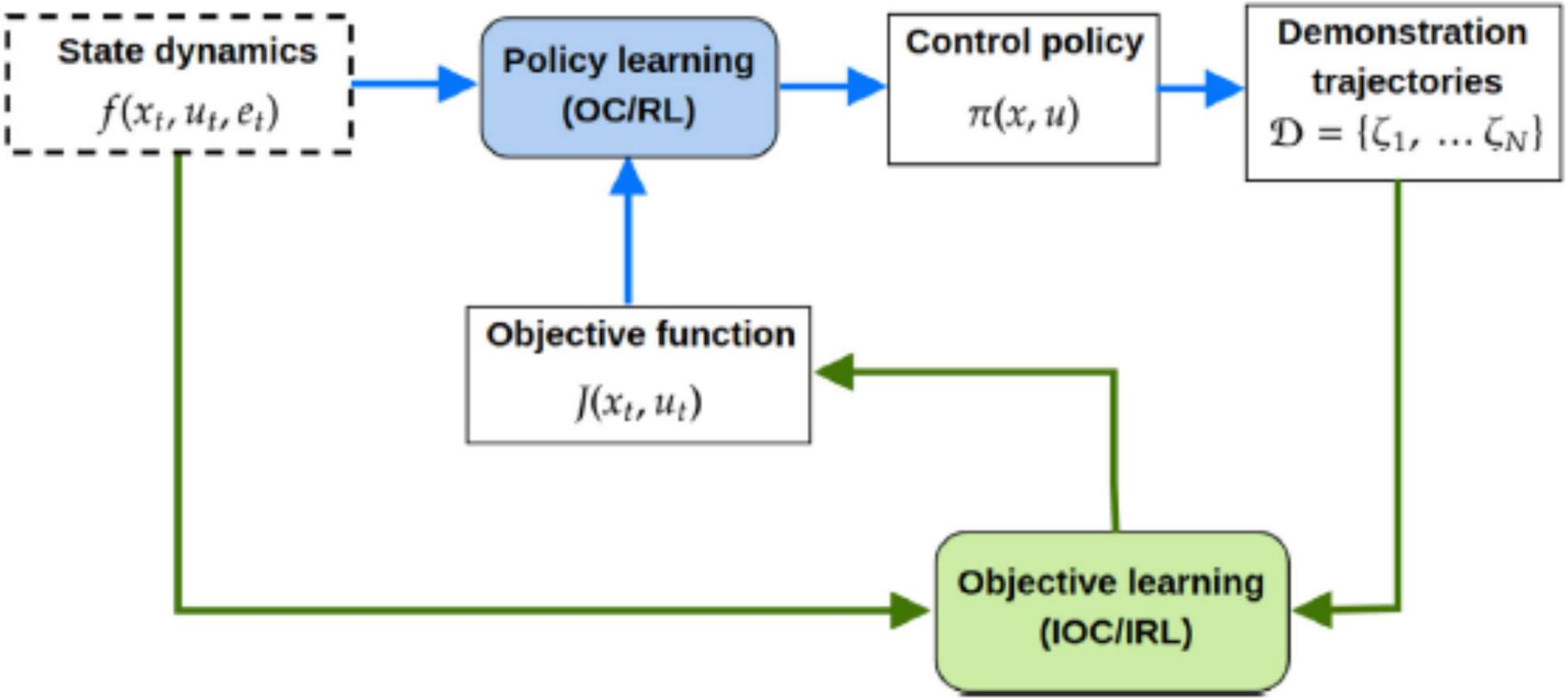
- Exploitation vs. exploration
- Reward Formulation
- “The Deadly Triad”: bootstrapping+off-policy+approximation
- Sim2real gap

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?

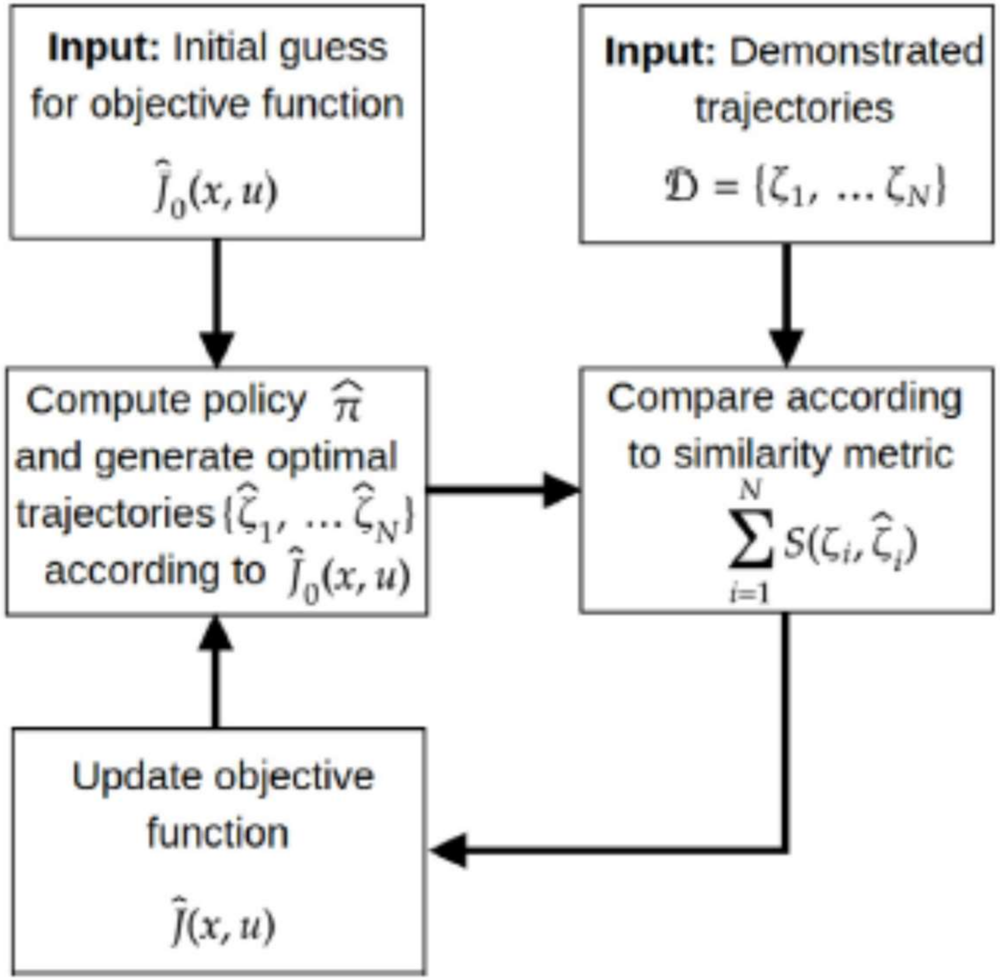
- Two approaches:
  - Inverse Reinforcement Learning (IRL): Use the human demonstrations to learn the reward function, then use RL to learn the policy
  - Behaviour Cloning (BC): Learn the policy directly

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
  - Behavoural 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 – 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?

Summary of Today's lecture

- We can represent sequential decision-making problems as Markov Decision Processes:
  - States, actions, transition function, reward, discount
- Our objective is to find the policy (a mapping from state to action) that maximises reward
- Three approaches:
  - Optimisation
  - Reinforcement Learning
  - Imitation Learning:
    - Inverse RL
    - Behaviour Cloning

Looking ahead to tomorrow’s lecture

- A closer look at behaviour cloning