

# Introduction to Reinforcement Learning

A mini course @ HCMUS, Vietnam

Lectures 1-3

Long Tran-Thanh

[Long.tran-thanh@warwick.ac.uk](mailto:Long.tran-thanh@warwick.ac.uk)

# Details of this mini course

- Reinforcement learning + bandit theory + recent advances in training LLMs
- Topics (wish list):
  - MDP, Bellman equations, dynamic programming (DP)
  - Model-free approaches: MC, TD(0)/TD( $\lambda$ ), SARSA/Q-learning
  - Function approximation + brief intro to deep RL
  - Policy gradient
  - Advanced RL methods (Natural PG, TRPO, PPO, GRPO)
  - Bandits
  - RLHF + applications to LLM training
  - Open problems



Andrew Barto



Richard Sutton



ACM.ORG

**Andrew Barto and Richard Sutton are the recipients of the 2024 ACM A.M. Turing Award for developing the...**

# Aims of this mini course

- RL+ Online learning are very large topics
- We do not aim to cover every aspect of them
- Focus on the technical foundations
  - Less focus on applications (e.g., robotics, video games, healthcare, etc)
  - We don't do numerical exercises
  - Some technical proofs (minimal)
- Source materials:
  - **Sutton & Barto book** (2<sup>nd</sup> ed – 2018)
  - **Mannor, Mansour & Tamar book** (online, not finalised: <https://sites.google.com/view/rlfoundations/home>)
  - Emma Brunskill's RL course at Stanford
  - David Silver's RL course at UCL
  - Nora Kessner's talk on LLM agents
  - My own materials
- More theory: Simons Institute's Theory of RL programme: <https://simons.berkeley.edu/programs/rl20>

# Other details of this course

- Lectures: 14<sup>th</sup> - 17<sup>th</sup> April 2025 (Mon-Tue): 8:30-11:30
- Breaks: 3x 15mins (last one = Q&A) OR 1 big break (30mins) + Q&A after lectures?

# Yann LeCun's cake analogy (NIPS/NeurIPS 2016)

## ■ "Pure" Reinforcement Learning (cherry)

- ▶ The machine predicts a scalar reward given once in a while.
- ▶ **A few bits for some samples**

## ■ Supervised Learning (icing)

- ▶ The machine predicts a category or a few numbers for each input
- ▶ Predicting human-supplied data
- ▶ **10→10,000 bits per sample**

## ■ Unsupervised/Predictive Learning (cake)

- ▶ The machine predicts any part of its input for any observed part.
- ▶ Predicts future frames in videos
- ▶ **Millions of bits per sample**



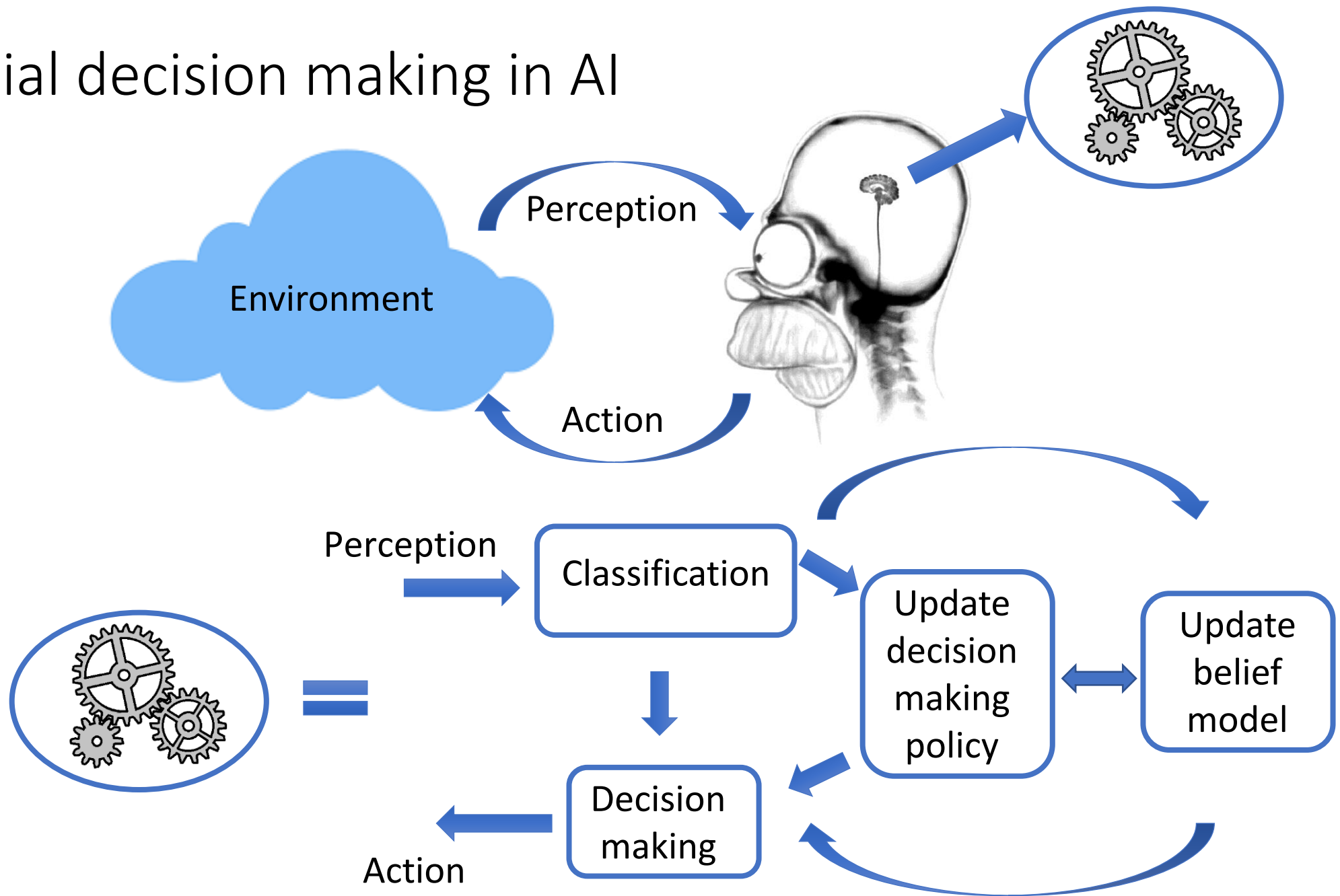
■ (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)

Original LeCun cake analogy slide presented at NIPS 2016, the highlighted area has now been updated.

# My answer to that analogy

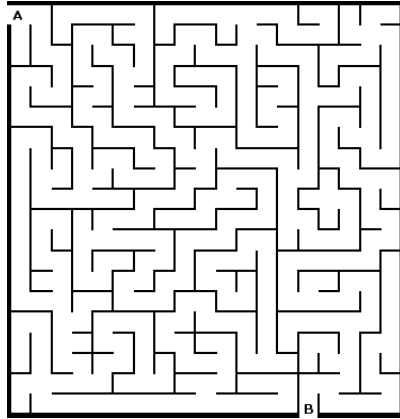
- Only IF AI/ML is passive
- Anything requires interactions, internal state changes, etc, we need LOTS of RL
  - Multi-agent settings
  - Human-AI systems
  - Strategic learnings
  - etc

# Sequential decision making in AI





# Some (real-world) examples



Source: *Dailymail.co.uk*



Source: *Google Research*



<https://towardsdatascience.com/reinforcement-learning-towards-general-ai-1bd68256c72d>

*Draft October 2022, comments to rsutton@ualberta.ca*

## The Alberta Plan for AI Research

**Richard S. Sutton, Michael Bowling, and Patrick M. Pilarski**

University of Alberta  
Alberta Machine Intelligence Institute  
DeepMind Alberta

*History suggests that the road to a firm research consensus is extraordinarily arduous.*

— Thomas Kuhn, *The Structure of Scientific Revolutions*

Herein we describe our approach to artificial intelligence (AI) research, which we call *the Alberta Plan*. The Alberta Plan is pursued within our research groups in Alberta and by others who are like minded throughout the world. We welcome all who would join us in this pursuit.

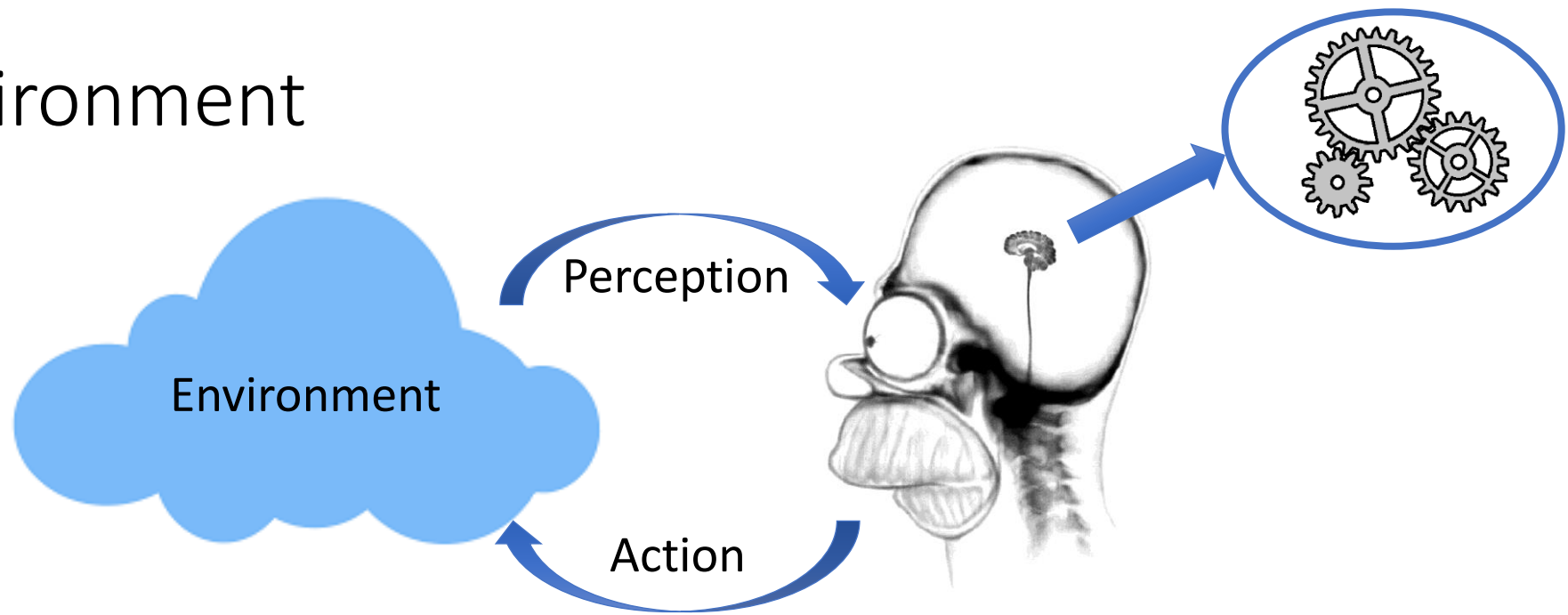
The Alberta Plan is a long-term plan oriented toward basic understanding of computational intelligence. It is a plan for the next 5–10 years. It is not concerned with immediate applications of what we currently know how to do, but rather with filling in the gaps in our current understanding. As computational intelligence comes to be understood it will undoubtedly profoundly affect our economy, our society, and our individual lives. Although all the consequences are difficult to foresee, and every powerful technology contains the potential for abuse, we are convinced that the existence of more far-sighted and complex intelligence will overall be good for the world.

Following the Alberta Plan, we seek to understand and create long-lived computational agents that interact with a vastly more complex world and come to predict and control their sensory input signals. The agents are complex only because they interact with a complex world over a long period of time; their initial design is as simple, general, and scalable as possible.

08.11173v2 [cs.AI] 6 Oct 2022

# Markov Decision Processes

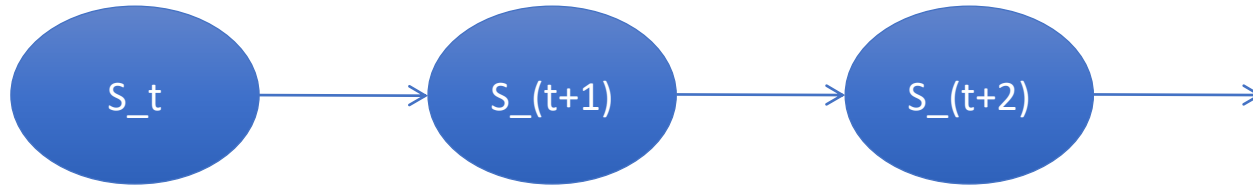
# Agent + environment



## Abstract model:

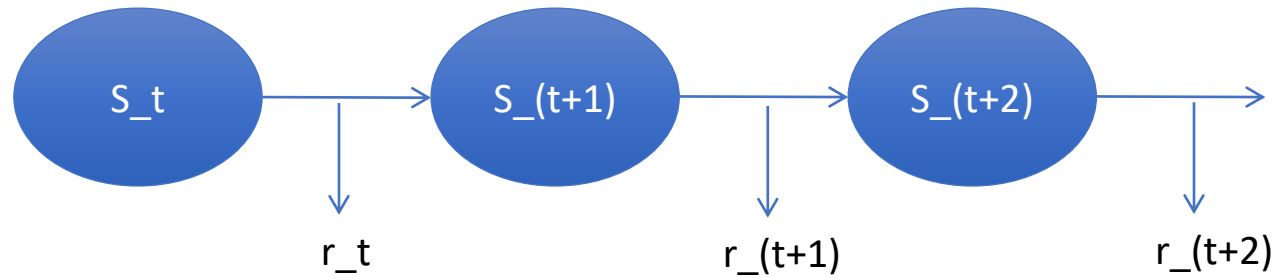
- Agent
- Environment
- Set of decision/action options (action set)
- Repeatedly choose actions (discrete time steps)
- Actions typically change environment's state
- Receive feedback info (new state, usefulness of action, etc)

# Markov process (Markov chain)



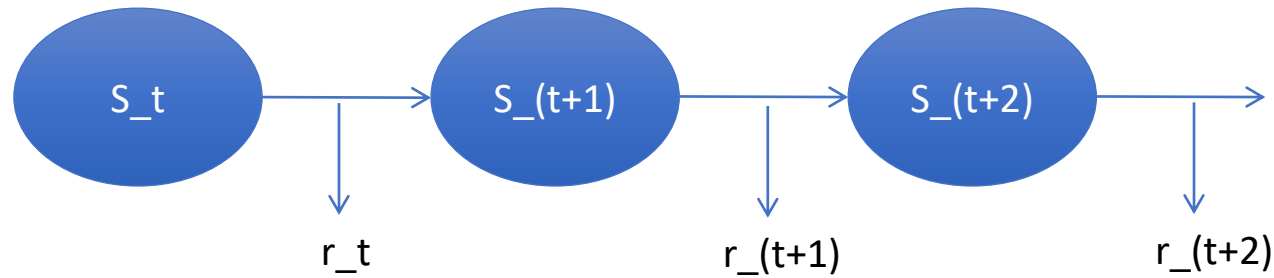
- State:  $s_t \in \mathcal{S}$
- History:  $h_{t-1} = \{s_1, s_2, \dots, s_{t-1}\}$
- Markov property:  $P(s_{t+1} = s' | h_t) = P(s_{t+1} = s' | s_t)$

# Markov reward process (MRP)



- State:  $s_t \in \mathcal{S}$
- History:  $h_{t-1} = \{s_1, s_2, \dots, s_{t-1}\}$
- Markov property:  $P(s_{t+1} = s' | h_t) = P(s_{t+1} = s' | s_t)$
- Receive (random) reward:  $r_t \in \mathbb{R} \quad t = 1, 2, \dots$
- Reward function:  $R(s_t = s) = \mathbb{E}[r_t | s_t = s]$

# Markov reward process (MRP)



- State:  $s_t \in \mathcal{S}$
- History:  $h_{t-1} = \{s_1, s_2, \dots, s_{t-1}\}$
- Markov property:  $P(s_{t+1} = s' | h_t) = P(s_{t+1} = s' | s_t)$
- Receive (random) reward:  $r_t \in \mathbb{R} \quad t = 1, 2, \dots$
- Reward function:  $R(s_t = s) = \mathbb{E}[r_t | s_t = s]$
- Return value:  $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$  where  $\gamma \in [0, 1]$  is a discount factor
- If rewards are bounded  $\rightarrow$  return is also bounded (**we assume it's bounded**)
- State value function:  $V(s) = \mathbb{E}[G_t | s_t = s]$
- Horizon T: can be finite or infinite

# Markov reward process (MRP)

How to calculate  $V(s)$ ?



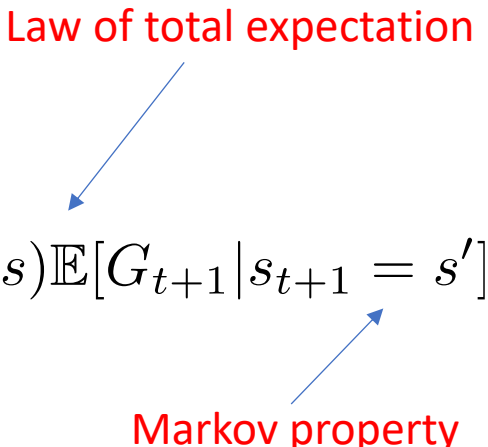
# Markov reward process (MRP)

How to calculate  $V(s)$ ?

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_t + \gamma G_{t+1} | s_t = s] \\ &= \mathbb{E}[r_t | s_t = s] + \gamma \sum_{s' \in \mathcal{S}} P(s_{t+1} = s' | s_t = s) \mathbb{E}[G_{t+1} | s_{t+1} = s'] \end{aligned}$$

Law of total expectation

Markov property



# Markov reward process (MRP)

How to calculate  $V(s)$ ?

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_t + \gamma G_{t+1} | s_t = s] \\ &= \mathbb{E}[r_t | s_t = s] + \gamma \sum_{s' \in \mathcal{S}} P(s_{t+1} = s' | s_t = s) \mathbb{E}[G_{t+1} | s_{t+1} = s'] \\ &= R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s_{t+1} = s' | s_t = s) V(s') \end{aligned}$$

Immediate reward

Discounted sum of future rewards

This is called the Bellman equation

# Markov reward process (MRP)

How to calculate  $V(s)$ ?

$$V(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s_{t+1} = s' | s_t = s) V(s')$$

Vector format:  $V = R + \gamma P V$

Analytical solution:  $V = (I - \gamma P)^{-1} R$

- Quite slow (requires calculation of matrix inverse  $\sim O(|S|^3)$ )
- Inverse might not exist

# Markov reward process (MRP)

Calculating  $V(s)$  with dynamic programming:

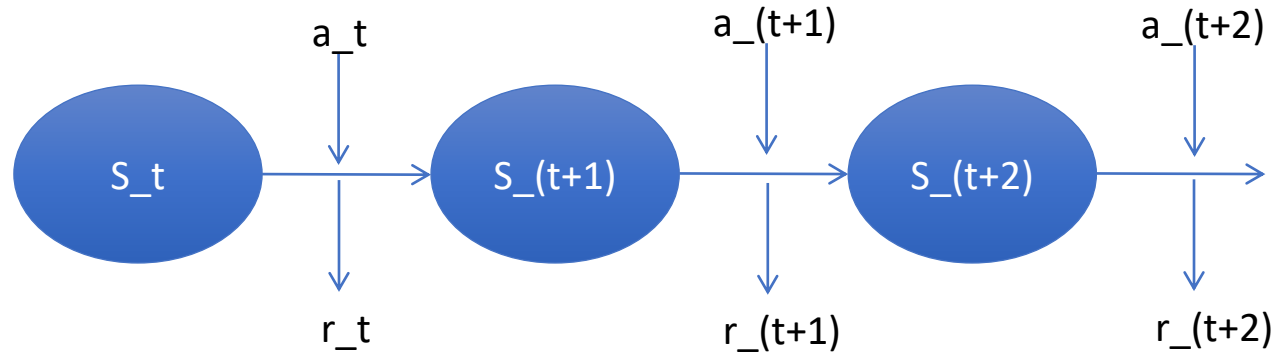
Initialisation:  $\forall s \in \mathcal{S} : V_0(s) = 0$

For  $k = 1$  until convergence do:

$$\forall s \in \mathcal{S} : V_k(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s) V_{k-1}(s')$$

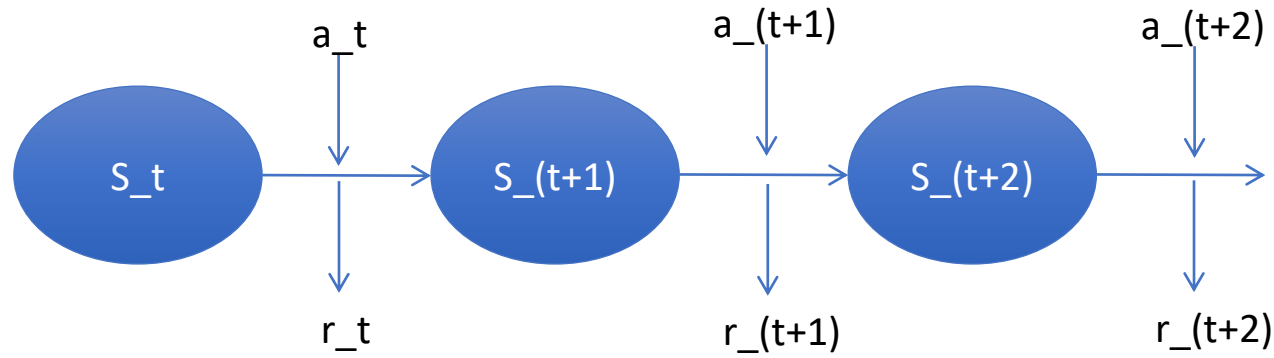
Computational complexity per round:  $O(|\mathcal{S}|^2)$

# Markov decision process (MDP)



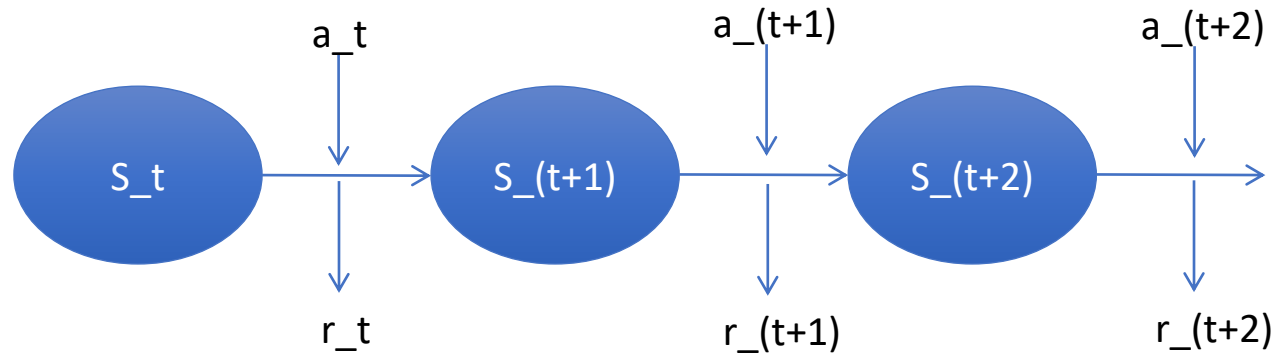
- MDP = MRP + actions
- At each  $t$ : take action  $a_t \in \mathcal{A}$
- Markovian transition probability:  $P(s_{t+1} = s' | s_t = s, a_t = a)$
- Reward function:  $R(s_t = s, a_t = a) = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$
- Discount factor:  $\gamma \in [0, 1]$
- MDP is tuple  $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$

# MDP policies



- A policy specifies which action to take at each state  $s$ 
  - Deterministic vs. stochastic policy
- Formal definition:  $\pi : \mathcal{S} \rightarrow \mathcal{A} \quad P(\pi(s) = a) := P_\pi(a|s)$

# MDP policies

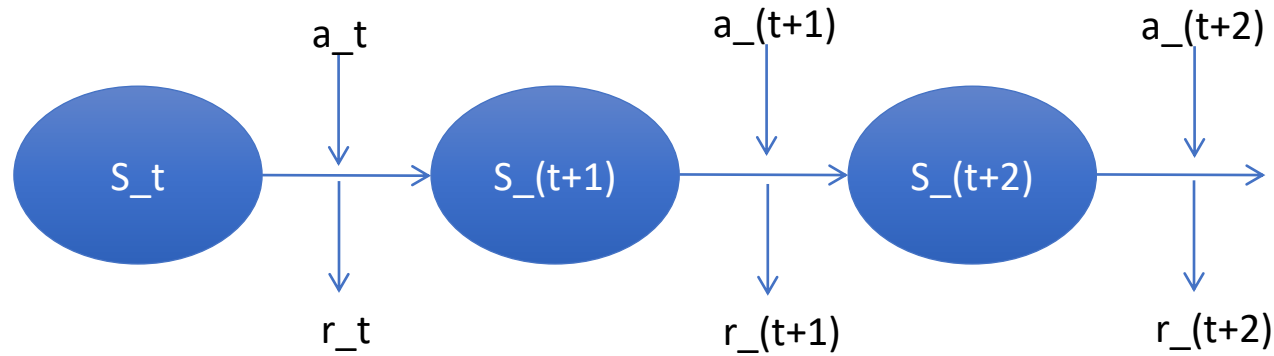


- A policy specifies which action to take at each state  $s$ 
  - Deterministic vs. stochastic policy
- Formal definition:  $\pi : \mathcal{S} \rightarrow \mathcal{A} \quad P(\pi(s) = a) := P_\pi(a|s)$

- MDP + policy  $\pi$  = MRP

- Reward function: 
$$R^\pi(s) = \sum_{a \in \mathcal{A}} P(\pi(s) = a) R(s, a)$$
- State transition probability: 
$$P^\pi(s'|s) = \sum_{a \in \mathcal{A}} P(\pi(s) = a) P(s'|s, a)$$

# MDP policies



- A policy specifies which action to take at each state  $s$ 
  - Deterministic vs. stochastic policy
- Formal definition:  $\pi : \mathcal{S} \rightarrow \mathcal{A} \quad P(\pi(s) = a) := P_\pi(a|s)$

- MDP + policy  $\pi$  = MRP

- Reward function: 
$$R^\pi(s) = \sum_{a \in \mathcal{A}} P(\pi(s) = a) R(s, a)$$
- State transition probability: 
$$P^\pi(s'|s) = \sum_{a \in \mathcal{A}} P(\pi(s) = a) P(s'|s, a)$$

Evaluating the value of a policy in MDPs = computing the value of a MRP



# MDP policy value evaluation

Calculating  $V(s)$  with dynamic programming:

Initialisation:  $\forall s \in \mathcal{S} : V_0(s) = 0$

For  $k = 1$  until convergence do:

$$\forall s \in \mathcal{S} : V_k^\pi(s) = R^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} P^\pi(s'|s) V_{k-1}^\pi(s')$$

# MDP policy value evaluation

Calculating  $V(s)$  with dynamic programming:

Initialisation:  $\forall s \in \mathcal{S} : V_0(s) = 0$

For  $k = 1$  until convergence do:

$$\forall s \in \mathcal{S} : V_k^\pi(s) = R^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} P^\pi(s'|s) V_{k-1}^\pi(s')$$

This is called the **Bellman backup/update operator** for a particular policy

$$V_k^\pi = B(V_{k-1}^\pi) = R^\pi + \gamma P^\pi V_{k-1}^\pi$$

# MDP policy value evaluation

Why dynamic programming solution will converge?

**Bellman backup/update operator** is a contraction operator:

$$V_k^\pi = B(V_{k-1}^\pi) = R^\pi + \gamma P^\pi V_{k-1}^\pi$$

It's easy to show that

$$\|B(V) - B(V')\| \leq \gamma \|V - V'\|$$

This implies that for rounds (k-1), k, and (k+1) in DP:

$$\|V_{k+1}^\pi - V_k^\pi\| \leq \gamma \|V_k^\pi - V_{k-1}^\pi\|$$

This will converge to 0 if  $\gamma < 1 \rightarrow$  DP converges in finite iterations (bounded sequence with decreasing differences)

# From evaluating a policy to computing the optimal policy

Optimal policy:  $\pi^*(s) = \arg \max_{\pi} V^{\pi}(s)$

**Claim 1:** If there exists an optimal policy with optimal  $V^*$ , then there also exists a deterministic policy with the same  $V^*$  state value function

**Claim 2:** There exists such optimal policy

# From evaluating a policy to computing the optimal policy

Optimal policy:  $\pi^*(s) = \arg \max_{\pi} V^{\pi}(s)$

**Claim 1:** If there exists an optimal policy with optimal  $V^*$ , then there also exists a deterministic policy with the same  $V^*$  state value function

**Claim 2:** There exists such optimal policy

## Proof sketch of Claim1:

- Any randomised policy = linear combination of deterministic policies. In fact, it lies within the convex hull of those deterministic policies
- If a randomised policy is optimal, so at least one of its deterministic supports

# From evaluating a policy to computing the optimal policy

Optimal policy:  $\pi^*(s) = \arg \max_{\pi} V^{\pi}(s)$

**Claim 1:** If there exists an optimal policy with optimal  $V^*$ , then there also exists a deterministic policy with the same  $V^*$  state value function

**Claim 2:** There exists such optimal policy

Proof sketch of Claim1:

- Any randomised policy = linear combination of deterministic policies. In fact, it lies within the convex hull of those deterministic policies
- If a randomised policy is optimal, so at least one of its deterministic supports

**Proof sketch of Claim 2:**

- There are finite number of deterministic policies (hence argmax does exist)

**Homework (optional):** Proof these claims in a technically rigorous way

# Computing the optimal policy

Key implication: we can just focus on deterministic policies

# Computing the optimal policy

Key implication: we can just focus on deterministic policies

However, the set of these policies is exponentially large:  $|\mathcal{A}|^{|\mathcal{S}|}$

Therefore we need better solution than exhaustive search (enumeration)



# Policy iteration (PI)

We focus on deterministic policies

- Set  $i = 0$
- Initialise  $\pi_i(s) = \pi_0(s)$  randomly for all state  $s$
- While ( $i = 0$  or  $\|\pi_i - \pi_{i-1}\|_1 > 0$ ) do:
  - Evaluate value function of  $\pi_i$
  - Improve policy into  $\pi_{i+1}$
  - $i = i + 1$

# Policy iteration (PI)

We focus on deterministic policies

- Set  $i = 0$
- Initialise  $\pi_i(s) = \pi_0(s)$  randomly for all state  $s$
- While ( $i = 0$  or  $\|\pi_i - \pi_{i-1}\|_1 > 0$ ) do:
  - Evaluate value function of  $\pi_i$
  - Improve policy into  $\pi_{i+1}$
  - $i = i + 1$

# State-action value (Q value)

Consider the value of a state-action pair:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi}(s')$$

# State-action value (Q value)

Consider the value of a state-action pair:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi}(s')$$

Meaning: the value of **taking action  $a$  now, then follow policy  $\pi$  in the future**

# Back to PI

Compute the state-action value of policy  $\pi_i$  for each (s,a) pair :

$$Q^{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi_i}(s')$$

Now, the policy improvement step:

$$\forall s \in \mathcal{S} : \pi_{i+1}(s) = \arg \max_{a \in \mathcal{A}} Q^{\pi_i}(s, a)$$

# Back to PI

Compute the state-action value of policy  $\pi_i$  for each (s,a) pair :

$$Q^{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi_i}(s')$$

Now, the policy improvement step:

$$\forall s \in \mathcal{S} : \pi_{i+1}(s) = \arg \max_{a \in \mathcal{A}} Q^{\pi_i}(s, a)$$

**Statement:** this policy iteration method will find the optimal policy in finite number of iterations

# Back to PI

Compute the state-action value of policy  $\pi_i$  for each (s,a) pair :

$$Q^{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi_i}(s')$$

Now, the policy improvement step:

$$\forall s \in \mathcal{S} : \pi_{i+1}(s) = \arg \max_{a \in \mathcal{A}} Q^{\pi_i}(s, a)$$

**Statement:** this policy iteration method will find the optimal policy in finite number of iterations

**Proof sketch:**

- The new policy is also deterministic (max\_a is deterministic)
- At least as good as the previous policy (use some simple algebra)
- Finite number of deterministic policies -> will stop changing (recall Claim 2 on slide 26)
- Once it converges, the final policy is optimal (proof by contradiction)

**Homework 2 (optional): Proof these claims in the proof sketch**

Hint: At least as good as the previous policy – taken from E. Brunskill's lecture slides (Stanford)

$$\begin{aligned} V^{\pi_i}(s) &\leq \max_a Q^{\pi_i}(s, a) \\ &= \max_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi_i}(s') \\ &= R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi_{i+1}(s)) V^{\pi_i}(s') \quad // \text{by the definition of } \pi_{i+1} \\ &\leq R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi_{i+1}(s)) \left( \max_{a'} Q^{\pi_i}(s', a') \right) \\ &= R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi_{i+1}(s)) \\ &\quad \left( R(s', \pi_{i+1}(s')) + \gamma \sum_{s'' \in \mathcal{S}} P(s''|s', \pi_{i+1}(s')) V^{\pi_i}(s'') \right) \\ &\quad \vdots \\ &= V^{\pi_{i+1}}(s) \end{aligned}$$



# Computing the optimal value function

- Value iteration (high level idea):
  - Consider episodes of finite time horizon  $H$
  - Maintain optimal value of starting in a state  $s$  with  $k$  time steps left until  $H$  (episode)
  - Iterate to consider longer and longer episodes until  $k = H$

# Computing the optimal value function

- Value iteration (high level idea):
  - Consider episodes of finite time horizon  $H$
  - Maintain optimal value of starting in a state  $s$  with  $k$  time steps left until  $H$  (episode)
  - Iterate to consider longer and longer episodes until  $k = H$
- Recall that a value function of a policy must satisfy the Bellman equation:

$$\forall s \in \mathcal{S} : V^\pi(s) = R^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} P^\pi V^\pi(s')$$

# Computing the optimal value function

- Value iteration (high level idea):
  - Consider episodes of finite time horizon  $H$
  - Maintain optimal value of starting in a state  $s$  with  $k$  time steps left until  $H$  (episode)
  - Iterate to consider longer and longer episodes until  $k = H$
- Recall that a value function of a policy must satisfy the Bellman equation:

$$\forall s \in \mathcal{S} : V^\pi(s) = R^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} P^\pi V^\pi(s')$$

- Bellman equation for the optimal value function:

$$\forall s \in \mathcal{S} : V^*(s) = \max_{a \in \mathcal{A}} \left( R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^*(s') \right)$$

# Dynamic programming for value iteration

- Optimal value function up to  $k$  more time steps:  $V_k$
- Optimal policy up to  $k$  more time steps:  $\pi_k$
- Initialise  $\forall s \in \mathcal{S} : V_0(s) = 0$
- For  $k = 1$  to  $H$  ( $H$  is the final horizon)

$$\forall s \in \mathcal{S} : V_{k+1}(s) = \max_{a \in \mathcal{A}} \left( R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_k(s') \right)$$

$$\forall s \in \mathcal{S} : \pi_{k+1}(s) = \arg \max_{a \in \mathcal{A}} \left( R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_k(s') \right)$$

# Value iteration vs. Policy iteration

- Value iteration:
  - Compute optimal value for horizon  $k \leq H$
  - Note this can be used to compute optimal policy of horizon  $k \leq H$
  - Increment  $k$  until  $H$
- Policy iteration
  - Compute infinite horizon value of a policy
  - Use to select another (better) policy
  - Closely related to a very popular method in RL: policy gradient (later)