COSC 4368
Fundamentals of Artificial Intelligence

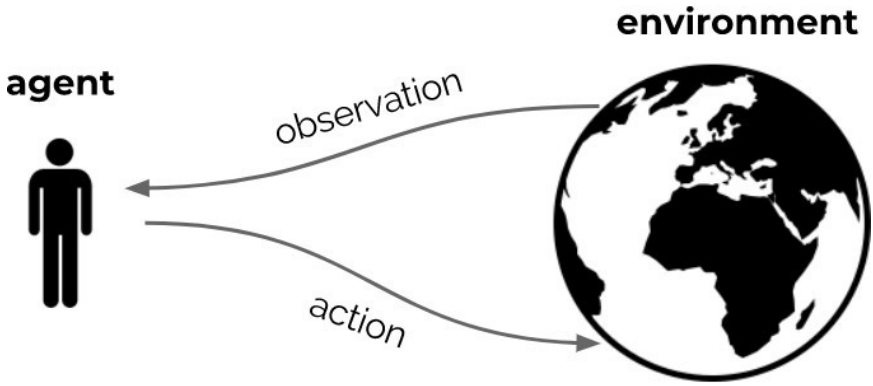
Reinforcement Learning
October 31st, 2023
(Slides from DeepMind)

What is Reinforcement Learning?

What is reinforcement learning?

- People and animals learn by **interacting with our environment**
- This differs from certain other types of learning
  - It is **active** rather than passive
  - Interactions are often **sequential** — future interactions can depend on earlier ones
- We are **goal-directed**
- We can learn **without examples** of optimal behaviour
- Instead, we optimise some **reward signal**

The interaction loop



Goal: optimise sum of rewards, through repeated interaction

The reward hypothesis

Reinforcement learning is based on the **reward hypothesis**:

> *Any goal can be formalized as the outcome of maximizing a cumulative reward*

Examples of RL problems

- Fly a helicopter → **Reward**: air time, inverse distance, ...
- Manage an investment portfolio → **Reward**: gains, gains minus risk, ...
- Control a power station → **Reward**: efficiency, ...
- Make a robot walk → **Reward**: distance, speed, ...
- Play video or board games → **Reward**: win, maximise score, ...

If the goal is to learn via interaction, these are all reinforcement learning problems (Irrespective of which solution you use)

What is reinforcement learning?

There are distinct reasons to learn:

1. Find **solutions**
   - A program that plays chess really well
   - A manufacturing robot with a specific purpose
2. **Adapt online**, deal with unforeseen circumstances
   - A chess program that can learn to adapt to you
   - A robot that can learn to navigate unknown terrains

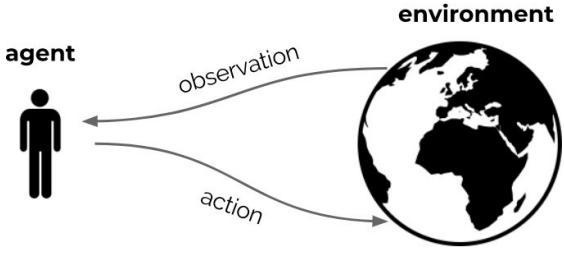
- Reinforcement learning can provide algorithms for both cases
- Note that the second point is not (just) about generalization — it is about continuing to learn efficiently online, during operation

What is reinforcement learning?

- Science and framework of **learning to make decisions** from **interaction**
- This requires us to think about
  - ...time
  - ...(long-term) consequences of actions
  - ...actively gathering experience
  - ...predicting the future
  - ...dealing with uncertainty
- Huge potential scope
- A formalisation of the AI problem

Formalizing the RL Problem

Agent and Environment



- At each step t the agent:
  - Receives observation O_t (and reward R_t)
  - Executes action A_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1} (and reward

Rewards

- A **reward** R_t is a scalar feedback signal
- Indicates how well agent is doing at step t — defines the goal
- The agent's job is to maximize cumulative reward

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots$$

- We call this the **return**

Reinforcement learning is based on the **reward hypothesis**:

> *Any goal can be formalized as the outcome of maximizing a cumulative reward*

Value
s

- We call the expected cumulative reward, from a state s, the **value**

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$
$$= \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3} + \ldots \mid S_t = s]$$

- The value depends on the actions the agent takes
- Goal is to **maximize value**, by picking suitable actions
- Rewards and values define **utility** of states and action (no supervised feedback)
- Returns and values can be defined recursively

$$G_t = R_{t+1} + G_{t+1}$$
$$v(s) = \mathbb{E}[R_{t+1} + v(S_{t+1}) \mid S_t = s]$$

Maximising value by taking actions

- Goal: **select actions to maximise value**
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
  - Refueling a helicopter (might prevent a crash in several hours)
  - Defensive moves in a game (may help chances of winning later)
  - Learning a new skill (can be costly & time-consuming at first)
- A mapping from states to actions is called a **policy**

Action values

- It is also possible to condition the value on **actions**:

$$
\begin{aligned}
q(s, a) &= \mathsf{E}\left[G_t \mid S_t = s, A_t = a\right] \\
&= \mathsf{E}\left[R_{t+1} + R_{t+2} + R_{t+3} + \ldots \mid S_t = s, A_t = a\right]
\end{aligned}
$$

- We will talk in depth about state and action values later

Core concepts

The reinforcement learning formalism includes

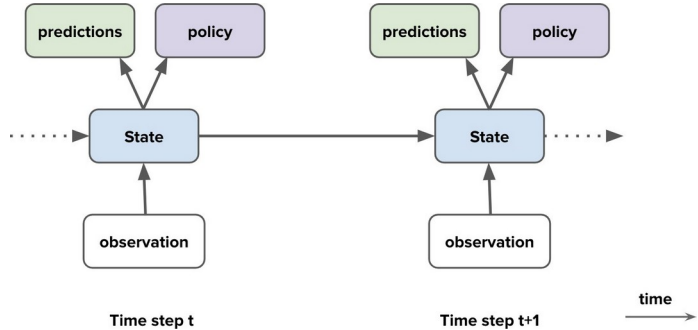
- **Environment** (dynamics of the problem)
- **Reward** signal (specifies the goal)
- **Agent**, containing:
  - Agent state
  - Policy
  - Value function estimate?
  - Model?
- We will now go into the agent

Inside the Agent: the Agent State

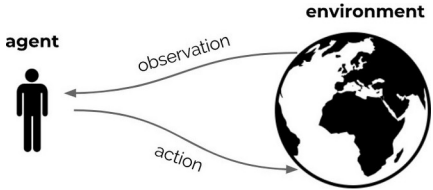
Agent components

Agent components

- **Agent state**
- Policy
- Value functions
- Model



Environment State



- The **environment state** is the environment's internal state
- It is usually invisible to the agent
- Even if it is visible, it may contain lots of irrelevant information

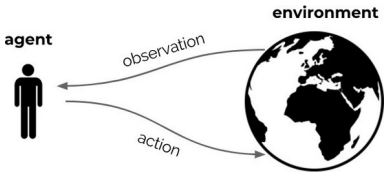
Agent State

- The **history** is the full sequence of observations, actions, rewards

$$H_t = O_0, A_0, R_1, O_1, ..., O_{t-1}, A_{t-1}, R_t, O_t$$

- For instance, the sensorimotor stream of a robot
- This history is used to construct the **agent state** S_t

Fully Observable Environments



Full observability
Suppose the agent sees the full environment state

- observation = environment state
- The agent state could just be this observation:

$$S_t = O_t = \text{environment state}$$

Markov decision processes

Markov decision processes (MDPs) are a useful mathematical framework

Definition

A decision process is Markov if

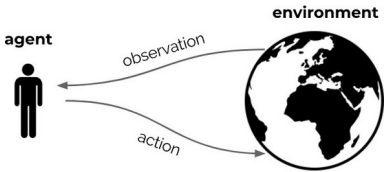
$$p(r, s \mid S_t, A_t) = p(r, s \mid H_t, A_t)$$

- This means that the state contains all we need to know from the history
- Doesn't mean it contains everything, just that adding more history doesn't help
- $\Longrightarrow$ Once the state is known, the history may be thrown away
  - The full environment + agent state is Markov (but large)
  - The full history H_t is Markov (but keeps growing)
- Typically, the agent state S_t is some compression of H_t
- Note: we use S_t to denote the **agent state**, not the **environment**

Partially Observable Environments

- **Partial observability**: The observations are not Markovian
  - A robot with camera vision isn't told its absolute location
  - A poker playing agent only observes public cards
- Now using the observation as state would not be Markovian
- This is called a **partially observable Markov decision process** (POMDP)
- The **environment state** can still be Markov, but the agent does not know it
- We might still be able to construct a Markov agent state

Agent State



- The agent's actions depend on its state
- The **agent state** is a function of the history
- For instance, $S_t = O_t$
- More generally:

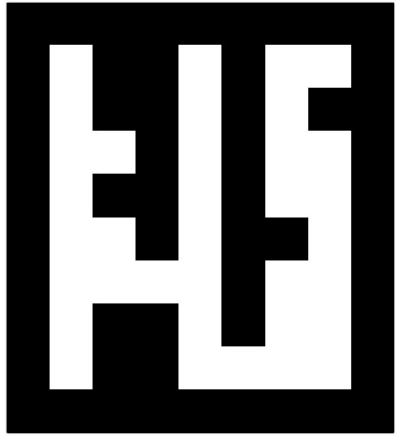
$$S_{t+1} = u(S_t, A_t, R_{t+1}, O_{t+1})$$

where u is a 'state update function'

- The agent state is often **much** smaller than the environment state

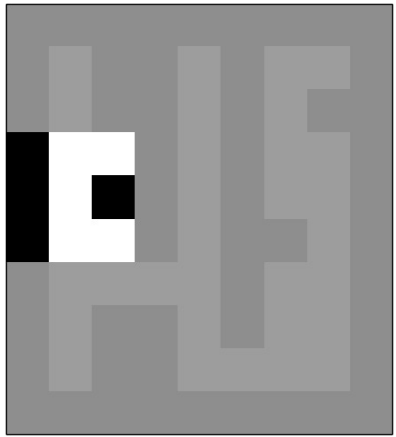
Agent State

The full environment state of a



Agent State

A potential observation

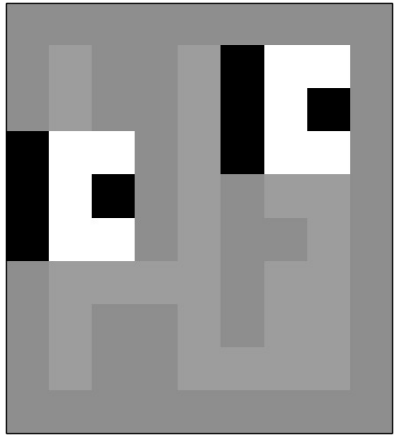


Agent State

An observation in a different

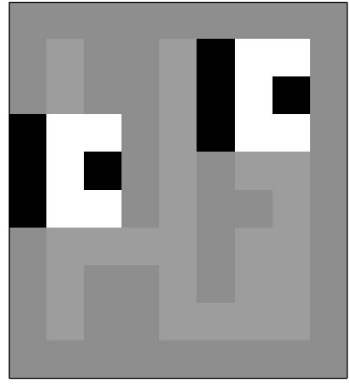
Agent State

The two observations are indistinguishable



Agent State

These two states are not



How could you construct a Markov agent state in this maze (for any reward signal)?

Partially Observable Environments

- To deal with partial observability, agent can construct suitable state representations
- Examples of agent states:
  - Last observation: $S_t = O_t$ (might not be enough)
  - Complete history: $S_t = H_t$ (might be too large)
  - A generic update: $S_t = u(S_{t-1}, A_{t-1}, R_t, O_t)$ (but how to pick/learn u?)
- Constructing a fully Markovian agent state is often not feasible
- More importantly, the state should allow good policies and value predictions

Inside the Agent: the Policy

Agent components

Agent components

- Agent state
- **Policy**
- Value function
- Model

Policy

- A **policy** defines the agent's behaviour
- It is a map from agent state to action
- Deterministic policy: $A = \pi(S)$
- Stochastic policy: $\pi(A|S) = p(A|S)$

Inside the Agent: Value Estimates

Agent components

Agent components

- Agent state
- Policy
- **Value function**
- Model

Value Function

- The actual value function is the expected return

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}\left[G_t \mid S_t = s, \pi\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s, \pi\right]
\end{aligned}
$$

- We introduced a **discount factor** $\gamma \in [0, 1]$
  - Trades off importance of immediate vs long-term rewards
- The value depends on a policy
- Can be used to evaluate the desirability of states
- Can be used to select between actions

Value Functions

- The return has a recursive form $G_t = R_{t+1} + \gamma G_{t+1}$
- Therefore, the value has as well

$$\begin{aligned} v_\pi(s) &= \mathsf{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t \sim \pi(s)\right] \\ &= \mathsf{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi(s)\right] \end{aligned}$$

  Here $a \sim \pi(s)$ means a is chosen by policy π in state s (even if π is deterministic)
- This is known as a **Bellman equation** (Bellman 1957)
- A similar equation holds for the optimal (=highest possible) value:

$$v_*(s) = \max_a \mathsf{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\right]$$

  This does **not** depend on a policy
- We heavily exploit such equalities, and use them to create algorithms

Value Function approximations

- Agents often approximate value functions
- We will discuss algorithms to learn these efficiently
- With an accurate value function, we can behave optimally
- With suitable approximations, we can behave well, even in intractably big domains

Inside the Agent: Models

Agent components

Agent components

- Agent state
- Policy
- Value function
- **Model**

Mode
l

- A **model** predicts what the environment will do next
- E.g., P predicts the next state

$$\mathsf{P}(s, a, s^{1}) \approx p\left(S_{t+1} = s^{1} \mid S_t = s, A_t = a\right)$$

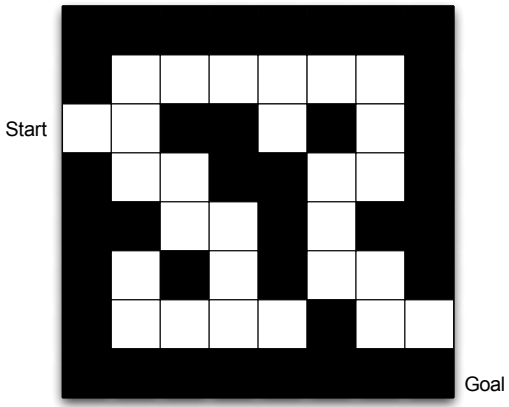
- E.g., R predicts the next (immediate) reward

$$\mathsf{R}(s, a) \approx \mathsf{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

- A model does not immediately give us a good policy - we would still need to plan
- We could also consider **stochastic** (**generative**) models

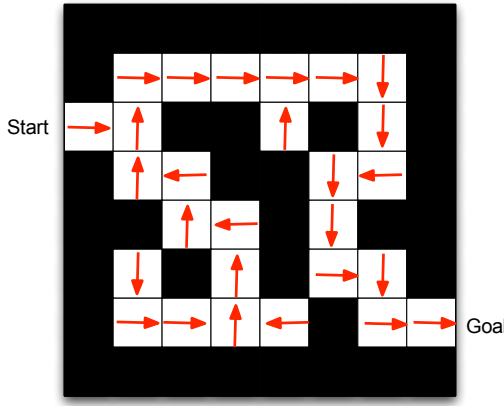
An Example

Maze Example



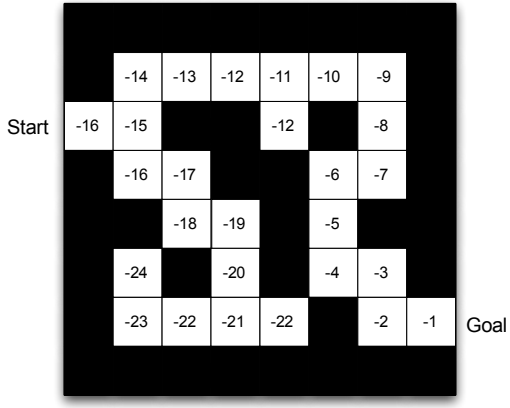
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example: Policy



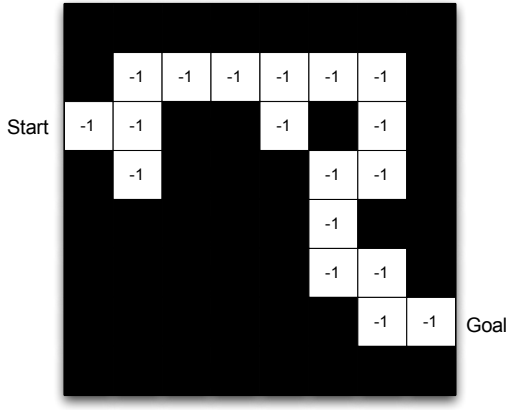
- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function



- Numbers represent value $v_\pi(s)$ of each state s

Maze Example: Model



- Grid layout represents partial transition model $P^a_{ss'}$
- Numbers represent immediate reward $R^a_{ss'}$ = -1 from each state s (same for all a and s' in this case)

Agent Categories

Agent Categories

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function
- Actor Critic
  - Policy
  - Value Function

Agent Categories

- Model Free
  - Policy and/or Value Function
  - No Model
- Model Based
  - Optionally Policy and/or Value Function
  - Model

Subproblems of the RL Problem

Prediction and Control

- **Prediction**: evaluate the future (for a given policy)
- **Control**: optimise the future (find the best policy)
- These can be strongly related:

$$\pi_*(s) = \underset{\pi}{\text{argmax}}\, v_\pi(s)$$

- If we could predict **everything** do we need anything else?

Learning and Planning

Two fundamental problems in reinforcement learning

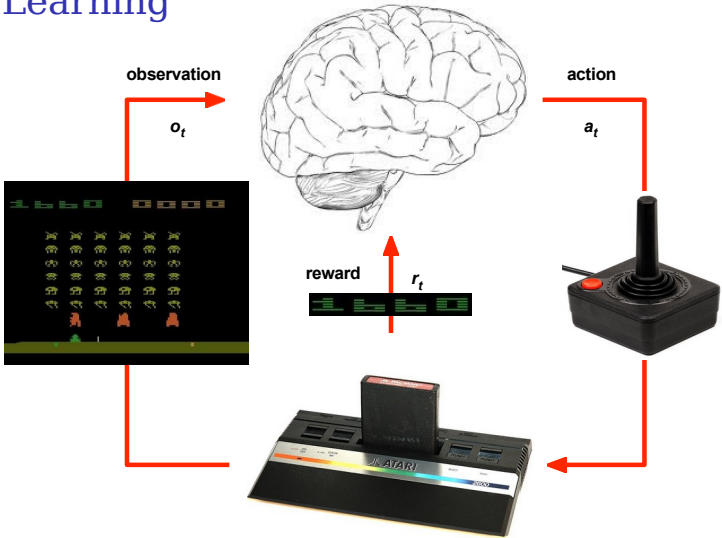
- Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
- Planning:
  - A model of the environment is given (or learnt)
  - The agent plans in this model (without external interaction)
  - a.k.a. reasoning, pondering, thought, search, planning

Learning Agent Components

- All components are functions
  - Policies: $\pi : S \to A$ (or to probabilities over A)
  - Value functions: $v : S \to R$
  - Models: $m : S \to S$ and/or $r : S \to R$
  - State update: $u : S \times O \to S$
- E.g., we can use neural networks, and use **deep learning** techniques to learn
- Take care: we do often violate assumptions from supervised learning (iid, stationarity)
- Deep learning is an important tool
- Deep reinforcement learning is a rich and active research field

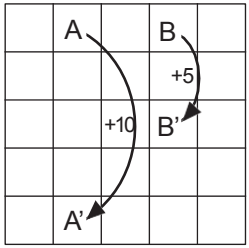
Examples

Atari Example: Reinforcement Learning

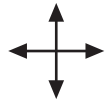


- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Gridworld Example: Prediction



(a)



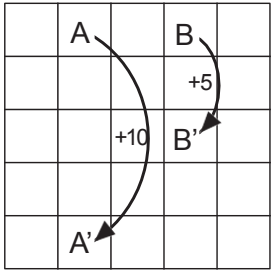
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

Reward is -1 when bumping into a wall, $\gamma = 0.9$

What is the value function for the uniform random policy?

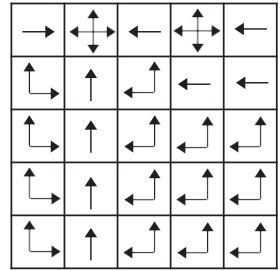
Gridworld Example: Control



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*



c) π^*

What is the optimal value function over all possible policies? What is the optimal policy?