Reinforcement learning @ UCL 2020

Lecture 1: Introduction

Hado van Hasselt
Senior Staff Research Scientist @ DeepMind

Admin

- RL lectures: Tuesdays 2-4pm, Thursday 9-11am
- Check Moodle for updates
- Use Moodle for questions
- Grading: assignments (50%) + exam (50%)
- Reading material:
  - Slides
  - Reinforcement Learning: An Introduction, Sutton & Barto 2018
    http://incompleteideas.net/book/the-book-2nd.html
- Background material: Sutton & Barto, Ch. 1 and Ch. 3

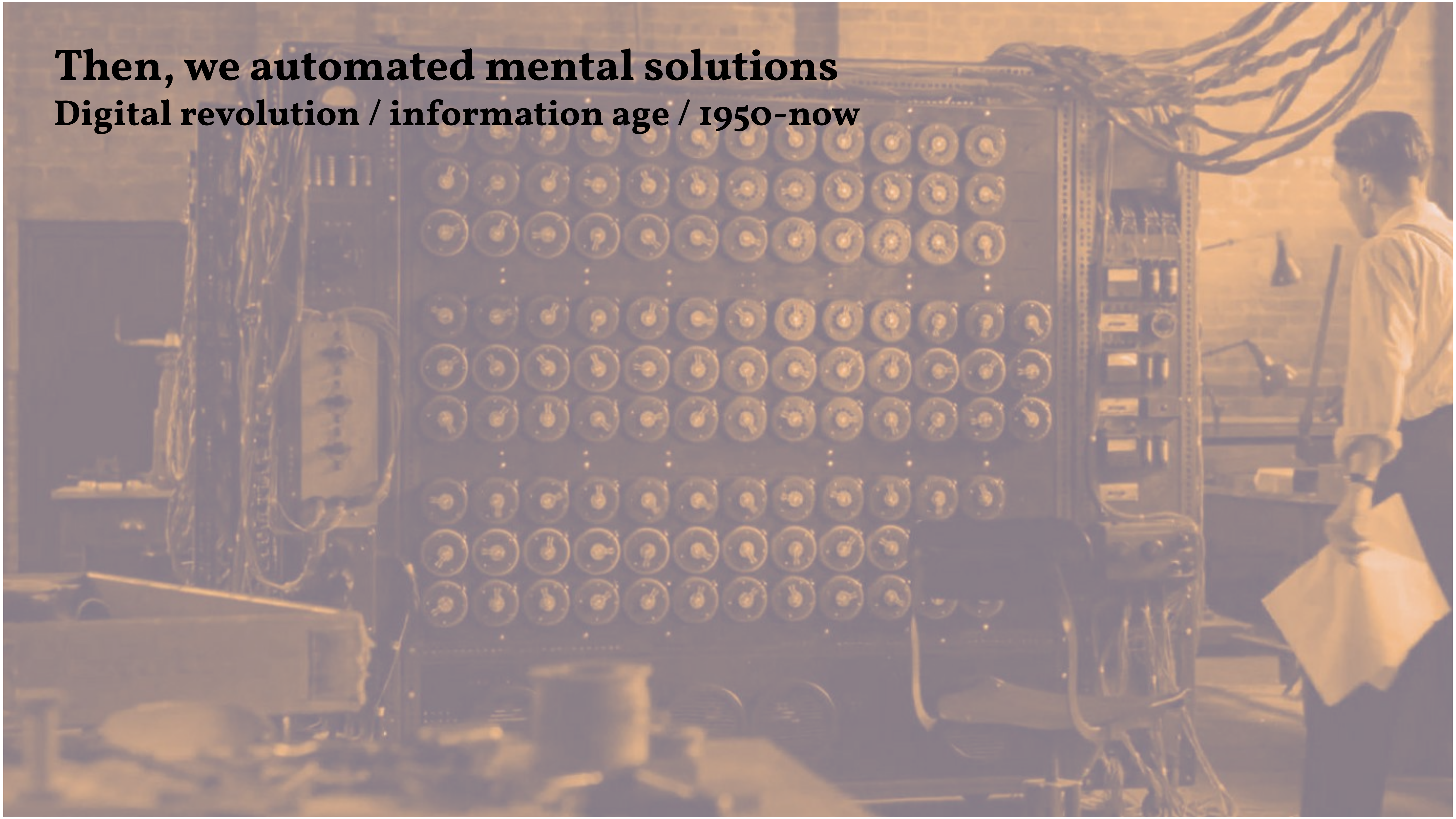
Artificial intelligence

What is AI?


First, we automated physical solutions
Industrial revolution / machine age / 1750-1940

Then, we automated mental solutions

Digital revolution / information age / 1950-now



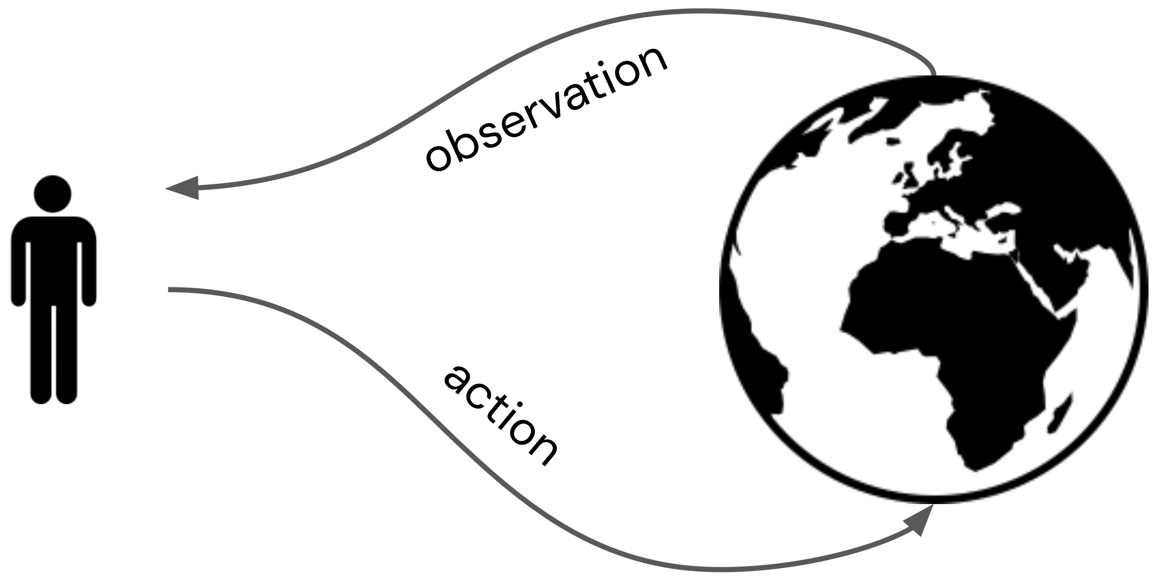
We still had to come up with solutions

Next steps:

1. Learn solutions from experience
2. Learn to find relevant experience

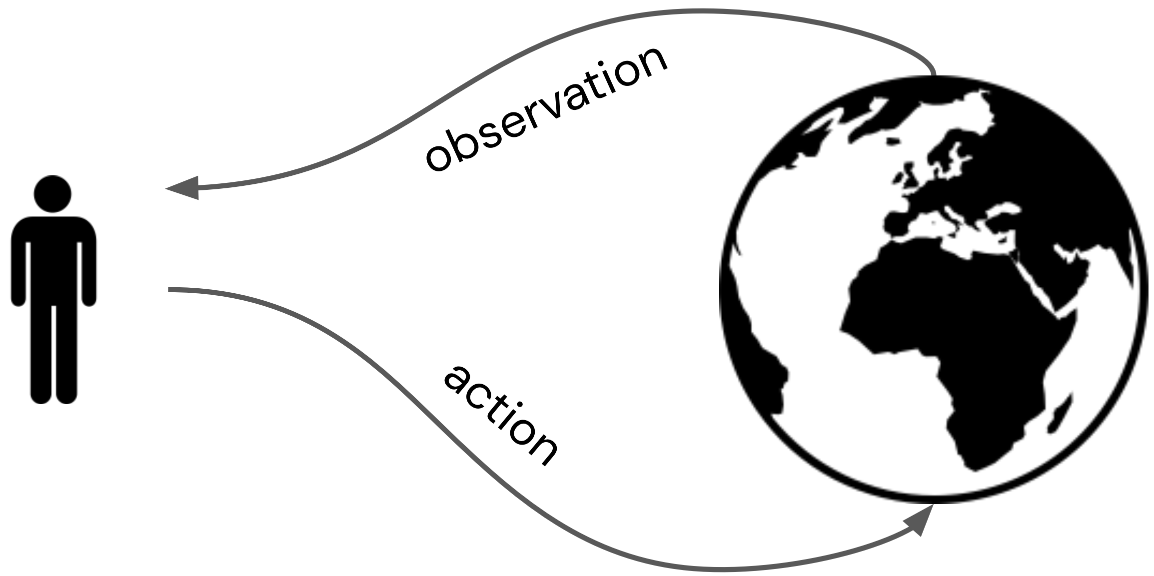
Then, all we need to do is specify goals

Reinforcement learning



Reinforcement learning is the science of learning to make decisions from experience

Reinforcement learning



Reinforcement learning is the science of learning to make decisions from experience

Decisions are **sequential** and can have **long-term effects**

Reinforcement learning

Goal:

Make good decisions, by learning from experience

Reinforcement learning formalises the **problem**,

not one specific solution

It provides a framework to think about how to learn to act

Reinforcement learning

Reinforcement learning is a formalisation of the AI problem

Reinforcement learning

Reinforcement learning is a formalisation of the AI problem

Solutions can use and rely on insights from related fields

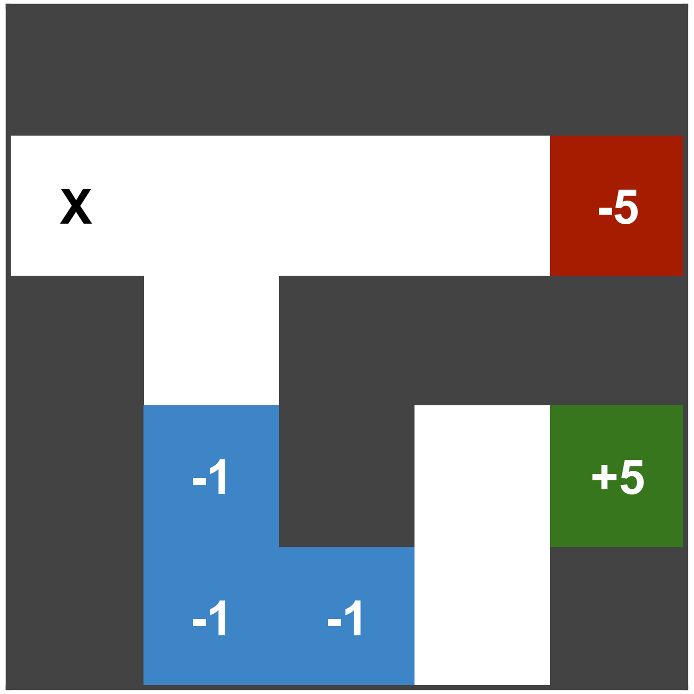
(E.g., deep reinforcement learning = RL + DL techniques)

Reinforcement learning

RL problems are **sequential decision problems**

Need to trade off near-term and long-term reward

Need to actively search for information (explore)



Reinforcement learning

Why learn? Learning is important to:

1. Automatically find good solutions

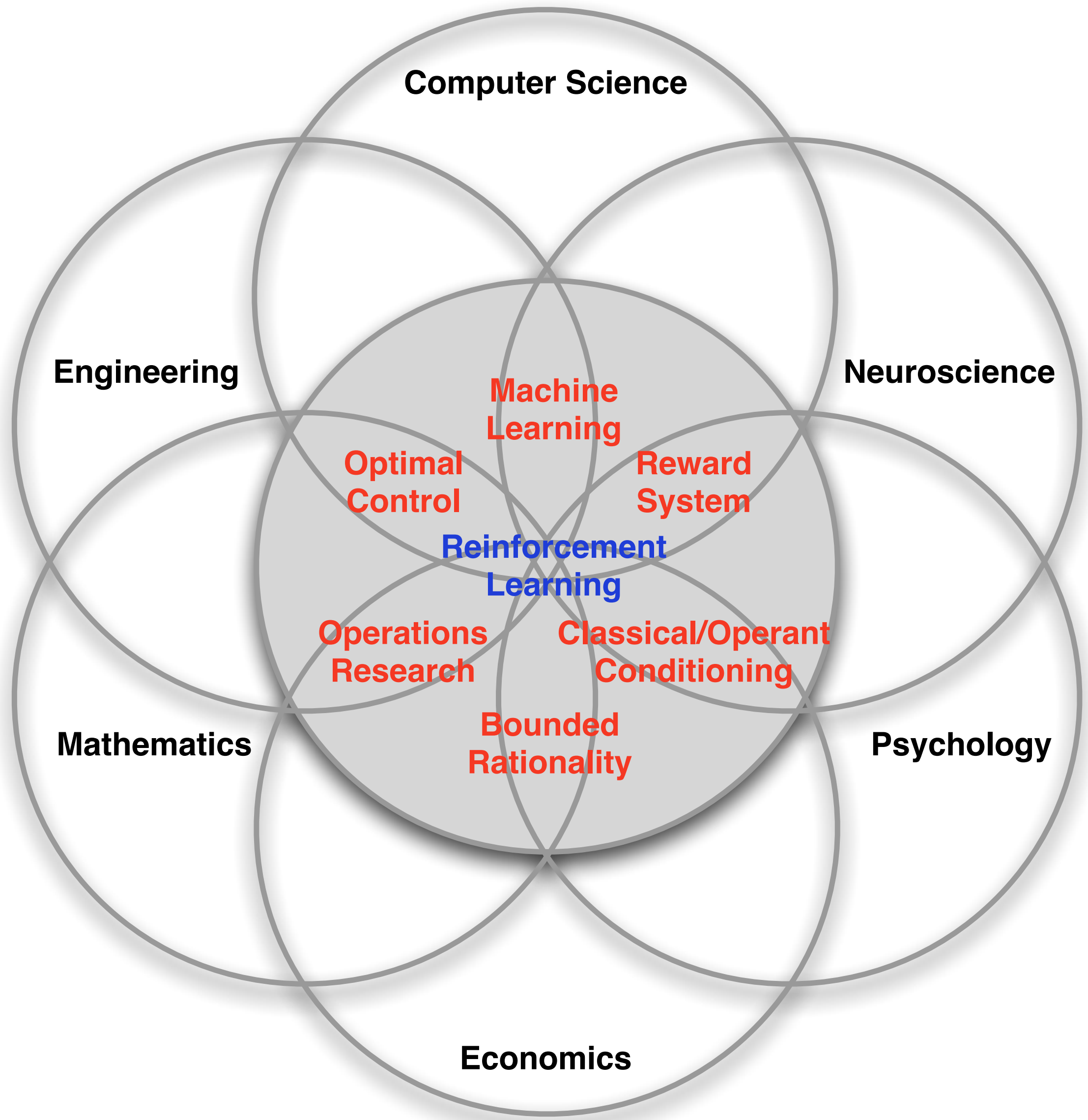
   (E.g., how to play Go, fold proteins, control a plant, etc.)

2. Adapt online

   (E.g., traverse an unexpected new terrain on a different planet)

RL provides a formalisation for both types of learning

Related Disciplines



Characteristics of Reinforcement Learning

How does reinforcement learning differ from other machine learning paradigms?

- No direct supervision, only a **reward** signal
- Feedback can be delayed, not instantaneous
- Time matters
- Earlier decisions affect later interactions

Examples of decision problems

- Examples:
  - Fly a helicopter
  - Manage an investment portfolio
  - Control a power station
  - Make a robot walk
  - Play video or board games
- These are all reinforcement learning problems
  (no matter which solution method you use)

Video

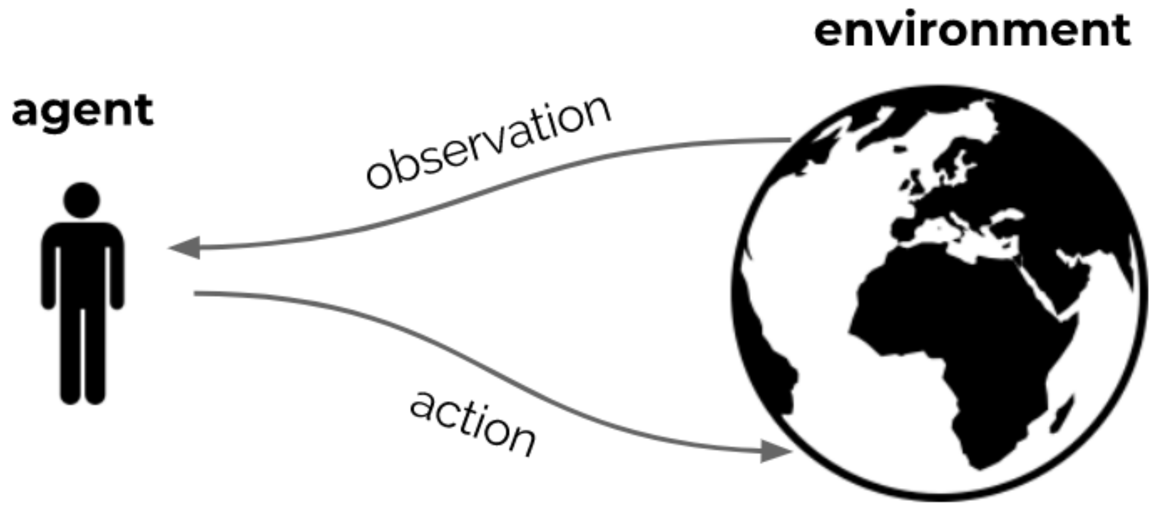
Atari

Core concepts

The reinforcement learning formalism includes

- **Environment** (dynamics of the problem)
- **Reward** signal (specifies the goal)
- **Agent**, containing:
  - Agent state
  - Policy
  - Value function (probably)
  - Model (optionally)

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 R_{t+1})

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

Values

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

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + R_{t+2} + R_{t+3} + ... \mid S_t = s\right]
\end{aligned}
$$

- 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

$$
\begin{aligned}
G_t &= R_{t+1} + G_{t+1} \\
v(s) &= \mathbb{E}\left[R_{t+1} + v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

Actions in sequential problems

- 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**:

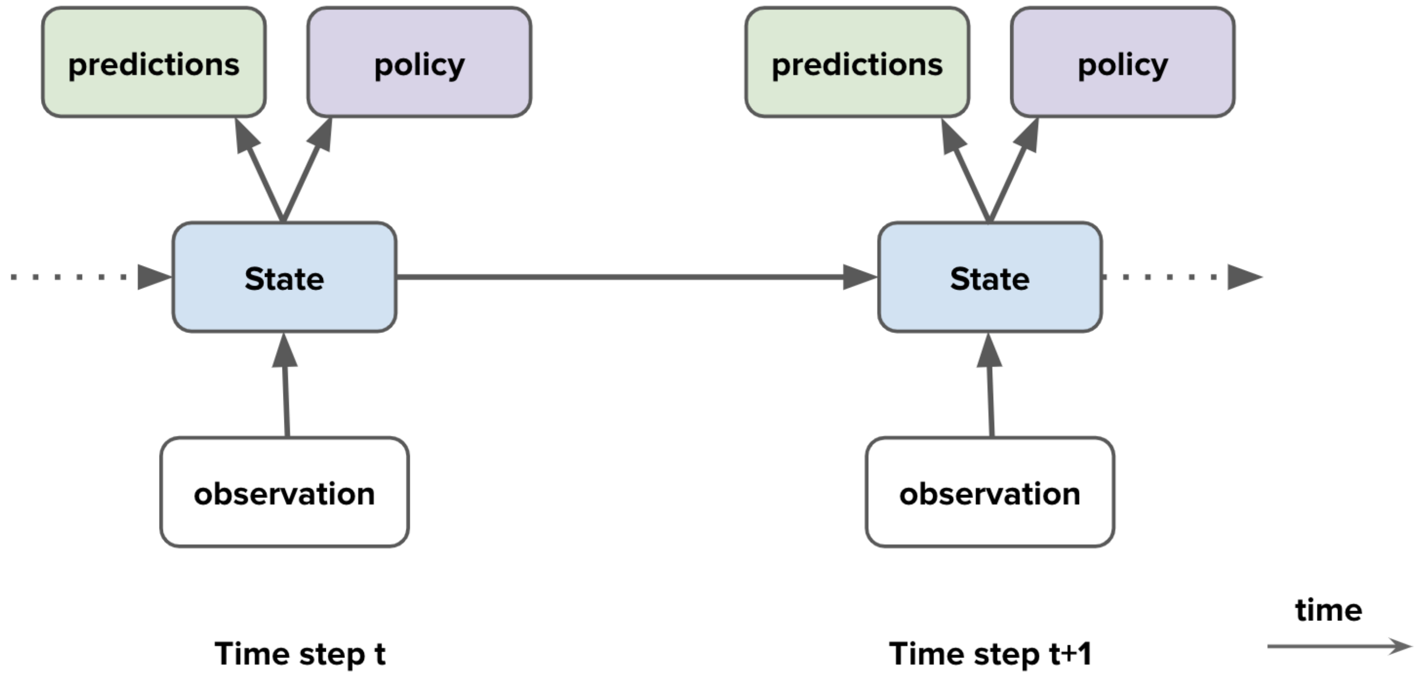
$$\begin{aligned} q(s, a) &= \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}\left[R_{t+1} + R_{t+2} + R_{t+3} + ... \mid S_t = s, A_t = a\right] \end{aligned}$$

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

Agent components

Agent components

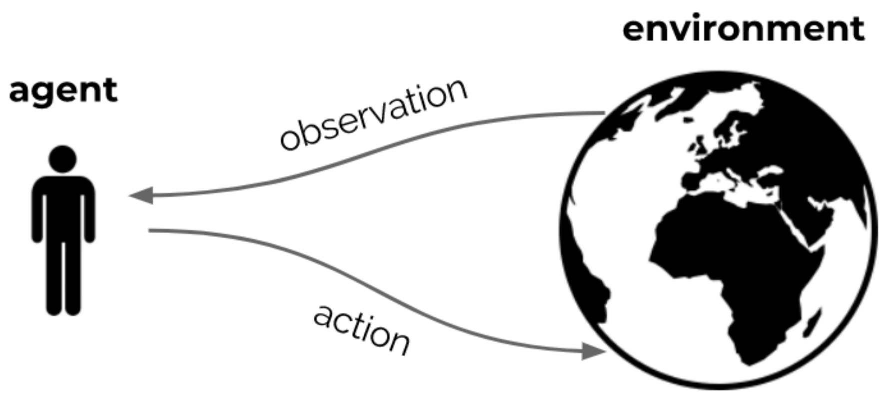
- **Agent state**
- Policy
- Value functions
- Model



State

- Actions depend on the **state** of the agent
- Both agent and environment may have internal state
- In the simplest case, there is only one state (next lecture)
- Often, there are many different states — sometimes infinitely many
- The state of the agent generally differs from the state of the environment
- The agent may not know the full state of the environment

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

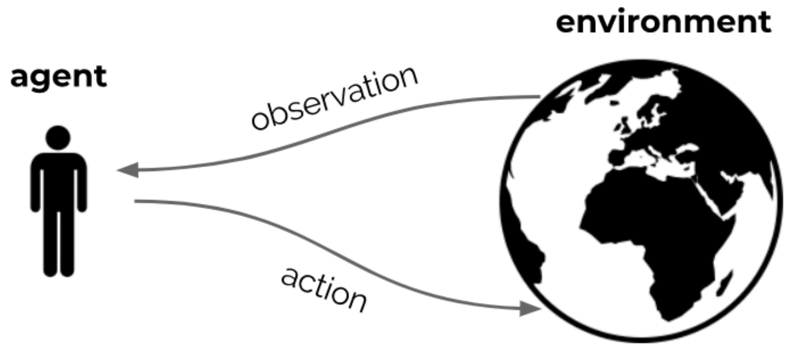
Agent State

- A **history** is a sequence of observations, actions, rewards

$$\mathcal{H}_t = O_0, A_0, R_1, O_1, ..., O_{t-1}, A_{t-1}, R_t, O_t$$

- For instance, the sensorimotor stream of a robot
- This history can be used to construct an **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:

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

- Then the agent is in a **Markov decision process**

Markov decision processes

Markov decision processes (MDPs) provide a useful mathematical framework

Definition

A decision process is Markov if

$$p(r, s \mid S_t, A_t) = p(r, s \mid \mathcal{H}_t, A_t)$$

- "The future is independent of the past given the present"

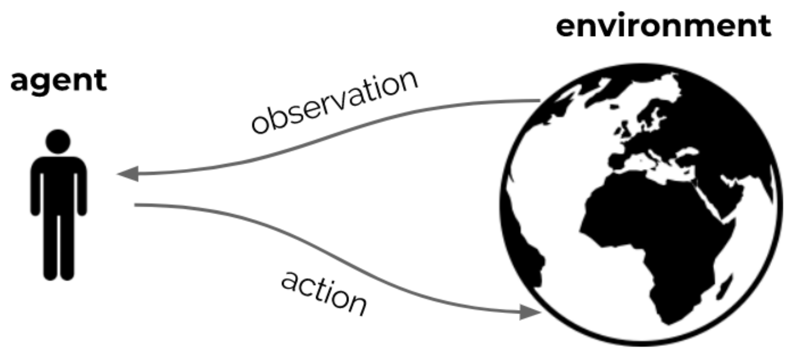
$$\mathcal{H}_t \to S_t \to \mathcal{H}_{t+1}$$

- Once the state is known, the history may be thrown away
- The environment state is typically Markov
- The history $\mathcal{H}_t$ is Markov

Partially Observable Environments

- **Partial observability**: The agent gets partial information
  - A robot with camera vision isn't told its absolute location
  - A poker playing agent only observes public cards
- Now the observation is not Markov
- Formally this is a **partially observable Markov decision process** (POMDP)
- The **environment state** can still be Markov, but the agent does not know it

Agent State



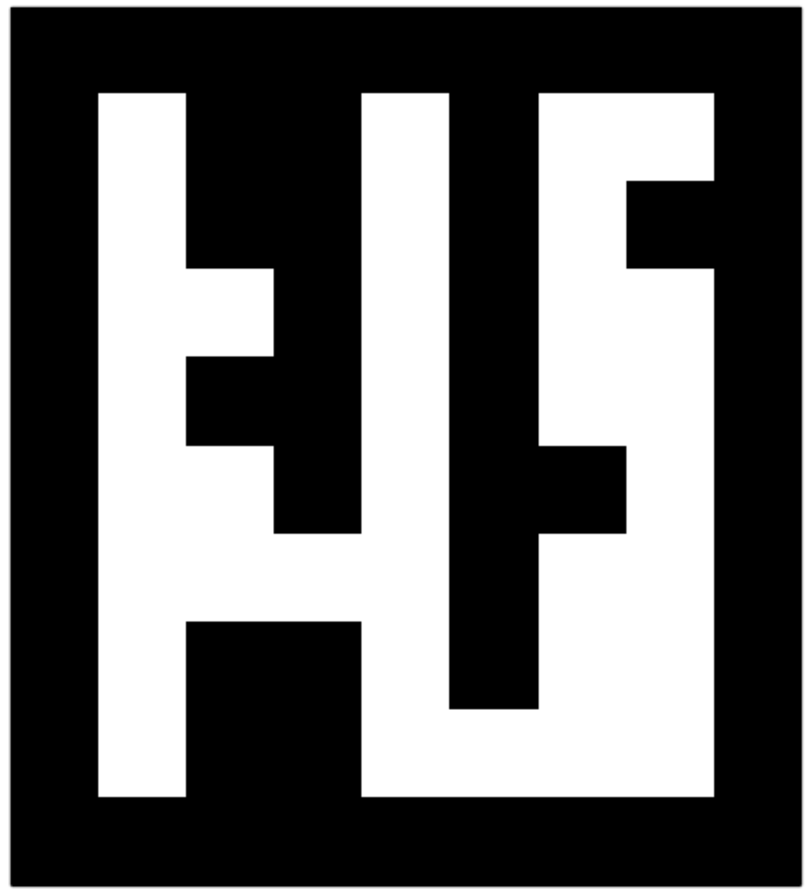
- The **agent state** is a function of the history
- The agent's action depends on its state
- For instance, $S_t = O_t$
- More generally:

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

  where u is a 'state update function'
- The agent state is typically much smaller than the environment state

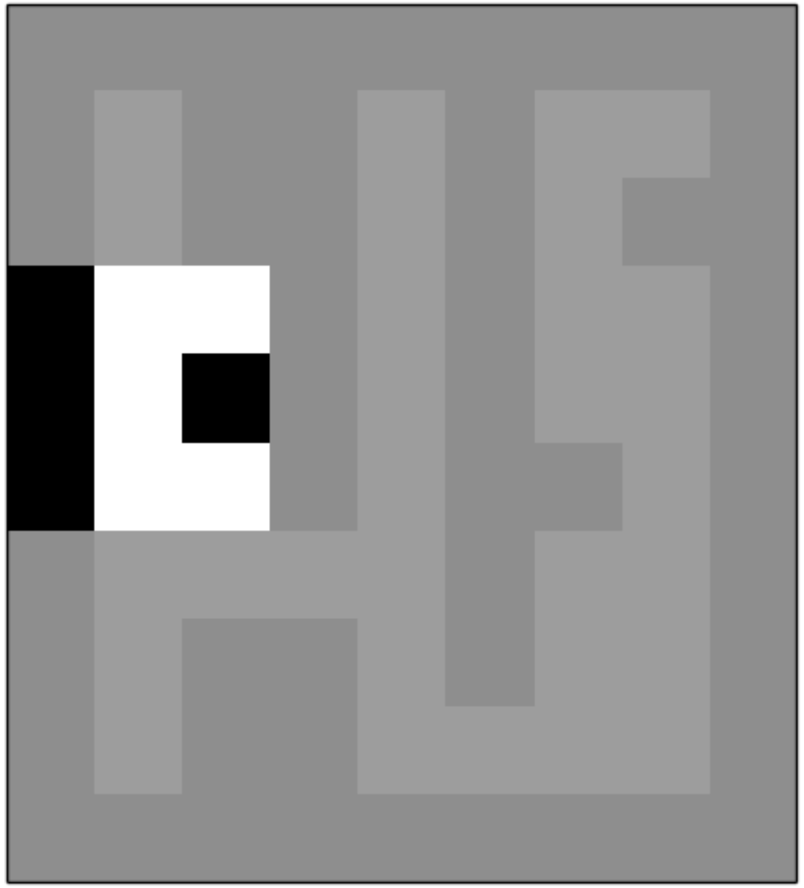
Agent State

The full environment state of a maze



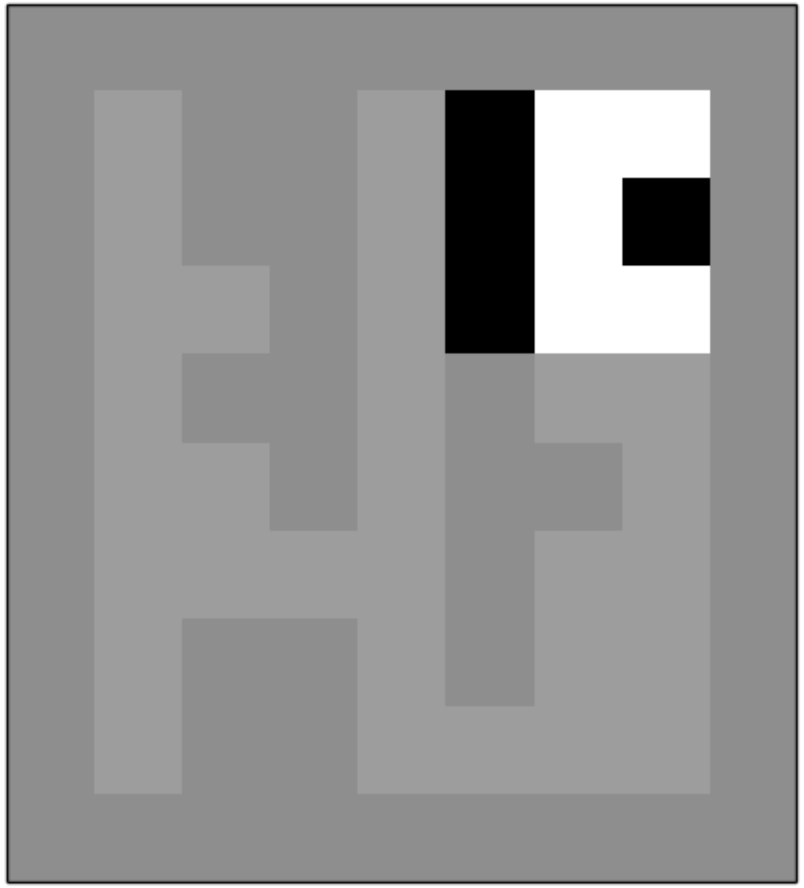
Agent State

A potential observation



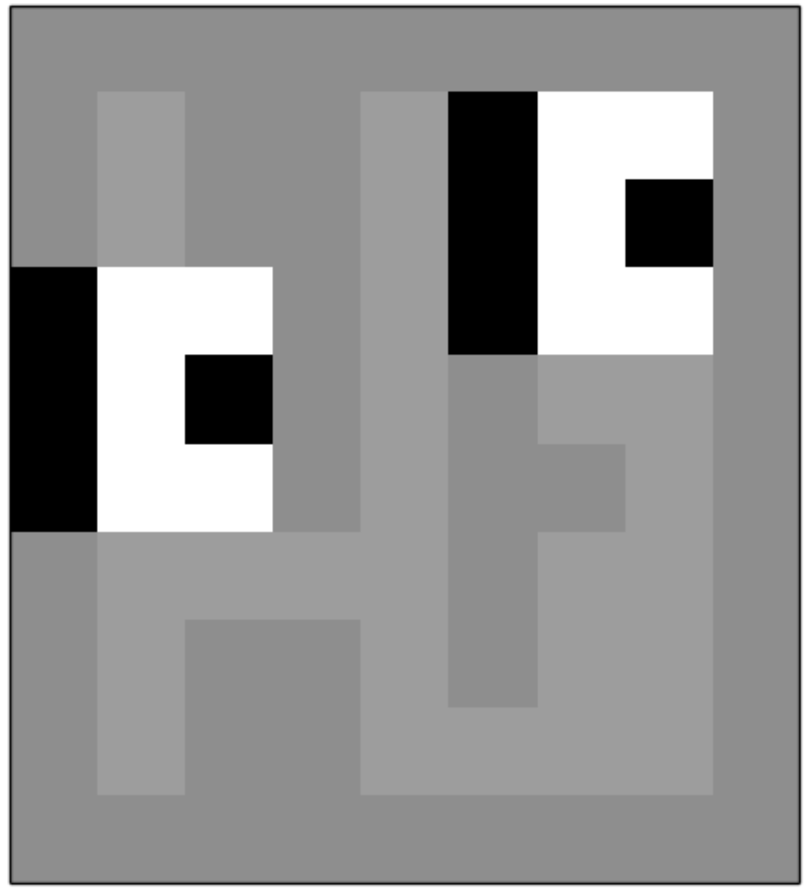
Agent State

An observation in a different location



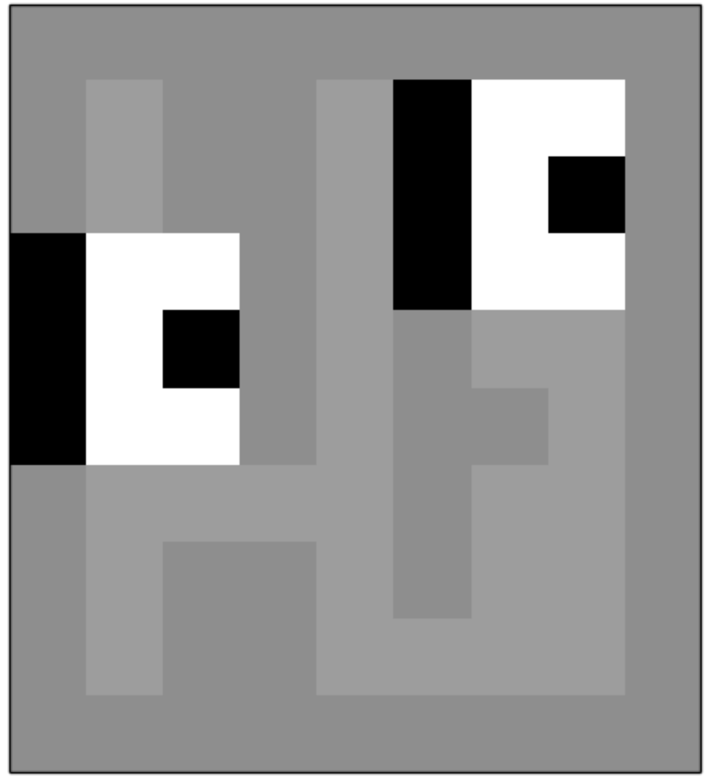
Agent State

The two observations are indistinguishable



Agent State

These two states are not Markov



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: $S_t = \mathcal{H}_t$ (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 Markov agent state is often not feasible
- More importantly, the state should allow good policies and value predictions

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: $\pi(A|S) = p\,(A|S)$

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

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

  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:

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

  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

Agent components

Agent components

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

Model

- A **model** predicts what the environment will do next
- E.g., $\mathcal{P}$ predicts the next state

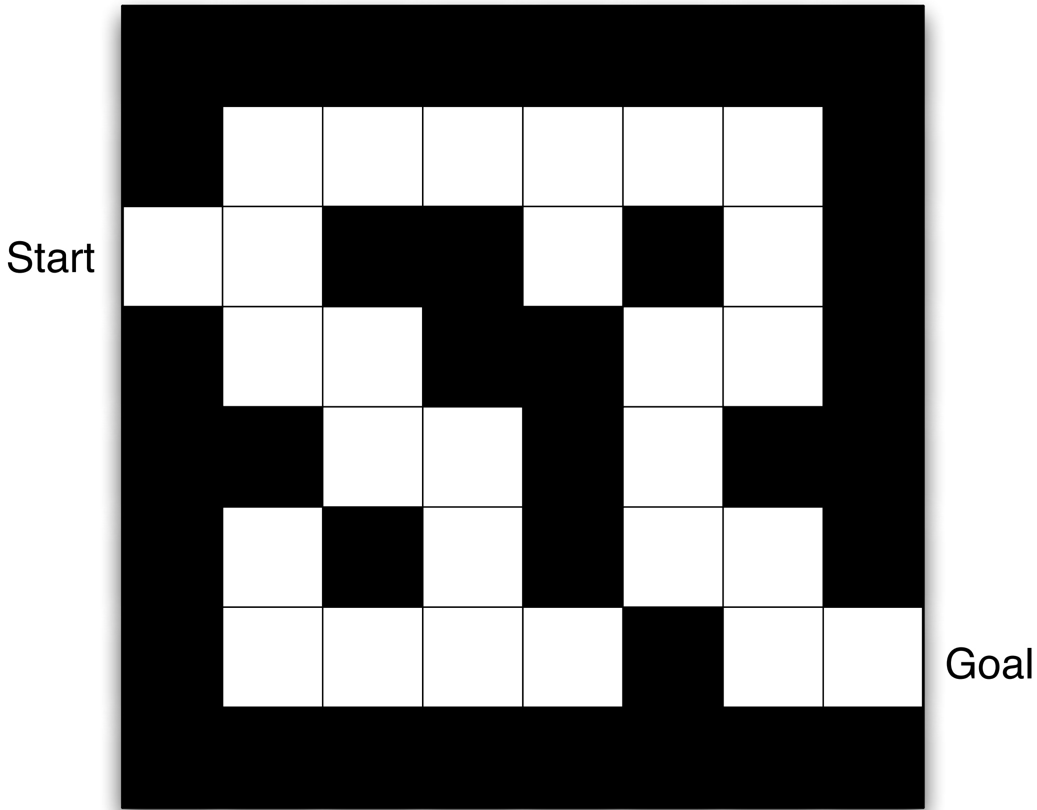
$$\mathcal{P}(s, a, s') \approx p\left(S_{t+1} = s' \mid S_t = s, A_t = a\right)$$

- E.g., $\mathcal{R}$ predicts the next (immediate) reward

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

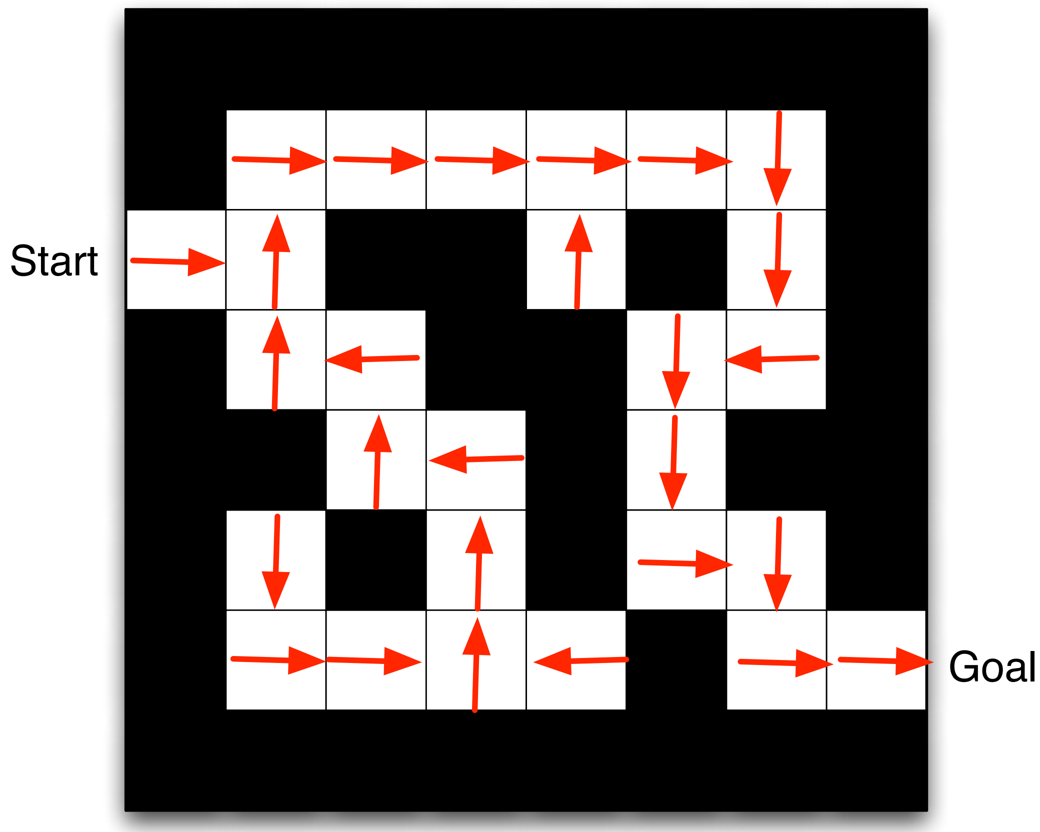
- A model does not immediately give us a good policy - we would still need to plan
- We could also consider **stochastic** (**generative**) models

Maze Example



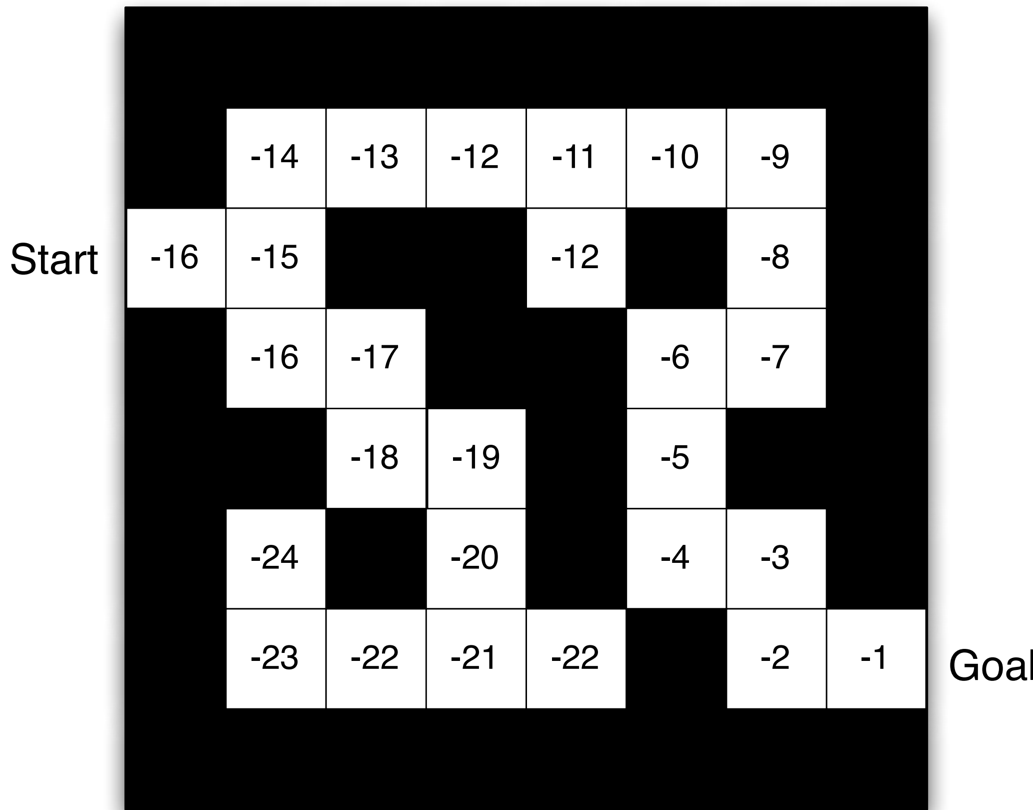
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example: Policy



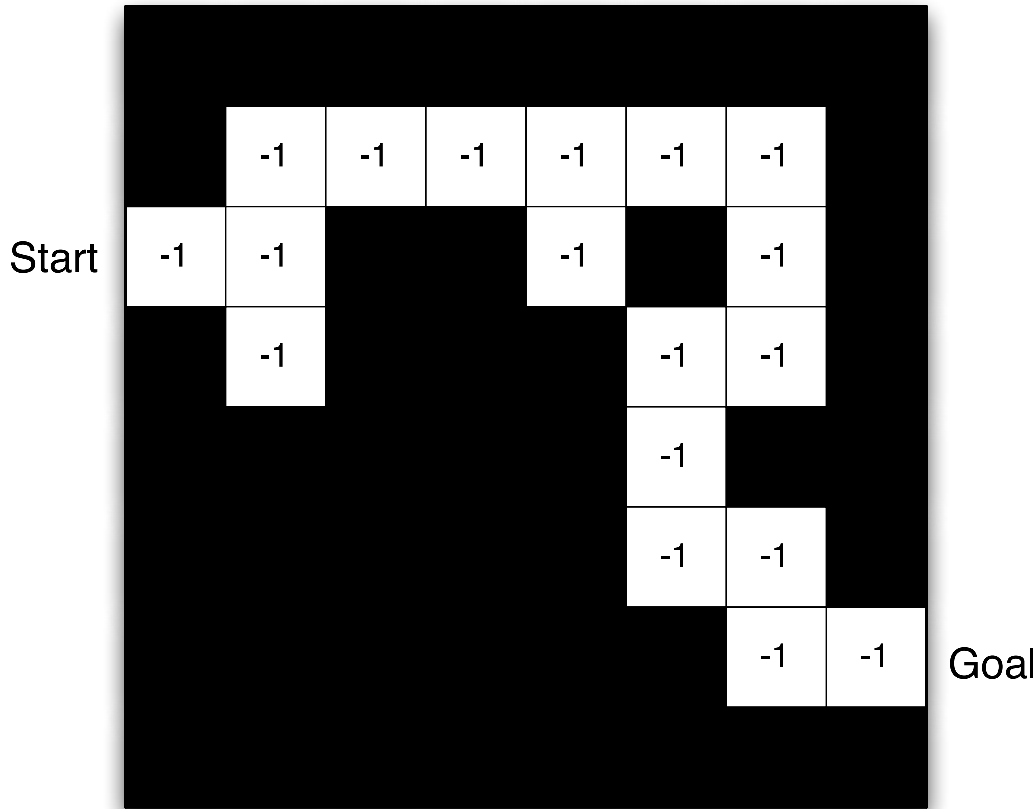
- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function



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

Maze Example: Model



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

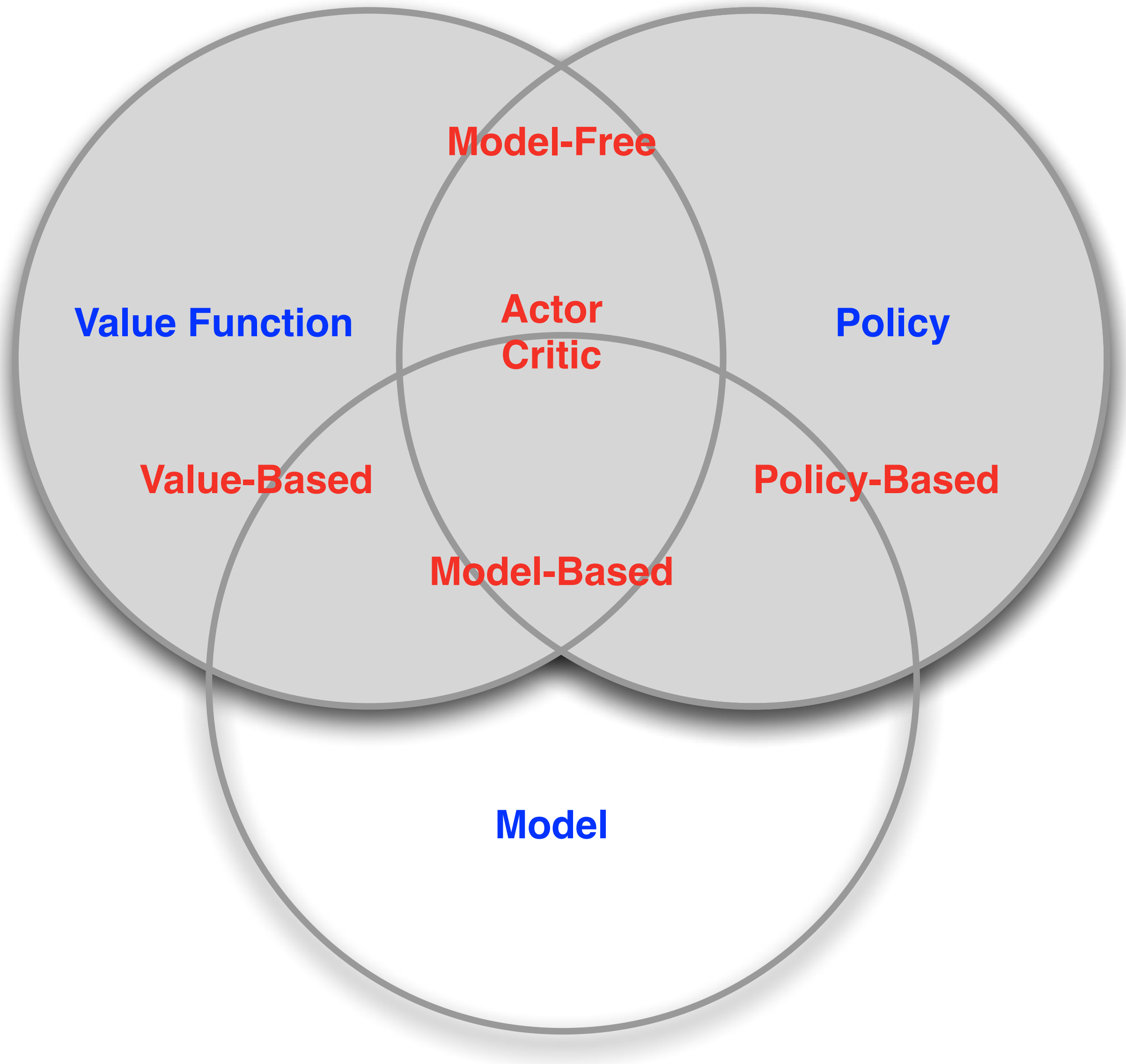
Categorizing agents

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

Categorizing agents

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

Agent Taxonomy



Challenges in reinforcement learning

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
  - The agent plans in this model (without external interaction)
  - a.k.a. reasoning, pondering, thought, search, planning

Prediction and Control

- Prediction: evaluate the future (for a given policy)
- Control: optimize the future (find the best policy)
- These are strongly related:

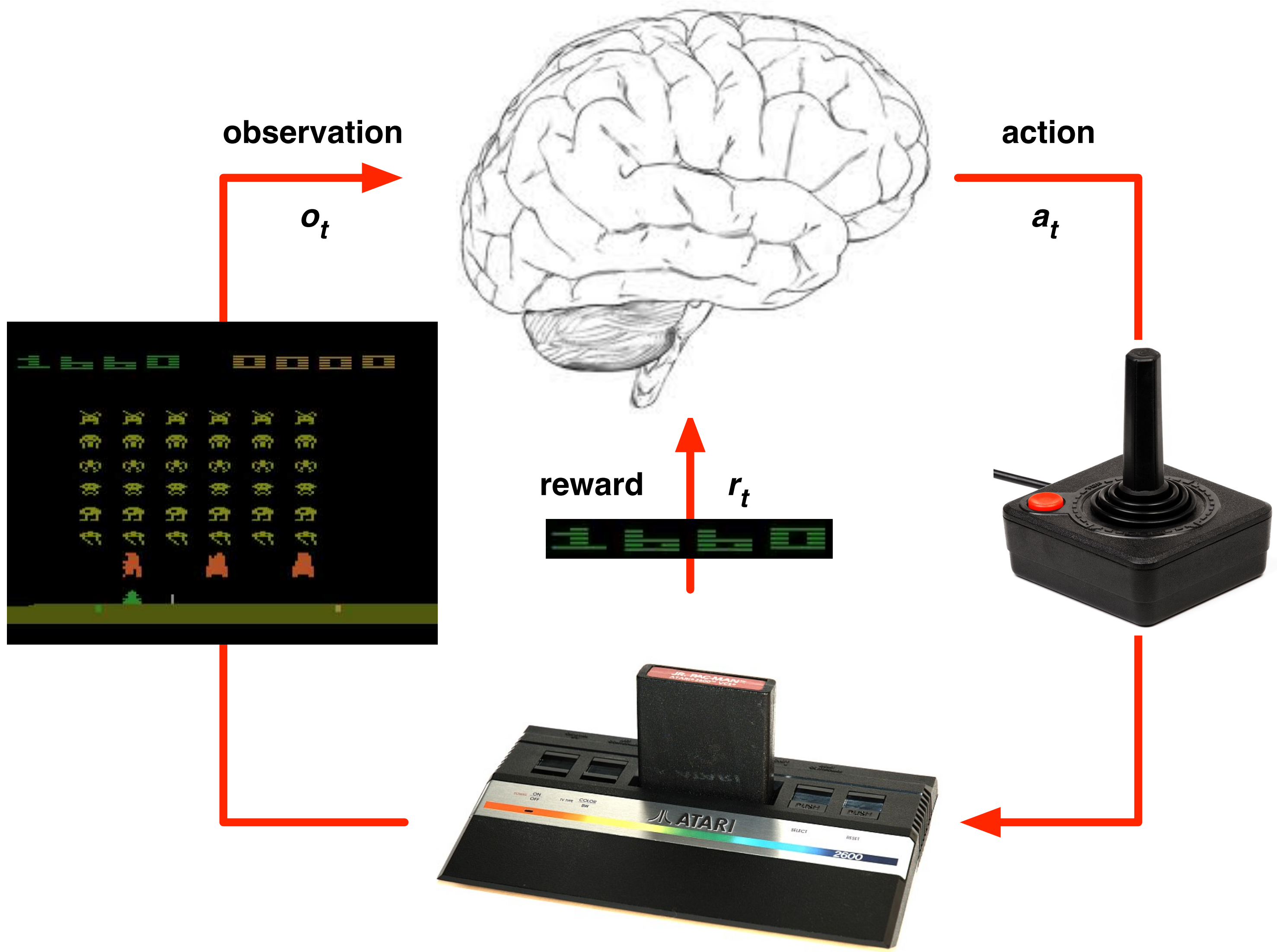
$$\pi_*(s) = \underset{\pi}{\operatorname{argmax}}\, v_\pi(s)$$

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

Learning the components of an agent

- All components are functions
  - Policies map states to actions
  - Value functions map states to values
  - Models map states to states and/or rewards
  - State updates map states and observations to new states
- We can represent these as neural networks, then use **deep learning** to optimize
- Take care: we often violate assumptions from supervised learning (iid, stationarity)
- Deep learning is an important tool
- Deep reinforcement learning is a rich and active research field

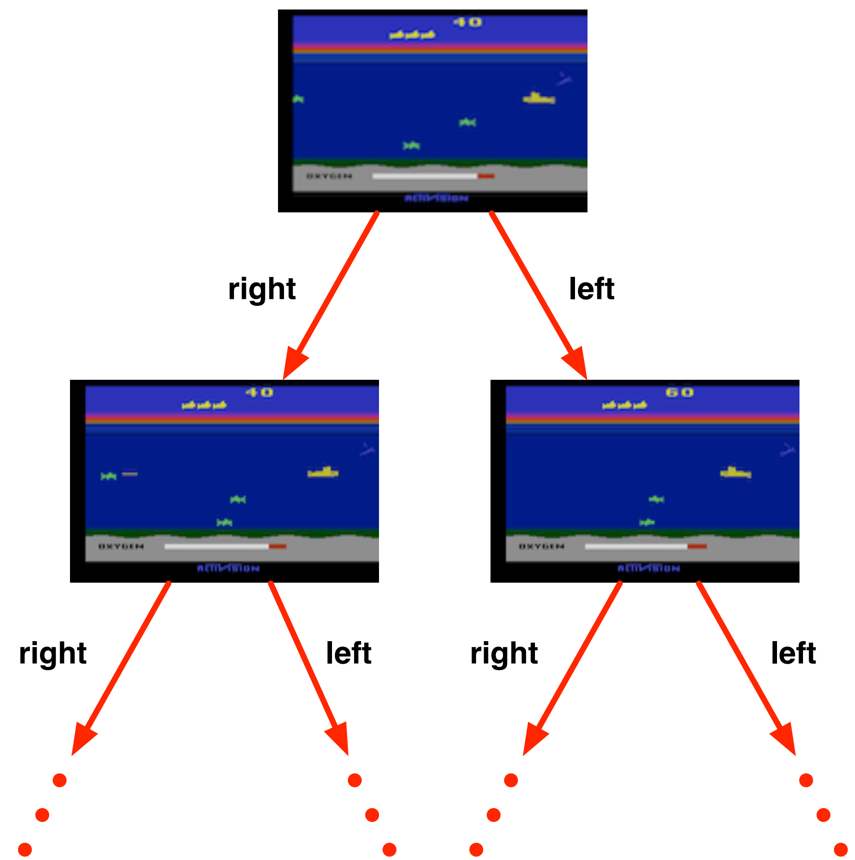
Atari Example: Reinforcement Learning



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

Atari Example: Planning

- Rules of the game are known
- Can query emulator: perfect model
- If I take action a from state s:
  - what would the next state be?
  - what would the score be?
- Plan ahead to find optimal policy
- Later versions add noise, to break algorithms that rely on determinism



Exploration and Exploitation

- We learn by trial and error
- The agent should discover a good policy
- ...from new experiences
- ...without sacrifycing too much reward along the way

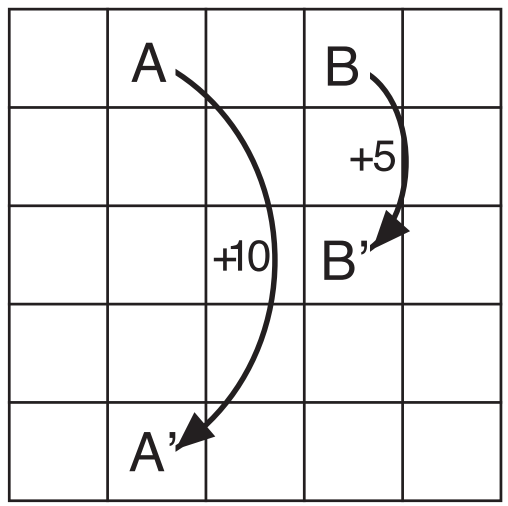
Exploration and Exploitation

- **Exploration** finds more information
- **Exploitation** exploits known information to maximise reward
- It is important to explore as well as exploit
- This is a fundamental problem that does not occur in supervised learning

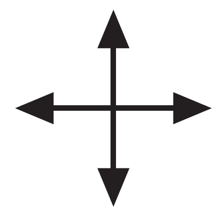
Examples

- Restaurant Selection
  - Exploitation: Go to your favourite restaurant
  - Exploration: Try a new restaurant
- Oil Drilling
  - Exploitation: Drill at the best known location
  - Exploration: Drill at a new location
- Game Playing
  - Exploitation: Play the move you currently believe is best
  - Exploration: Try a new strategy

Gridworld Example: Prediction



(a)



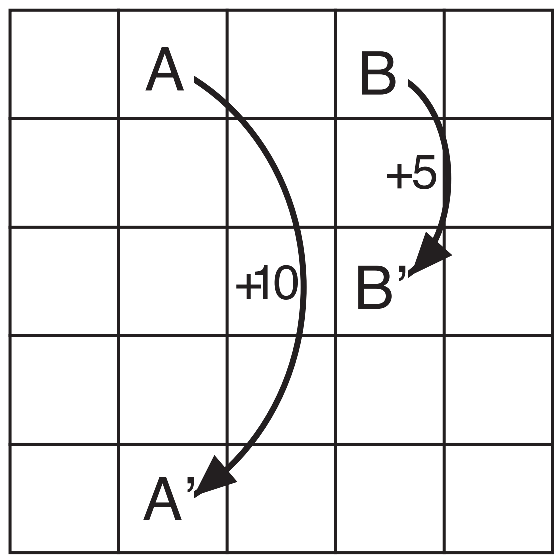
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

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

What is the value function for the uniform random policy?

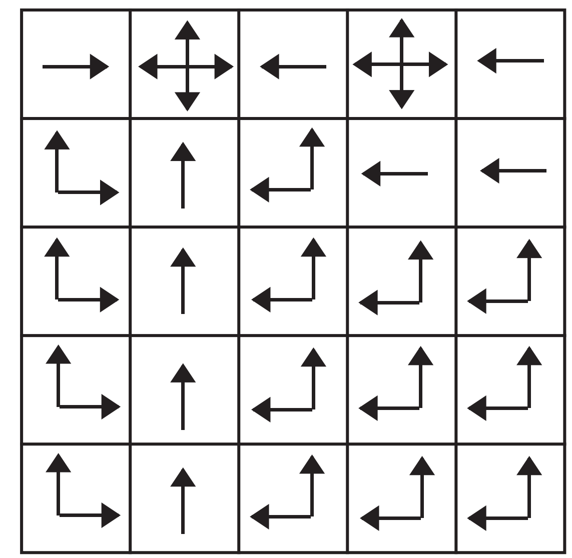
Gridworld Example: Control



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?

Course

- In this course, we discuss how to learn by interaction
- The focus is on understanding core principles and learning algorithms

Topics include

- Exploration, in bandits and in sequential problems
- Markov decision processes, and planning by dynamic programming
- Model-free prediction and control (e.g., Q-learning)
- Policy-gradient methods
- Challenges in deep reinforcement learning
- Integrating learning and planning

Video

Locomotion