Reinforcement learning

Dr. Aissa Boulmerka

The National School of Artificial Intelligence
aissa.boulmerka@ensia.edu.dz

2024-2025

Chapter 1
Introduction to reinforcement learning

Outline

- Presentation
- Information about the course
- Introduction to Reinforcement Learning (RL)
- Examples & Applications of RL
- Formalizing the RL problem

Presentation

- Aissa Boulmerka
- PhD From ESI, Algiers, Algeria/UQO, Gatineau, Ottawa, Canada (2016)
- Assistant Professor, Mila University Center (2009)
- Associate Professor, Mila University Center (2019)
- Associate Professor, National Higher School of Artificial Intelligence (2024)
- **Teaching :** Image Processing, Machine Learning, Deep Learning, Theory of computing, etc.
- **Research Areas:** Deep learning models applied to image/video segmentation and recognition.
- **Emails :** aissa.boulmerka@ensia.edu.dz

Information About the course

- **Course Description:**

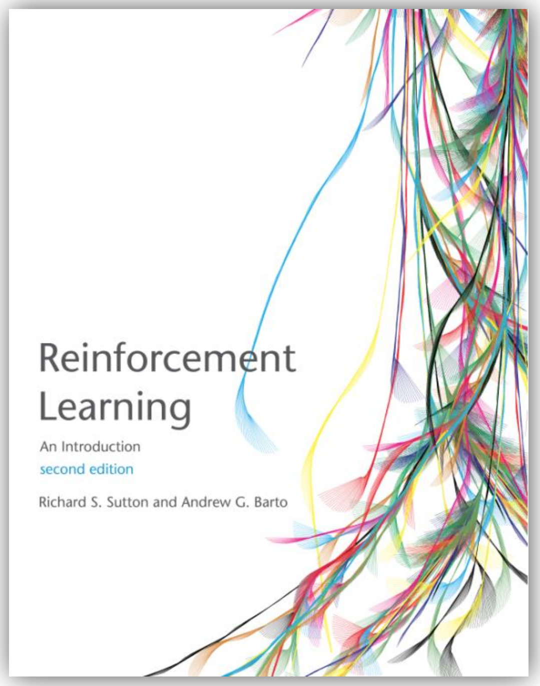
  Reinforcement Learning (RL) is a general framework that can capture the evolving and unpredictable learning environment and has been used to design intelligent agents that achieve high level performances on challenging tasks. This course will provide a solid introduction to the field of reinforcement learning and students will learn about the core challenges and approaches, including generalization and exploration.

- **Prerequisite :**

  Linear algebra, probability, programming.

- **Textbook:**

  Richard S. Sutton and Andrew G. Barto. **Reinforcement Learning: An Introduction.** MIT Press, 2018.



Course Content

Chapter 1: Introduction to reinforcement learning

Chapter 2: Multi-armed Bandits

Chapter 3: Finite Markov Decision Processes

Chapter 4: Dynamic Programming (*)

Chapter 5: Monte Carlo Methods (*)

Chapter 6: Temporal-Difference Learning

Chapter 7: Planning and Learning with Tabular Methods

Chapter 8: On-policy Prediction with Approximation

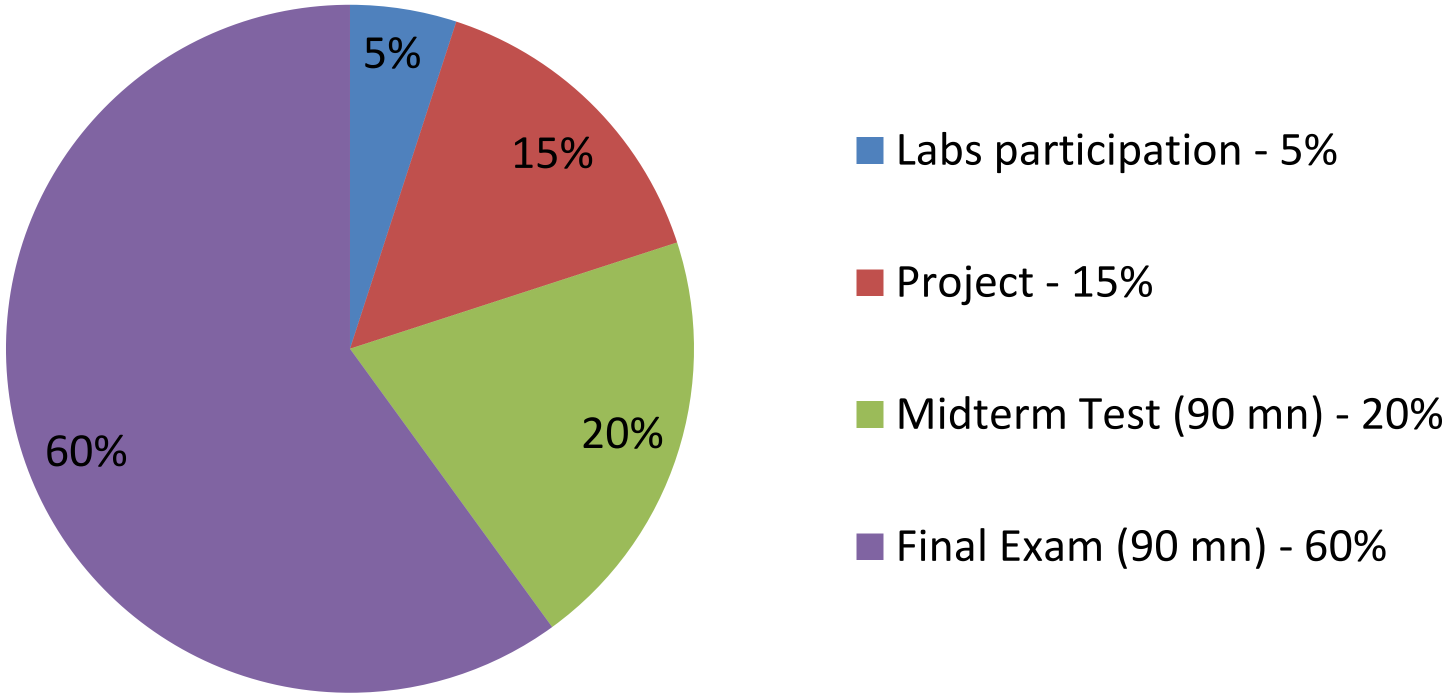
Chapter 9: On-policy Control with Approximation

Chapter 10: Policy Gradient Methods

Chapter 11: Advanced Topics (*)

(*) this chapter will be presented by Dr. Ferhi Nadir

Evaluation


5%
15%
20%
60%
🟦 Labs participation - 5%
🟥 Project - 15%
🟩 Midterm Test (90 mn) - 20%
🟪 Final Exam (90 mn) - 60%

Introduction to reinforcement learning (RL)

What is Reinforcement Learning (RL)

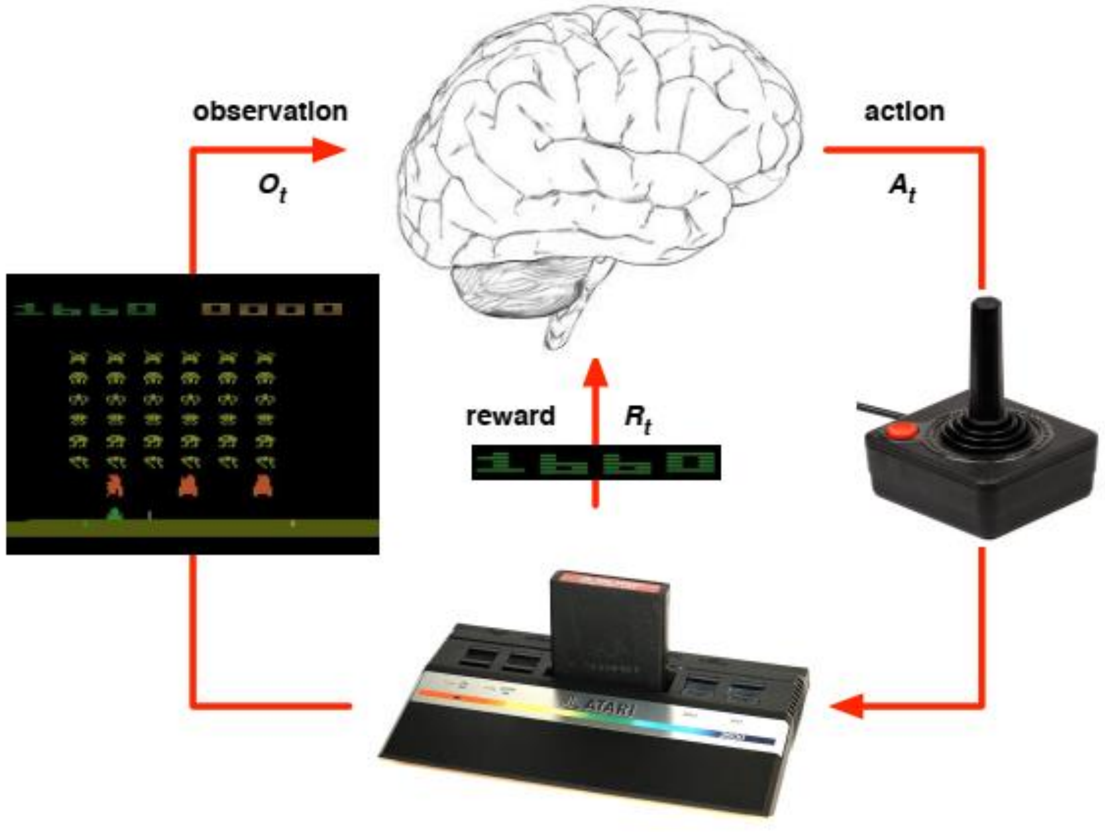
- A way of programming **agents** by **reward** and punishment without needing to specify **how** the task is to be achieved (*).
- This approach allows us to train intelligent entities (**robots, software, or agents**) in a much more **flexible and efficient** way.
- Instead of **hard-coding** every action or behavior the agent needs to perform, we simply guide it by providing **positive reinforcement** for **desirable behaviors** and **negative reinforcement** for **undesirable** ones.
- Over time, the agent **learns** on its own how to **solve problems and achieve goals**.

- (*) L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement learning: A survey." *Journal of artificial intelligence research* 4 (1996): 237-285.

Learning through trial and error

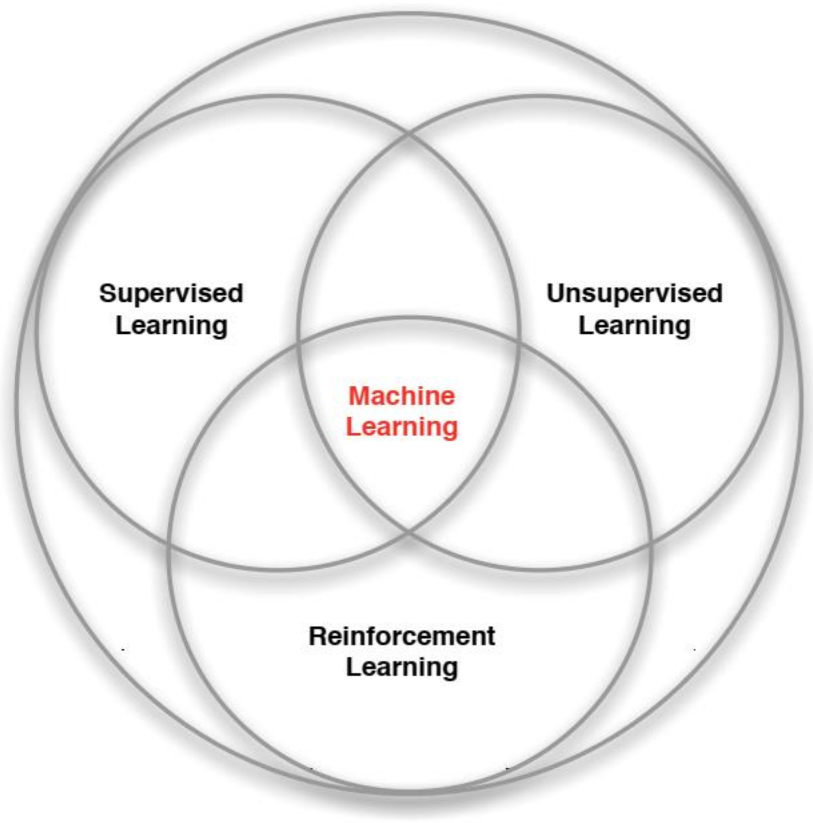
- The process of learning to ride a bike relies on trial and error, with feedback from falls (pain) and successes (reward), rather than detailed explanations.

Atari Example



David Silver. RL Course at UCL.

Branches of Artificial Intelligence



David Silver. RL Course at UCL.

Characteristics of Reinforcement Learning

What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data*)
- Agent’s actions affect the subsequent data it receives

* non-independent, identically distributed data

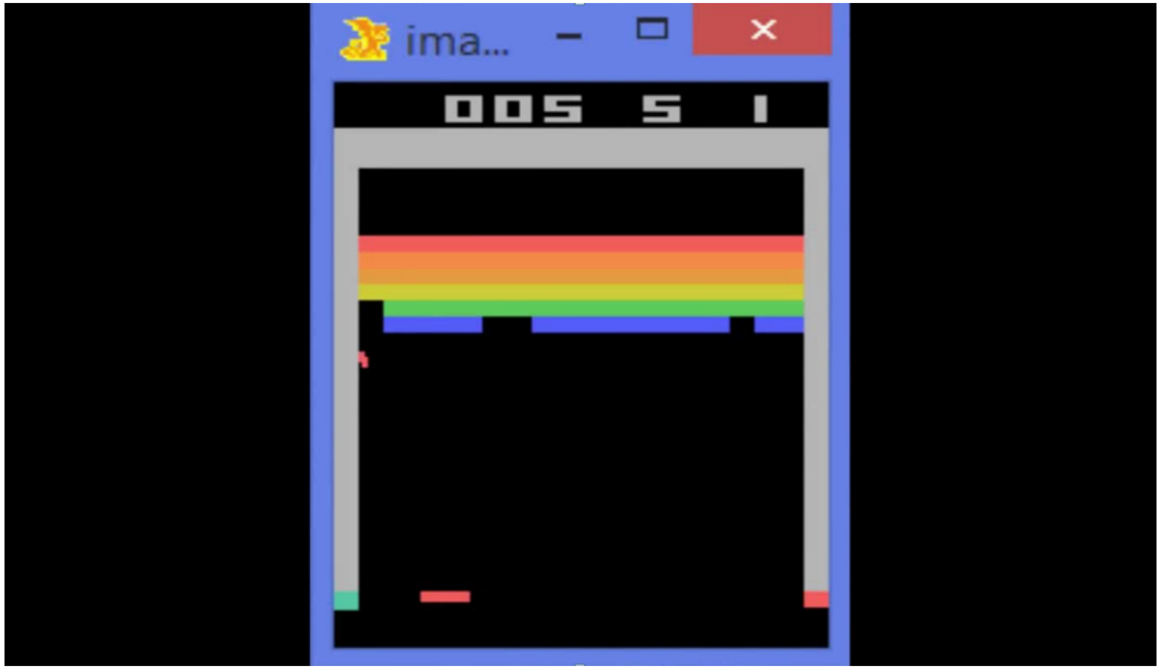
Examples & applications of RL

Helicopter Flight Manoeuvers

Video: https://www.youtube.com/watch?v=0JL04JJjocc

Pieter Abbeel, Adam Coates, Morgan Quigley, Andrew Ng. An Application of Reinforcement Learning to Aerobatic Helicopter Flight. NIPS 2006.

Computer Games: Atari 2600 Games



Video : https://www.youtube.com/watch?v=V1eYniJ0Rnk

Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature (2015).

Games (Chess, Go, etc.) and video games

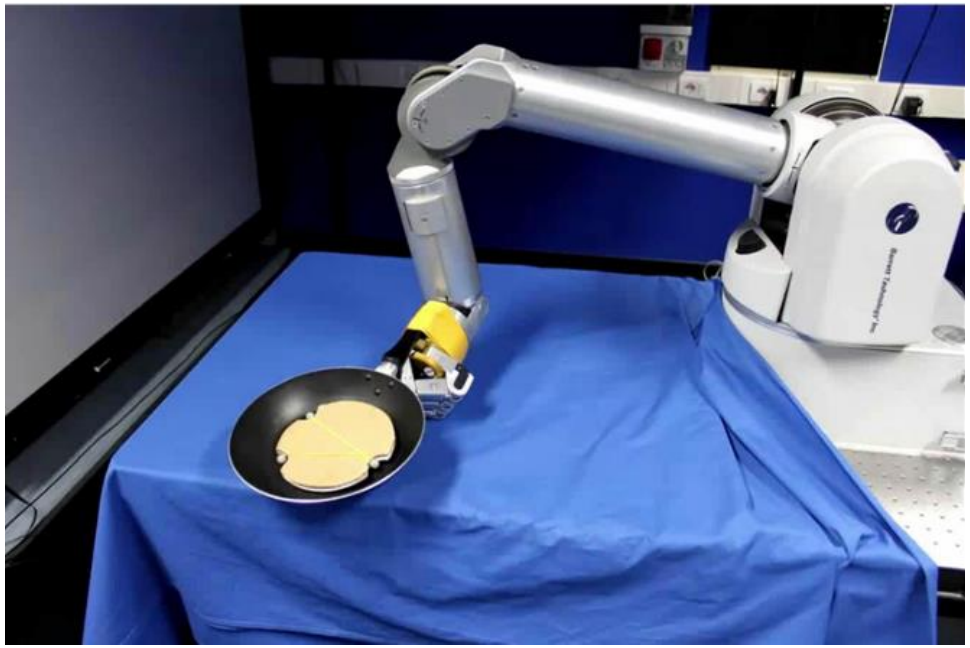
Chess

Go

- Matthew Lai, Giraffe: Using deep reinforcement learning to play chess (2015)
- Silver, D., Schrittwieser, J., Simonyan, K. et al. Mastering the game of Go without human knowledge. Nature (2017).

Robotics: Robot Learns to Flip Pancakes

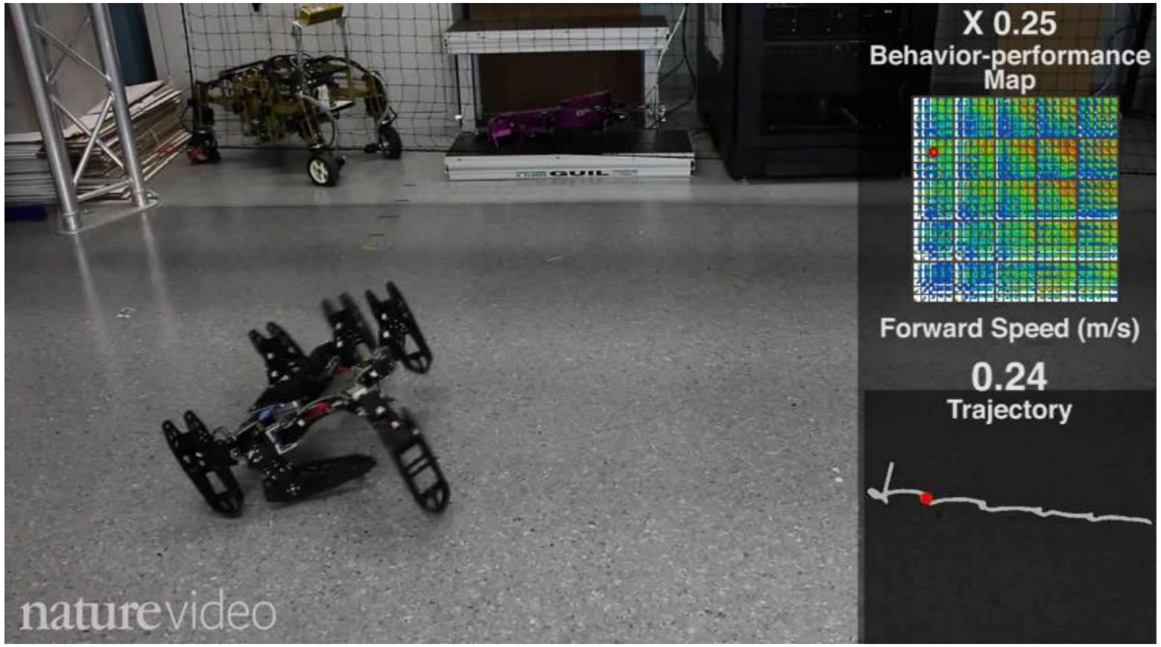
Robotics manipulation and autonomous systems: eg. elevator optimization, humanoid robots, emergency response robots, self-driving vehicles, etc.



Video : https://www.youtube.com/watch?v=bxtPyJqVrmk

P. Kormushev, et. al. Robot motor skill coordination with EM-based Reinforcement Learning. IEEE IROS 2010.

Robotics: Injured robots learn to limp



Video : https://www.youtube.com/watch?v=KFDMm666QBU

Antoine Cully, et. al. Robots that can adapt like animals. Nature (2015).

Finance: investment portfolio

Other RL application areas and success stories

- **Video Games:** super-human performance in Go, chess, victories in complex strategy games
- **Supply chain and manufacturing:** Amazon Robotics uses RL to optimize warehouse robots for picking and placing items efficiently.
- **Energy**: Deepmind’s RL reduce Google Data center cooling by 40%
- **NLP:** ChatGPT (OpenAI) uses RL from Human Feedback (RLHF) to fine-tune responses, making them coherent and context-aware.
- **Recommender systems** : example: online advertisements.

Basically, any complex dynamical system that is difficult to model analytically can be application of RL

Formalizing the RL problem

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 (we call this the **return**)

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

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

Definition:

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

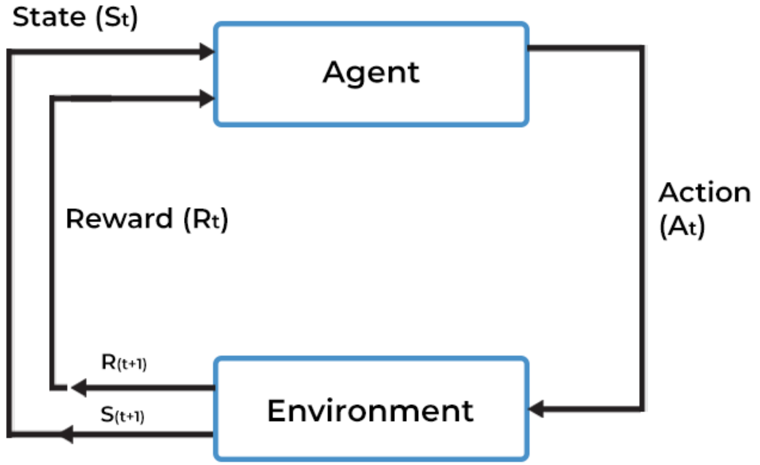
Examples of Rewards

- **Fly stunt manoeuvres in a helicopter**
  - + reward for following desired trajectory
  - - reward for crashing
- **Defeat the world champion at chess or Go**
  - +/- reward for winning/losing a game
- **Manage an investment portfolio**
  - + reward for each amount of money in the bank
- **Control a power station**
  - + reward for producing power
  - - reward for exceeding safety thresholds
- **Make a humanoid robot walk**
  - + reward for forward motion
  - - reward for falling over
- **Play many different Atari games better than humans**
  - +/- reward for increasing/decreasing score

Sequential Decision Making

- **Goal: select actions to maximize total future reward**
- 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:**
  - A **financial investment** (may take months to mature)
  - Refuelling a **helicopter** (might prevent a crash in several hours)
  - Learning a **new skill** (can be costly & time-consuming at first)
- **Conclusion:** Instead of taking the most immediately **rewarding action** at every step, the agent must learn to **balance** short-term and long-term rewards to **maximize success** over time.

RL Interaction loop framework



- 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})
- Increment **time** t at env. step

History and State

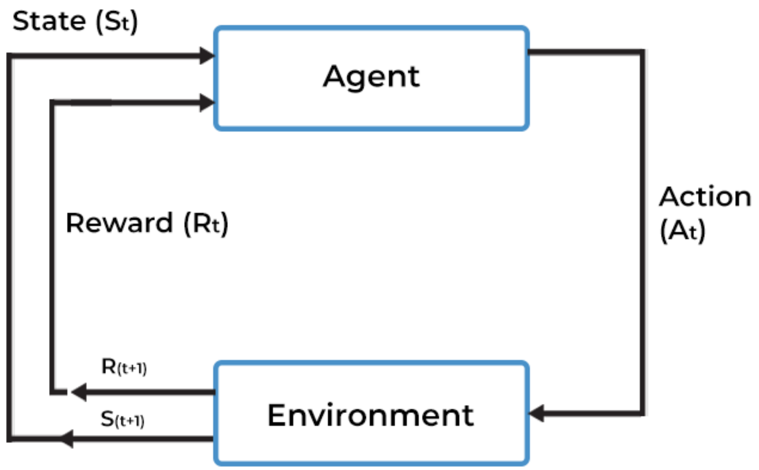
- The **history** is the full sequence of **observations**, **actions**, **rewards**

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

- i.e. all observable variables up to time t
- For instance, the sensorimotor stream of a robot
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- This history is used to construct the **agent state** $\mathcal{S}_t$
- Formally, state is a function of the history:

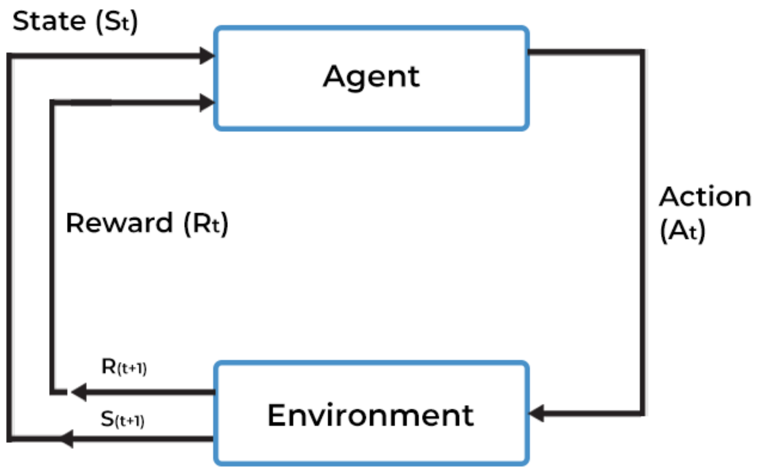
$$\mathcal{S}_t = f(\mathcal{H}_t)$$

Environment State



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

Agent State



- The **agent state** S_t^a is the agent's internal representation
- RL algorithms use this information to pick the **next action**
- The **agent state** is a function of the history, for instance

$$S_t^a = f(H_t)$$

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

Information State

- An information state (a.k.a. Markov state) contains all useful information from the history.

Definition:

A state S_t is Markov if and only if

$$\boldsymbol{P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \cdots, S_t]}$$

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

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future
- The environment state S_t^e is Markov
- The history H_t is Markov

Fully Observable Environments

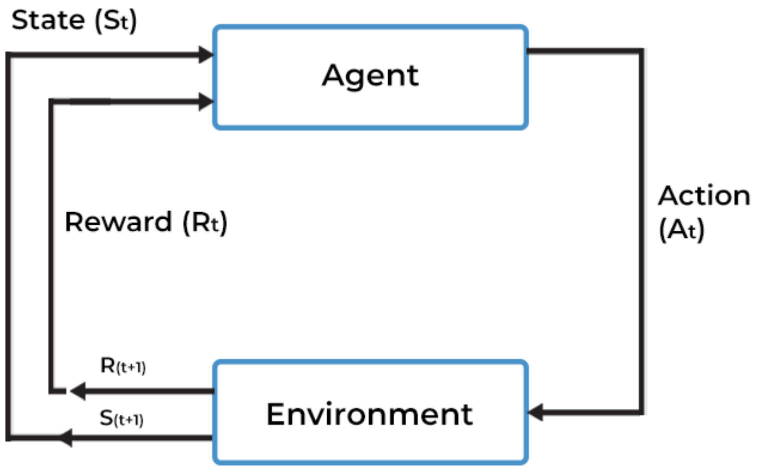
Full observability

Suppose the agent sees the full environment state

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

$$O_t = S_t^e = S_t^a$$

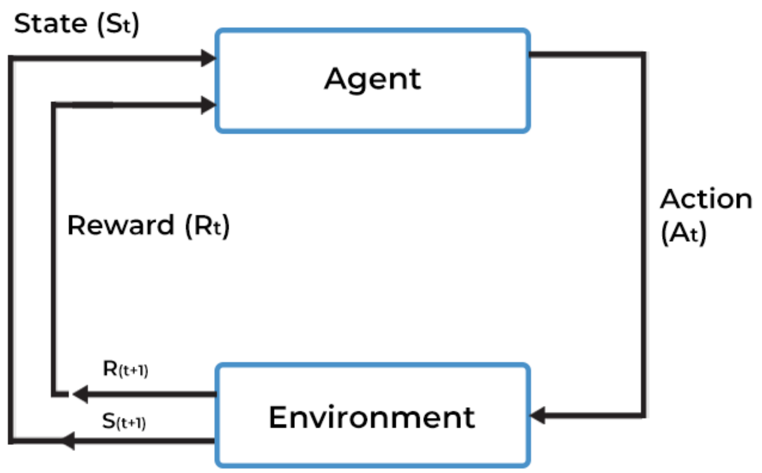
- Formally, this is a **Markov decision process** (MDP)



Partially Observable Environments

- **Partial observability:** The observations are not Markovian
  - A robot with camera vision isn't told its absolute location
  - A trading agent only observes current prices
- Now agent state $\neq$ environment state
- Formally this is a **partially observable Markov decision process (POMDP)**
- Agent must construct its own state representation S_t^a, e.g.
  - Complete history: $S_t^a = H_t$
  - Beliefs of environment state: $S_t^a = (P[S_t^e = s^1], \cdots, P[S_t^e = s^n])$
  - Recurrent neural network: $S_t^a = \sigma(S_{t-1}^a W_s + Q_t W_o)$

Components of RL problem



An RL agent may include one or more of these components:

- **Policy :** the way the agent behaves at a given time
- **Value function:** how good is each state and/or action
- **Model**: agent’s representation of the environment

Policy

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

Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, \pi]$$

- We introduced a **discount factor** $\gamma \in [0,1]$
  - Trades off importance of immediate vs long-term rewards
- The value depends on a **policy** π
- The goal is to **maximize value**, by picking suitable actions
- Rewards and values define **utility** of states and action (no supervised feedback)
- It is also possible to condition the value on **actions**:

$$q_\pi(s, \mathrm{a}) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = \mathrm{a}]$$

Models

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

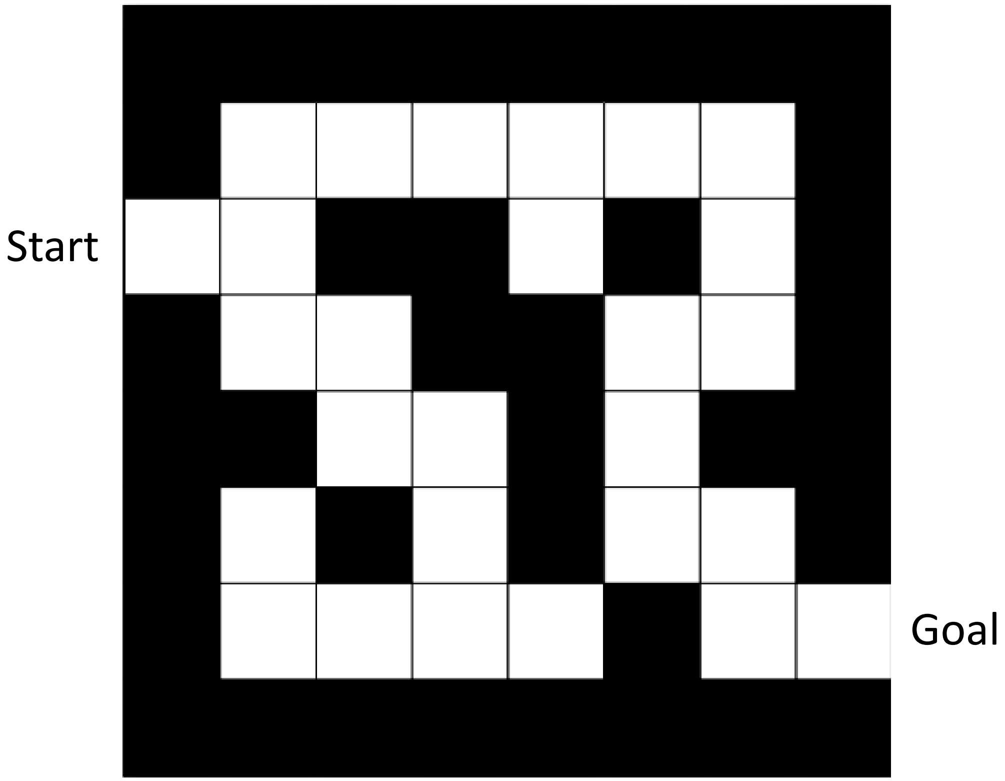
$$\mathcal{P}(s, a, s') \approx P(S_{t+1} = s' | \, S_t = s, A_t = a)$$

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

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

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

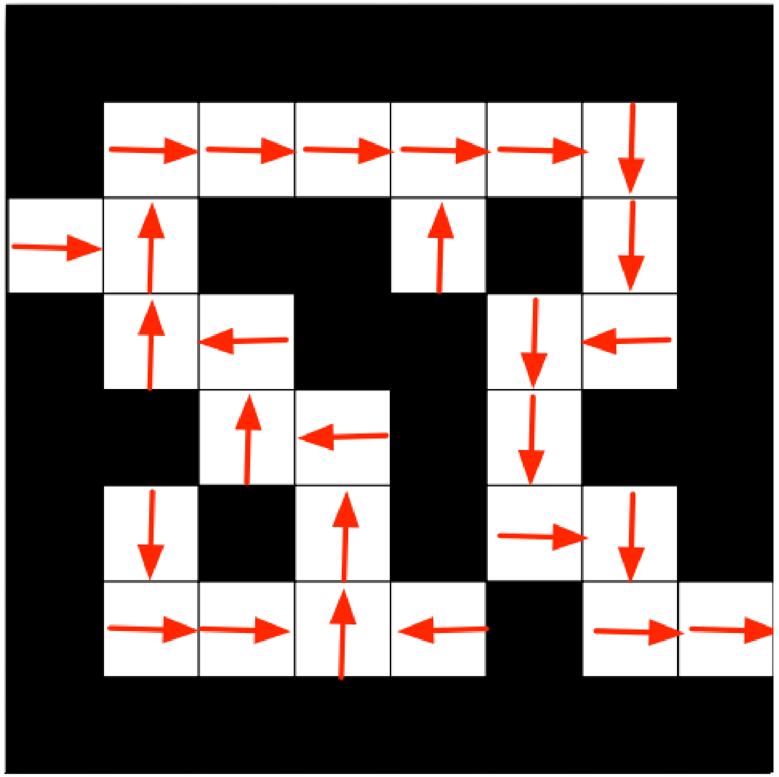
Maze Example



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

Maze Example: Policy

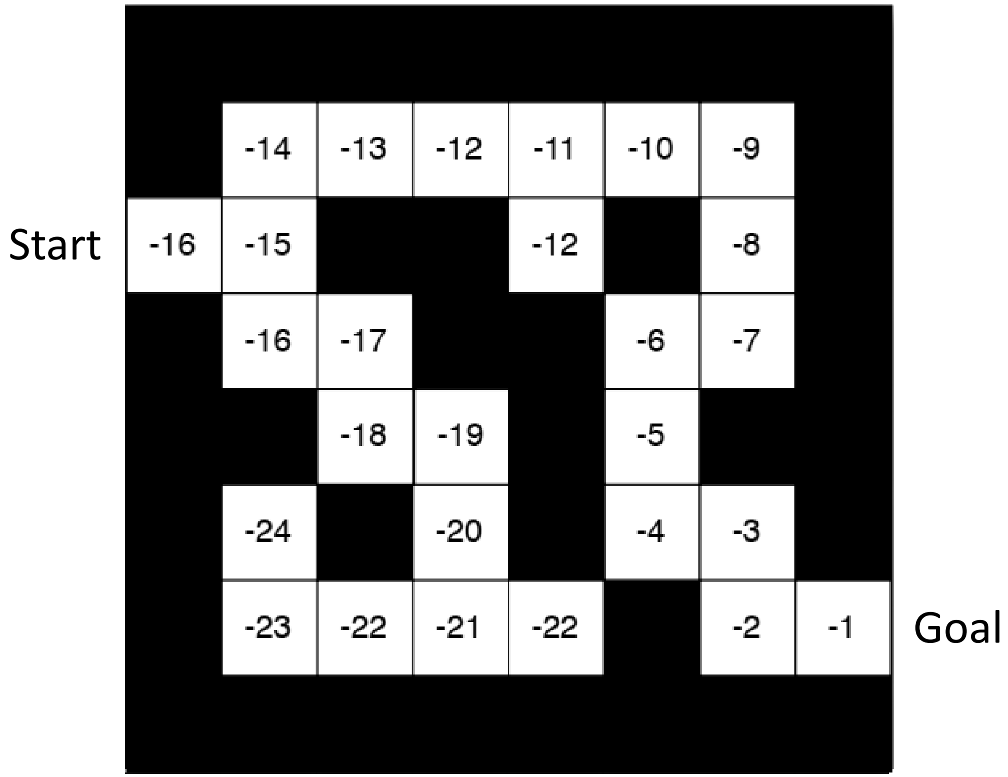
Start



Goal

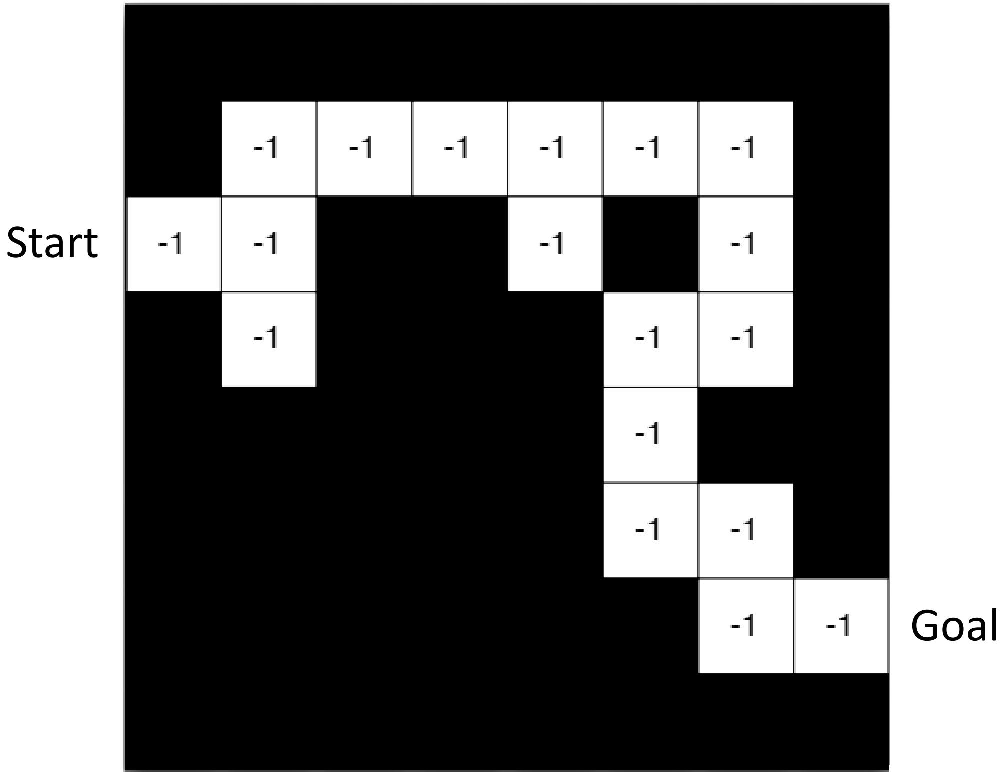
- Arrows represent **policy** $\pi(s)$ for each state s

Maze Example: Value Function



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

Maze Example: Model



- Grid layout represents partial transition model
- Numbers represent immediate reward from each state s (same for all a and s' in this case)

Agent Categories (1)

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

Agent Categories (2)

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

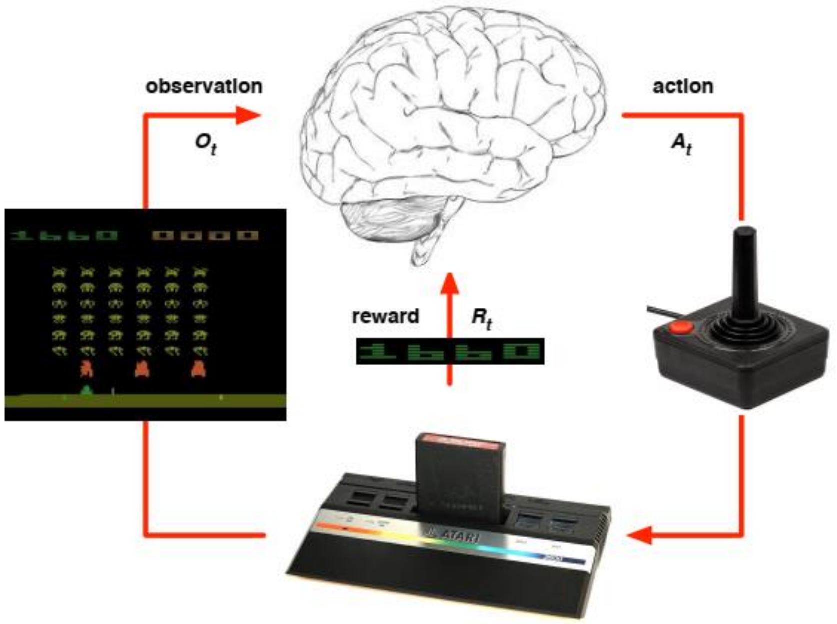
Subproblems of the RL Problem

Learning and Planning

Two fundamental problems in reinforcement learning:

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

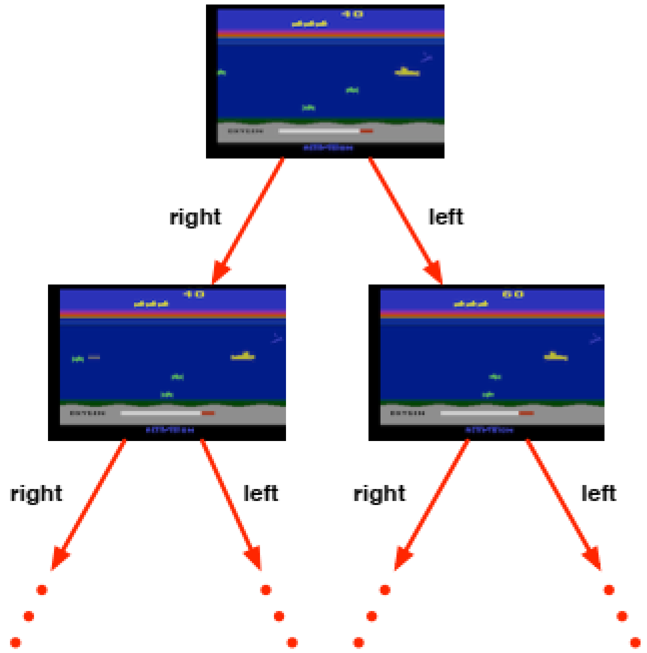
Atari Example: Learning



- 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 inside agent’s brain
- 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
  - e.g. tree search



Learning Agent Components

- All components are functions
  - **Policies:** $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (or to probabilities over $\mathcal{A}$)
  - **Value functions:** $v : \mathcal{S} \rightarrow \mathbb{R}$
  - **Models:** $m : \mathcal{S} \rightarrow \mathcal{S}$ and/or $r : \mathcal{S} \rightarrow \mathbb{R}$
  - **State update:** $u : \mathcal{S} \times \mathcal{O} \rightarrow \mathcal{S}$
- 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

Exploration and Exploitation

- Reinforcement learning is like **trial-and-error learning**
- The agent should discover a **good policy**
- ...from its experiences of the environment
- ...without losing too much reward along the way
- **Exploration** finds more information about the environment
- **Exploitation** exploits known information to maximize reward
- It is usually important to explore as well as exploit

Examples for Exploration and Exploitation

- **Restaurant Selection**
  - Exploitation Go to your favorite restaurant
  - Exploration Try a new restaurant
- **Online Banner Advertisements**
  - Exploitation Show the most successful advert
  - Exploration Show a different advert
- **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

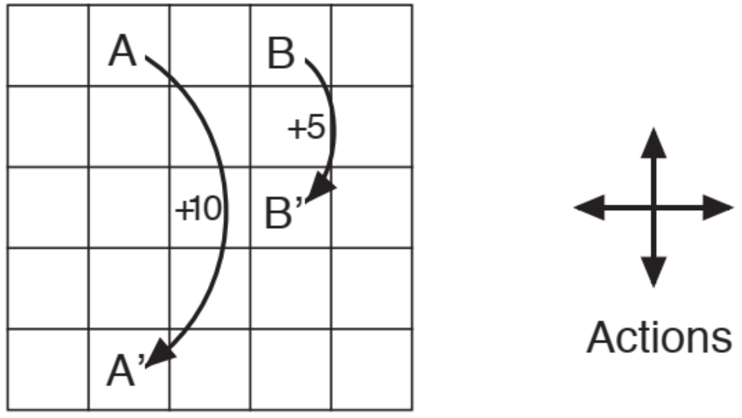
Prediction and Control

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

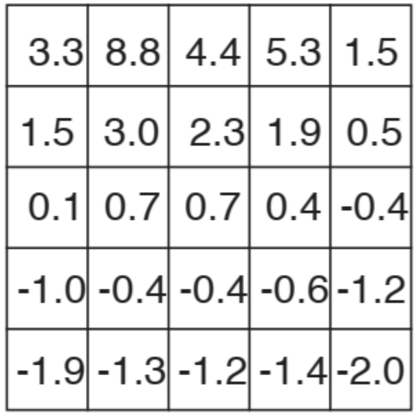
$$\pi_*(s) = \underset{\pi}{\operatorname{argmax}}\, v_\pi(s)$$

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

Gridworld Example: Prediction



(a)

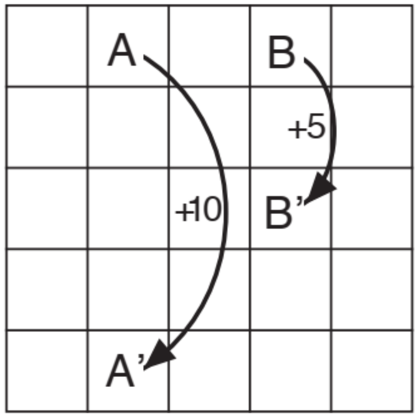


3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

- Reward is -1 when bumping into a wall, $\gamma = 0.9$
- What is the value function for the uniform random policy?

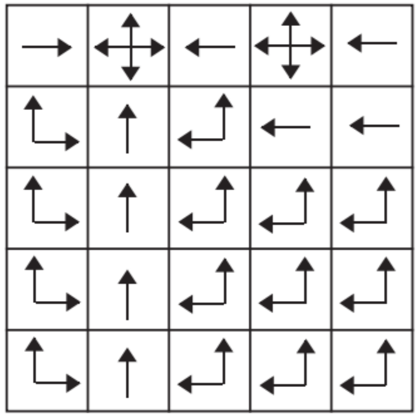
Gridworld Example: Control



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?

Thank you!
Q/A