

Reinforcement Learning (AI644)

Prepared by Dr. Ashraf Elsayed

Lecture 1





مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY

Reinforcement Learning

Lecture 1

Prepared by
Dr. Ashraf Elsayed

www.midocean.ae

Introduction to Reinforcement Learning

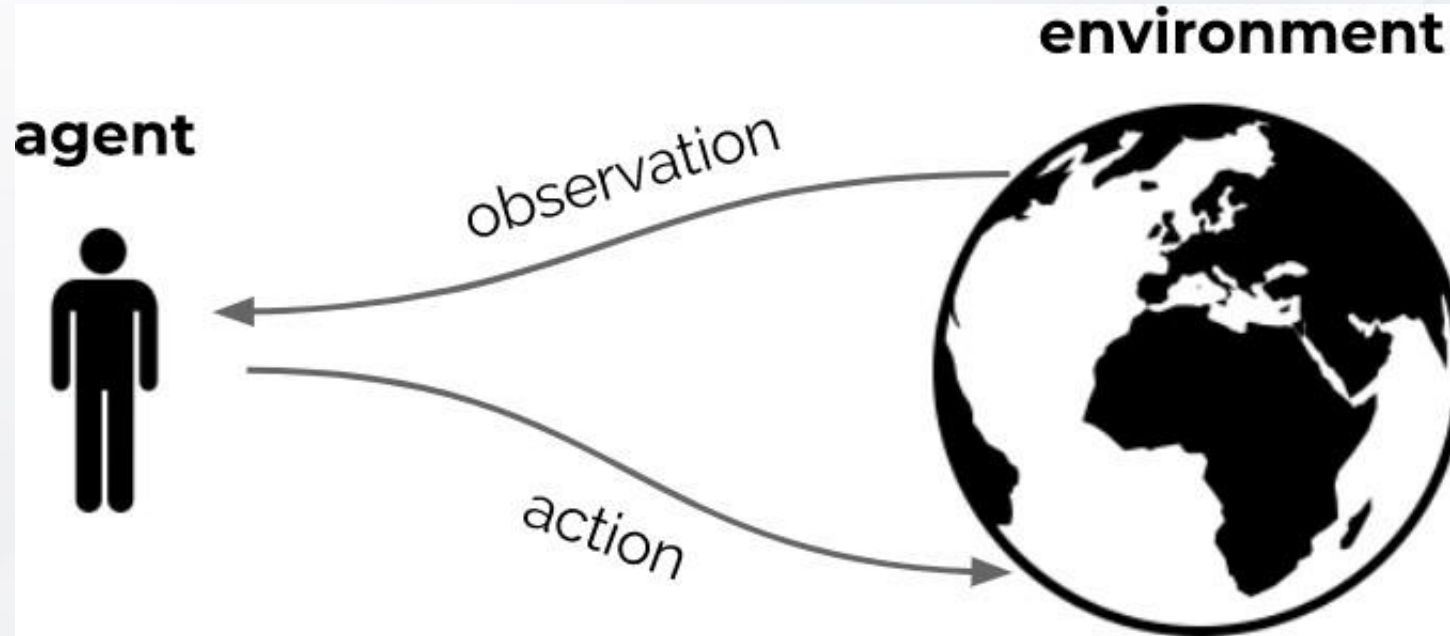
What is Reinforcement Learning?

- RL is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent takes actions in the environment, and based on these actions, it receives feedback in the form of rewards or penalties. The goal of the agent is to learn a policy, which is a mapping from states to actions, that maximizes the cumulative reward over time.

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 optimize some **reward signal**
- Framework of **learning to make decisions** from **interaction**

The interaction loop



Goal: optimize 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, maximize 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:

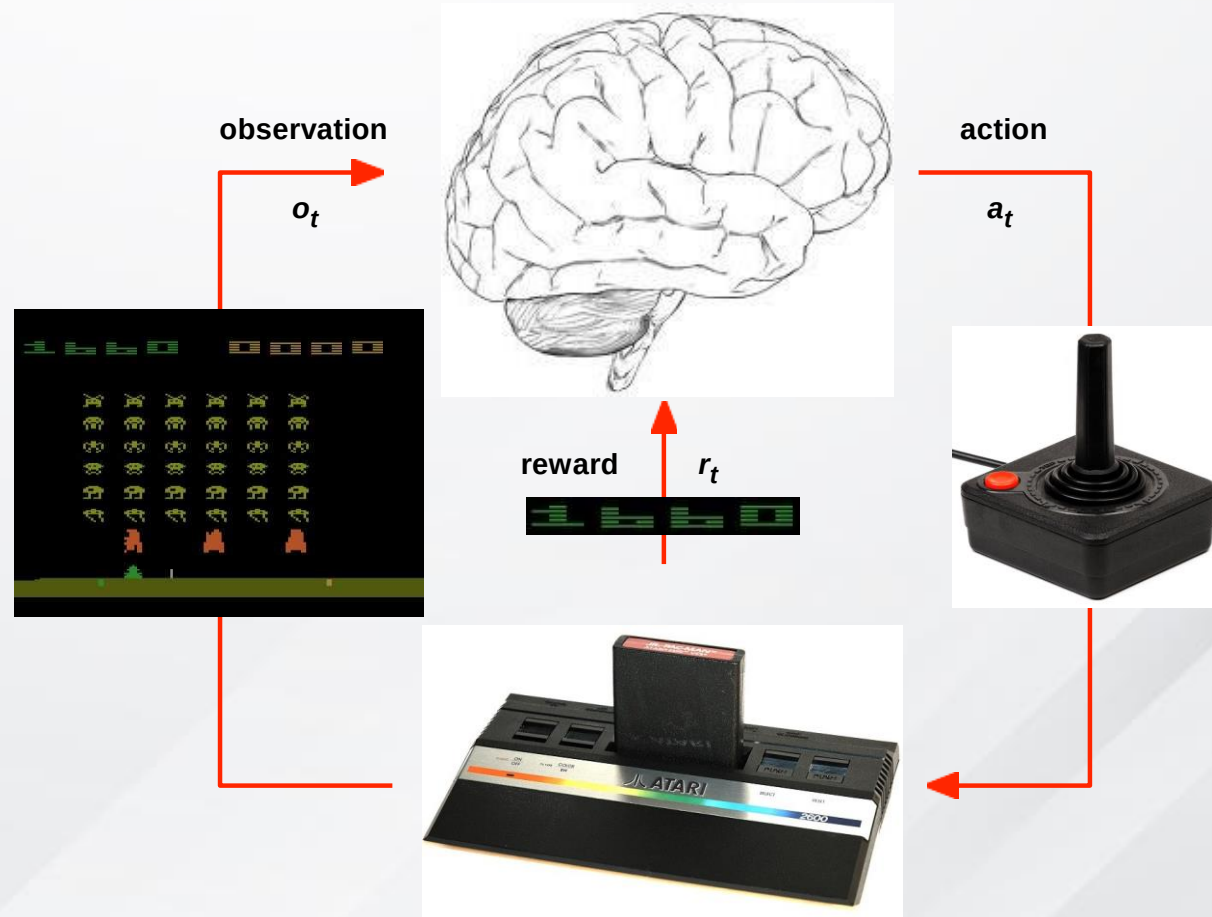
1. Find **solutions**

- A program that plays chess really well
- A manufacturing robot with a specific purpose

2. **Adapt online**, deal with unforeseen circumstances

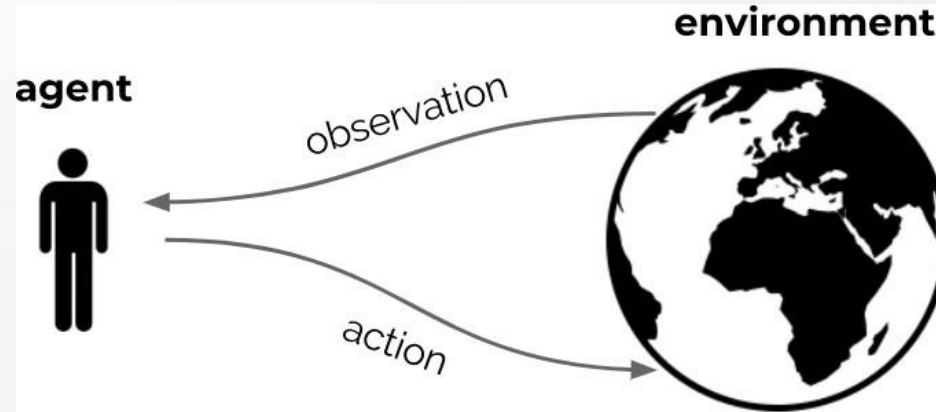
- A chess program that can learn to adapt to you
- A robot that can learn to navigate unknown terrains
- Reinforcement learning can provide algorithms for both cases
- Note that the second point is not (just) about generalization — it is about continuing to learn efficiently online, during operation

Atari Example: Reinforcement Learning



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

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

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

- 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**

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

- The value depends on the actions the agent takes
- Goal is to **maximize value**, by picking suitable actions
- Returns and values can be defined recursively

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

Maximizing value by taking actions

- Goal: **select actions to maximize 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**:

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a]$$

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

Core concepts

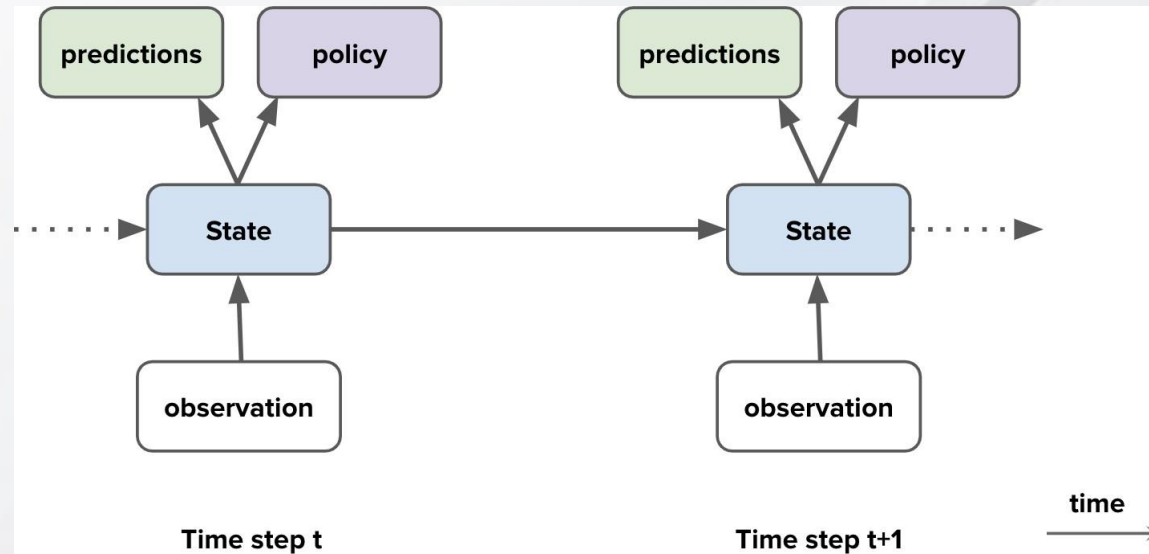
The reinforcement learning formalism includes

- **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

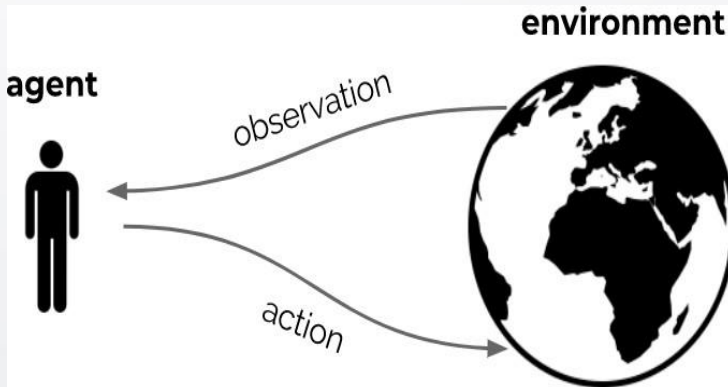
Agent components

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

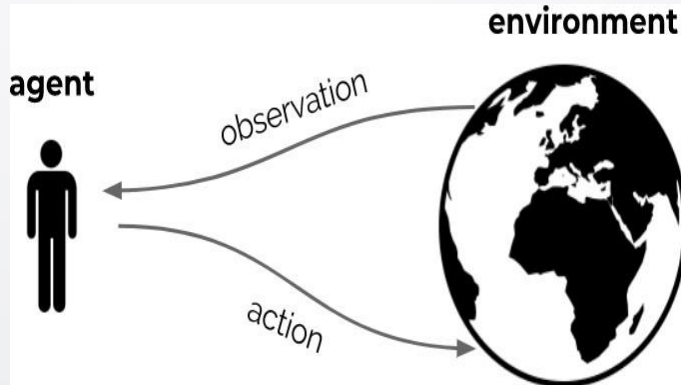
Agent State

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

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

- 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

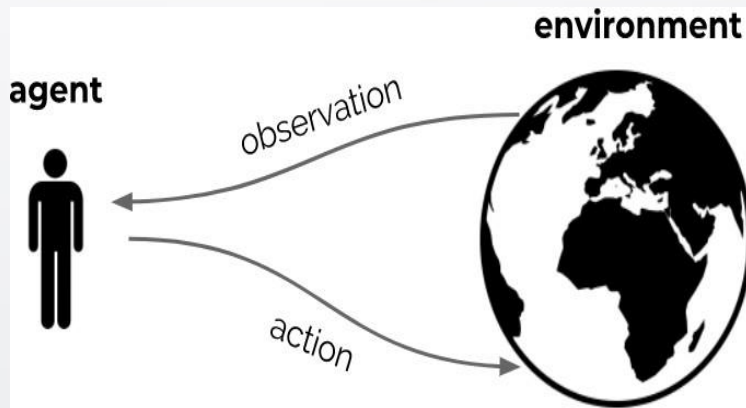
$$p(r, s \mid S_t, A_t) = p(r, s \mid H_t, A_t)$$

- 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
- 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 state**

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 maze



Agent State

A potential observation



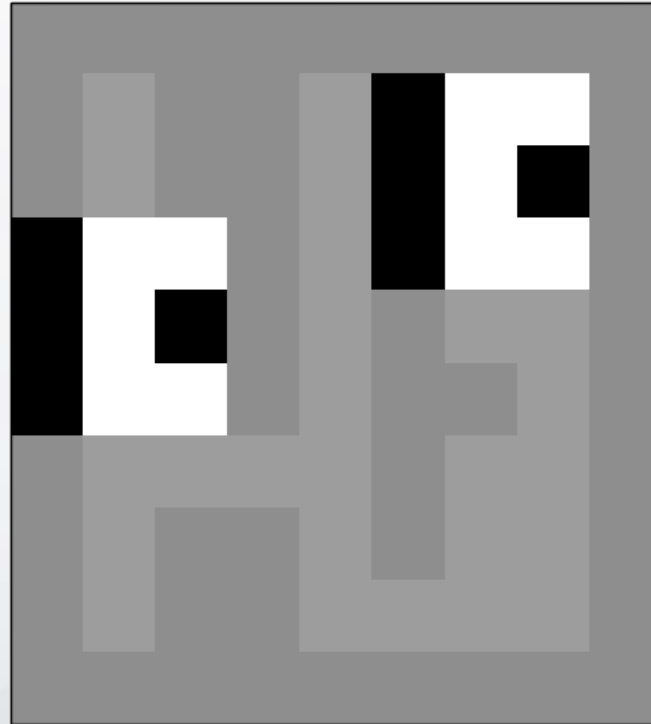
Agent State

An observation in a different location



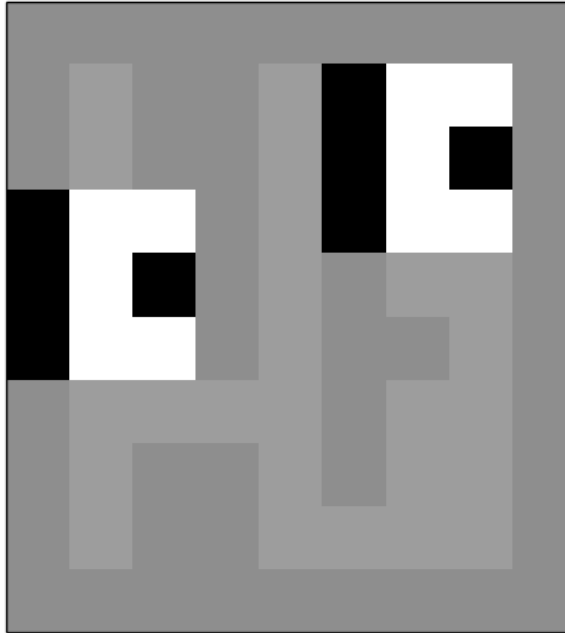
Agent State

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: $S_t = H_t$ (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

Agent components

Agent components

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

Policy

- A **policy** defines the agent's behavior
- 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

$$v_{\pi}(s) = E[G_t \mid S_t = s, \pi]$$

$$= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s, \pi]$$

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

Value Function

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

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

Here $a \sim \pi(s)$ means a is chosen by policy π in state s

- This is known as a **Bellman equation** (Bellman 1957)
- A similar equation holds for the optimal (=highest possible) value:

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

This does **not** depend on a policy

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

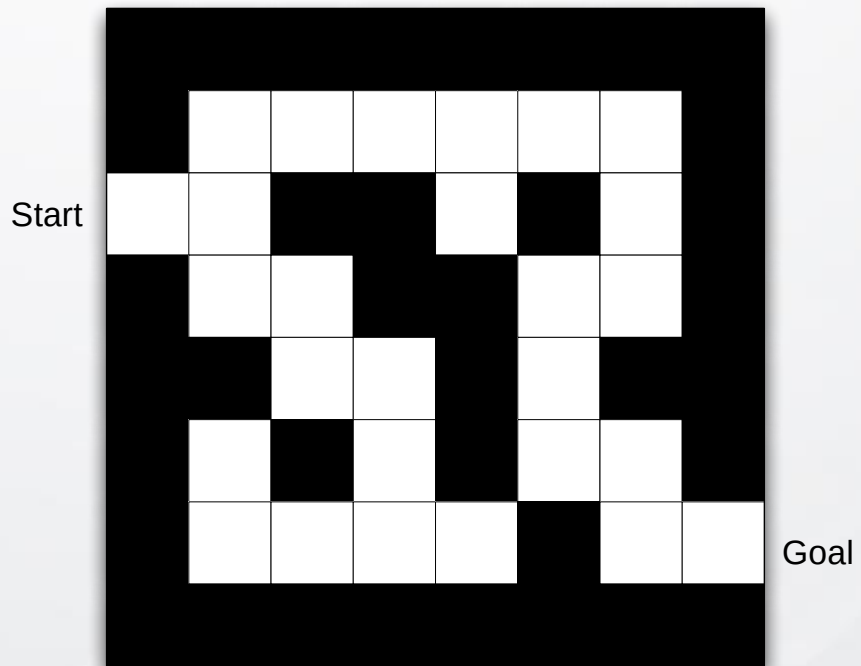
$$\mathcal{P}(s, a, s') \approx p(S_{t+1} = s' \mid S_t = s, A_t = a)$$

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

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

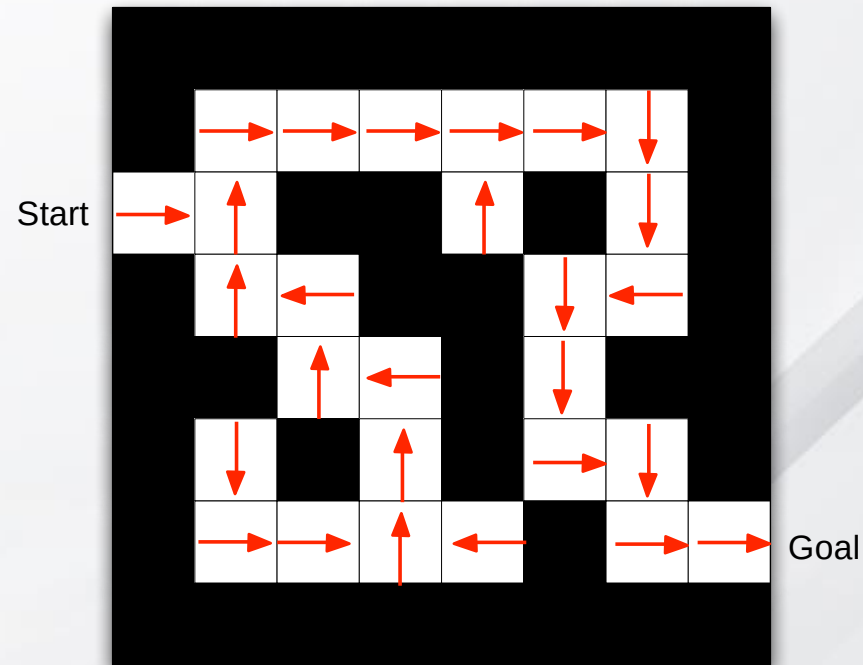
- A model does not immediately give us a good policy - we would still need to plan
- We could also consider **stochastic** (**generative**) models

Maze Example



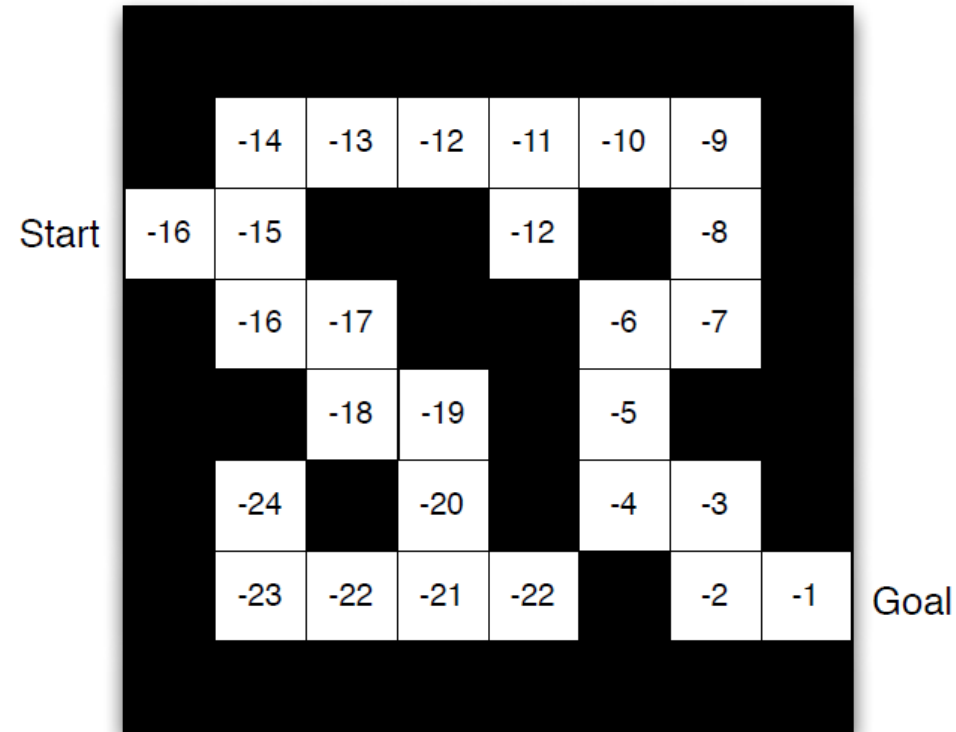
- ▶ Rewards: -1 per time-step
- ▶ Actions: N, E, S, W
- ▶ States: Agent's location

Maze Example: Policy



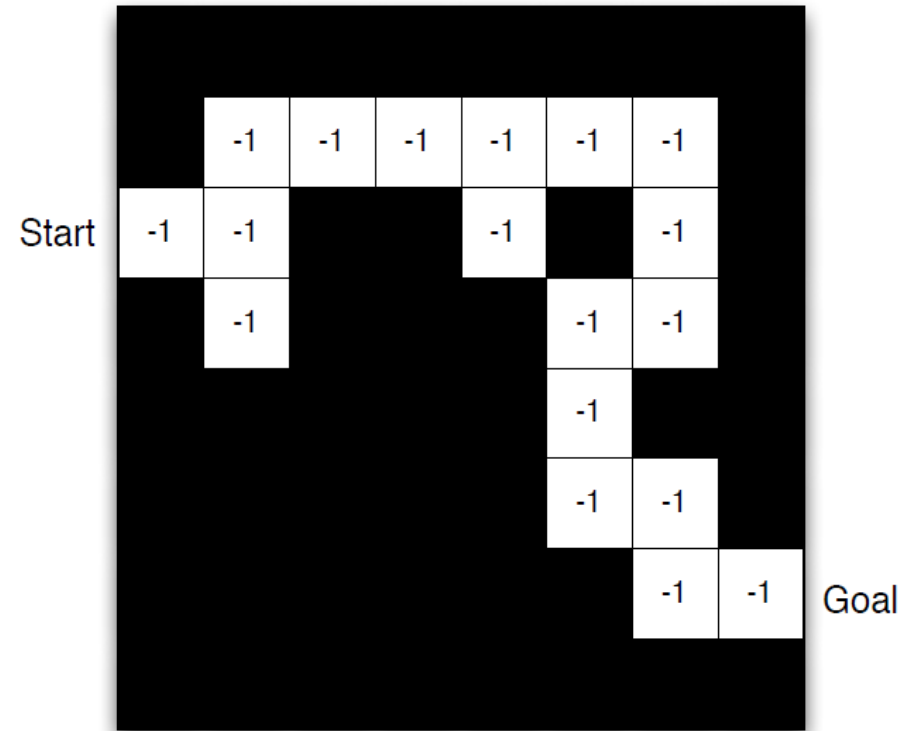
- ▶ Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function



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

Maze Example: Model



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

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

Learning Agent Components

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



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY



Thank You

