Reinforcement Learning

ANNA REALI, FELIPE LENO, REINALDO BIANCHI

Advanced Institute for Artificial Intelligence

Content

- Introduction
- MDP and Policies
- Reinforcement Learning and Q-learning
- Deep RL and DQN
- Policy Gradient Methods
- Applications

Introduction

Learning

- Learning modifies the agent's decision mechanisms to improve performance

- Learning is essential for unknown environments
  - i.e., when designer lacks omniscience

- Learning is useful as a system construction method
  - i.e., expose the agent to reality rather than trying to write it down

Learning Agent

- Design of a learning agent depends on what feedback is available
- Type of feedback:
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

Supervised Learning

- Consider a database containing records (X, Y).
  - Variables $X = \{X_1, . . . , X_n\}$ are observed; they are called features or attributes.
  - Labels are values of a class variable Y.
- When **every** record contains a **label**, we have **supervised learning**.
- So, learning here is to produce a function **g** using data:
  - $\hat{Y}$ = **g**($X_1, . . . , X_n$). ***Goal***: $\hat{Y}$ = Y

Unsupervised Learning

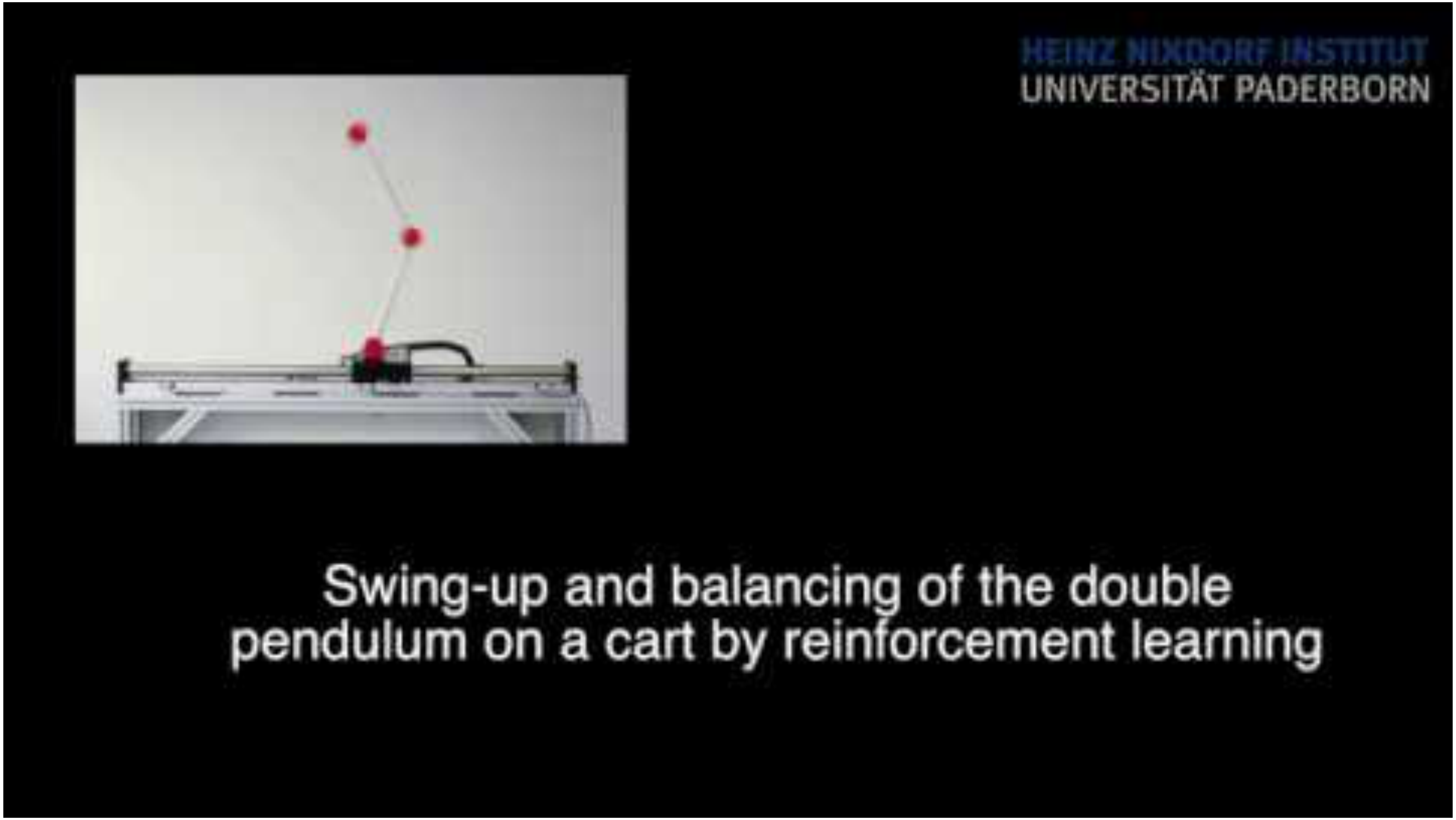
- When every label Y is missing, we have **unsupervised learning**.
  - It is typically about **finding structure hidden** in collections of unlabeled data.

- General case: some labels missing: **semi-supervised learning**.

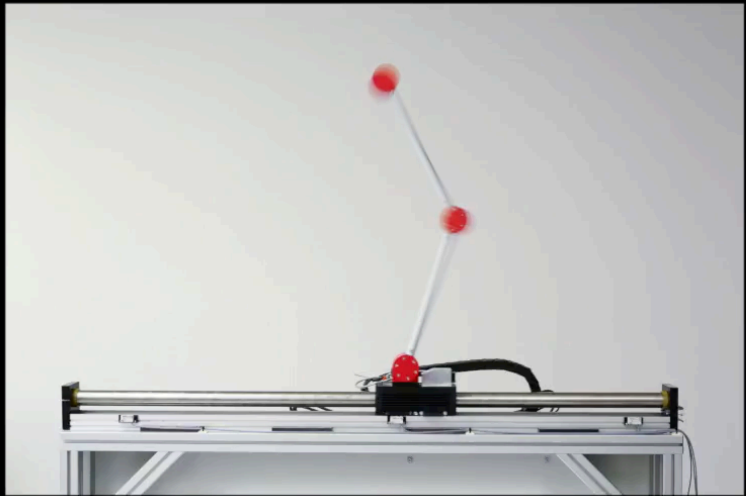
Reinforcement Learning

- Feedback is delayed, occasional
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives
- Trial and error learning (via experiences)

- **Task**: to learn from this **indirect**, **delayed reward**, to choose **sequences of actions** that produce the **greatest cumulative reward in the long run**


HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN
Swing-up and balancing of the double pendulum on a cart by reinforcement learning

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN



Swing-up and balancing of the double pendulum on a cart by reinforcement learning


TUDelft
DCSC

When we apply RL?

- RL addresses the question of **how** an autonomous agent that **senses** and **acts** in its environment can learn to **choose optimal actions** to **get as much reward** as it can over the **long run**.

➔ Applied to **Sequential Decision Problems**

Elements of RL

- **Reward signal**: indicates what is good in an immediate sense
- **Goal:** should specify what we want to achieve, not how we want to achieve it.
- **Value function**: specifies what is good in the long run.
- **Action choices** are made based on value judgments. We seek actions that bring about states of highest value.

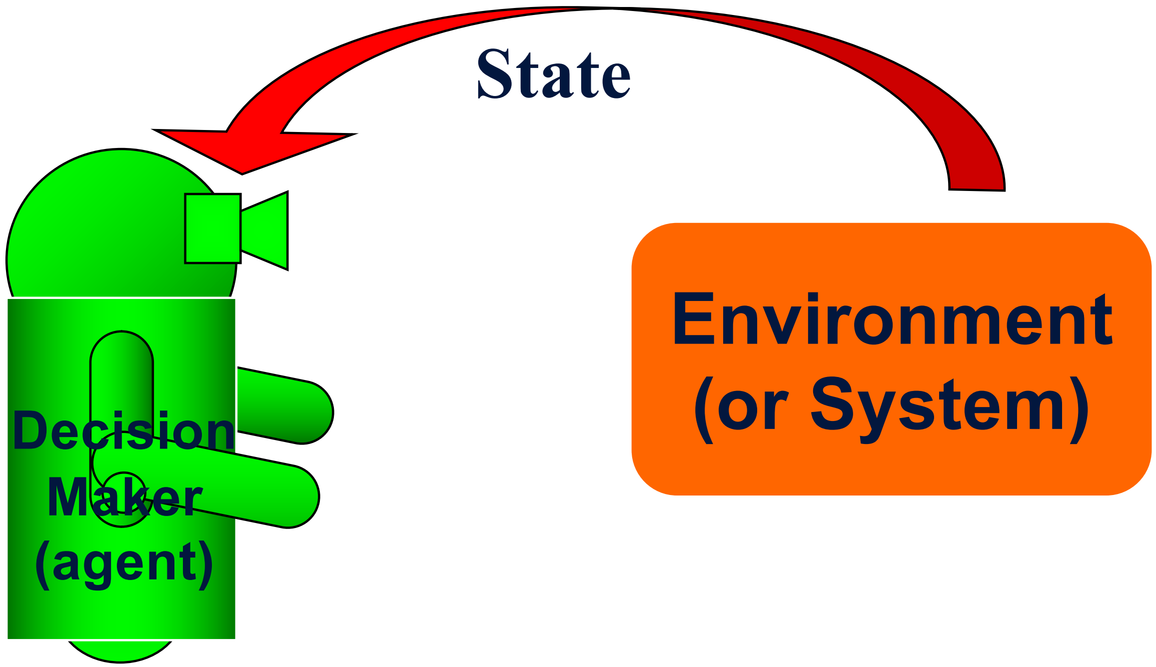
MDP and Policies

Sequential decision problem

s_0

At each time step the decision maker:

1. Observes the state of the system;

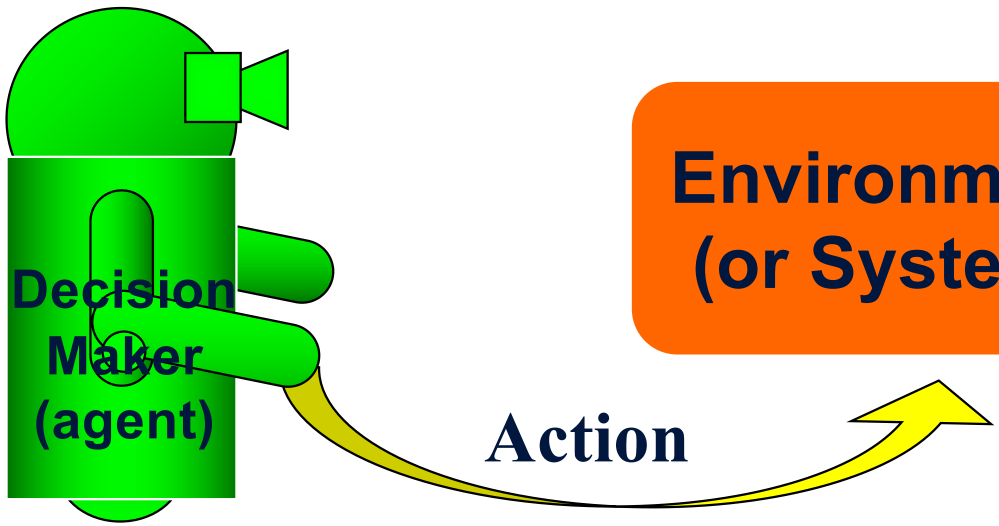


Sequential decision problem

$s_0 \xrightarrow{a_0}$

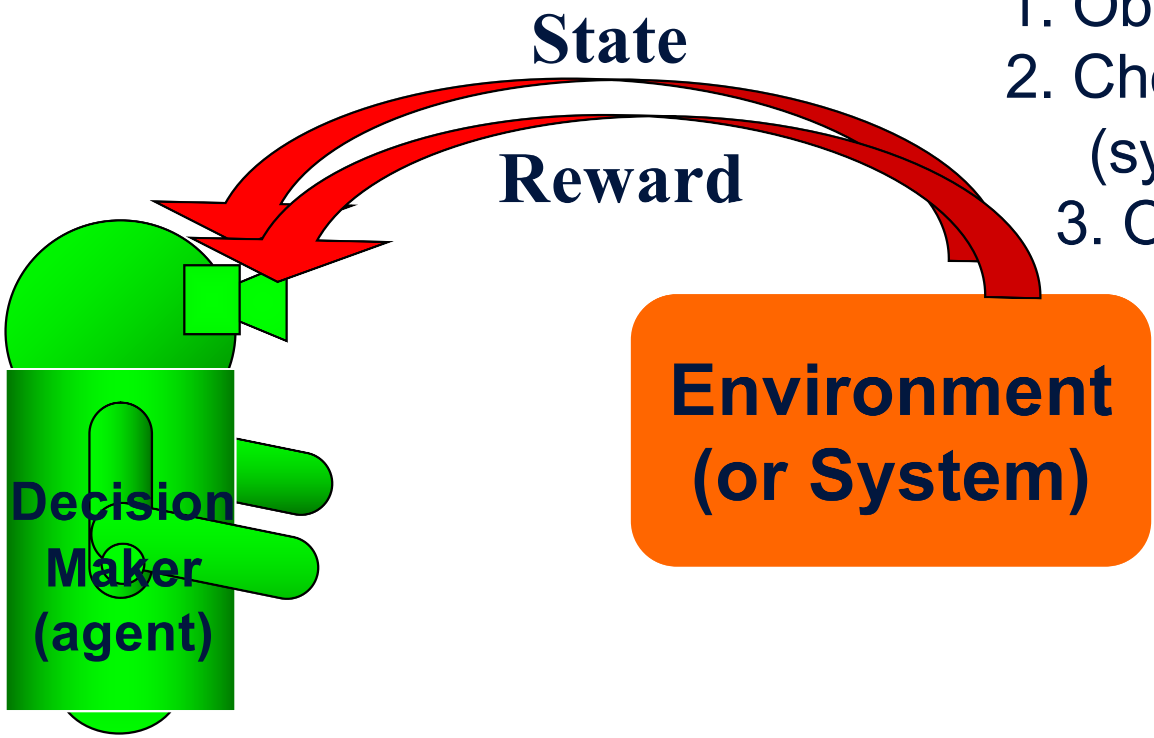
At each time step the decision maker:

1. Observes the state of the system;
2. Chooses an action and applies it;
   (system evolves to a new state)



Sequential decision problem

$s_0 \xrightarrow[r_0]{a_0} s_1$

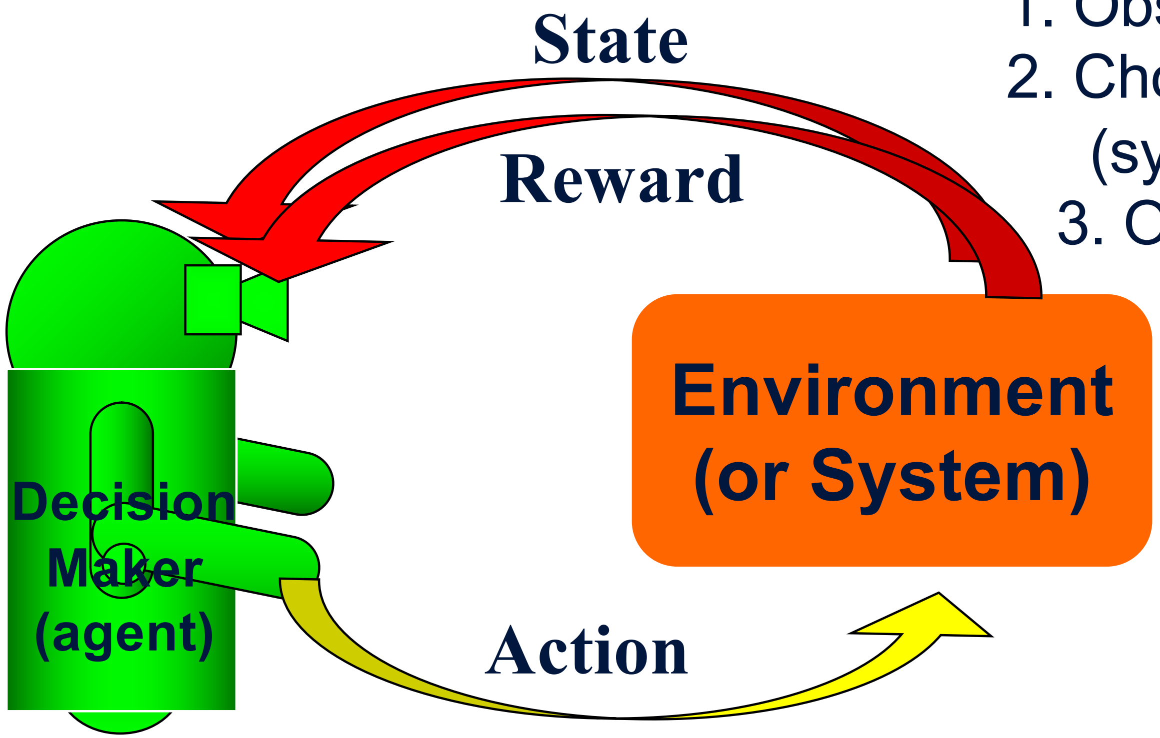


At each time step the decision maker:

1. Observes the state of the system;
2. Chooses an action and applies it;
   (system evolves to a new state)
3. Observes the new state and an immediate reinforcement;

Sequential decision problem

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \ \dots$$



At each time step the decision maker:

1. Observes the state of the system;
2. Chooses an action and applies it;
   (system evolves to a new state)
3. Observes the new state and an immediate reinforcement;

Repeat 1 – 3

We want learn how to map states to actions so as to maximize the expected sum of rewards.

MDP – Model Formulation

- An **MDP** is defined as <S, A, T, R >:
  - S is the set of possible states (arbitrary finite set);
  - A is the set of allowable actions (arbitrary finite set);
  - T: $S \times A \times S \rightarrow [0,1]$ is the transition probability function; $t(s,a,s') = P(s'|s,a)$ – the probability of transition from s to s' given action a
  - R: $S \times A \rightarrow \Re$ is the immediate reward function; $r(s,a)$ – the reward for taking action a in state s

State

- Experience is a sequence of observations, actions, rewards

$$o_1, a_1, r_1, o_2, a_2, r_2, o_3, \cdots o_t, a_t, r_t$$

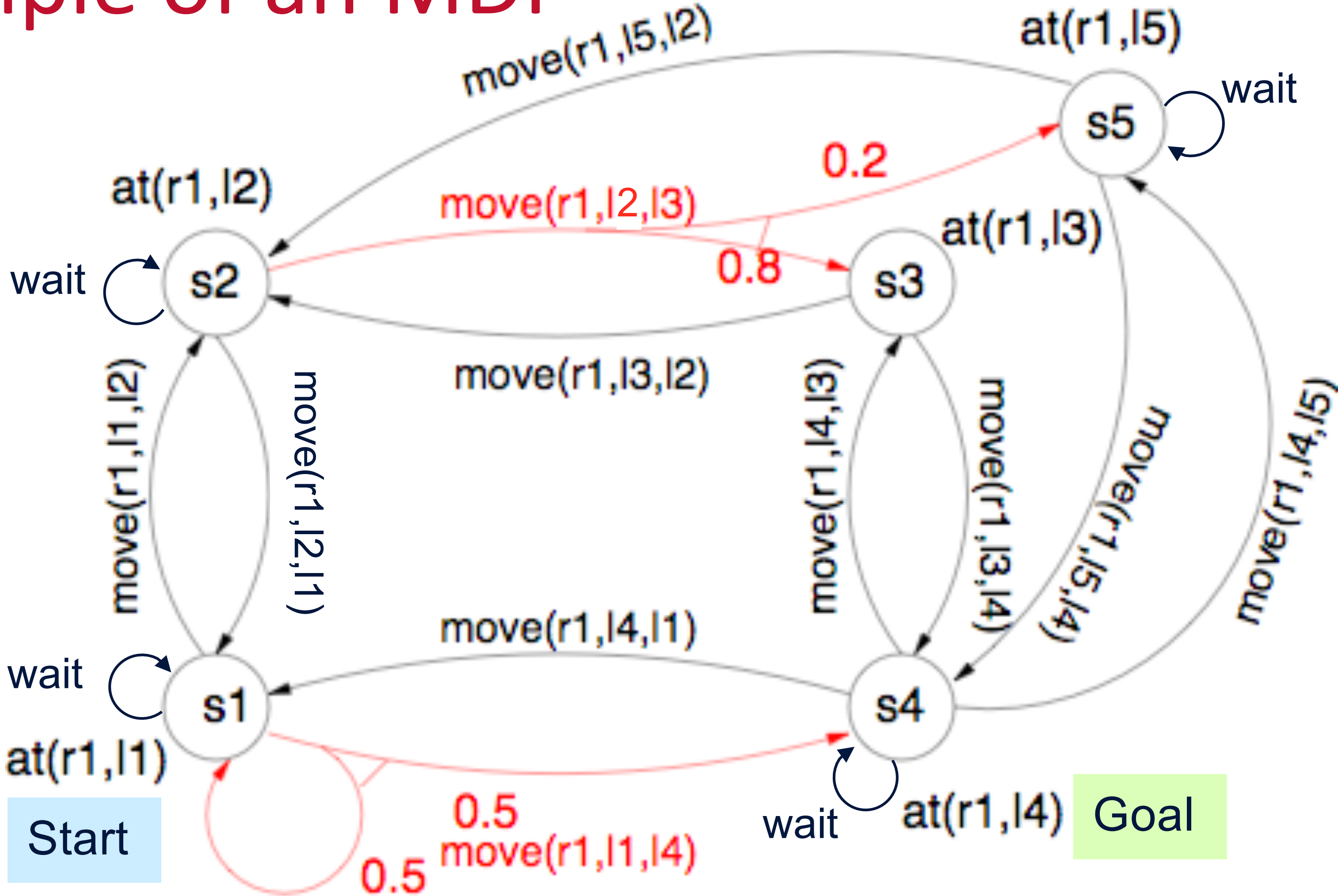
- The **state** is a summary of experience

$$s = f(o_1, a_1, r_1, o_2, a_2, r_2, o_3, \cdots o_t, a_t, r_t)$$

- In a **fully observed** environment

$$s = f(o_t)$$

Example of an MDP



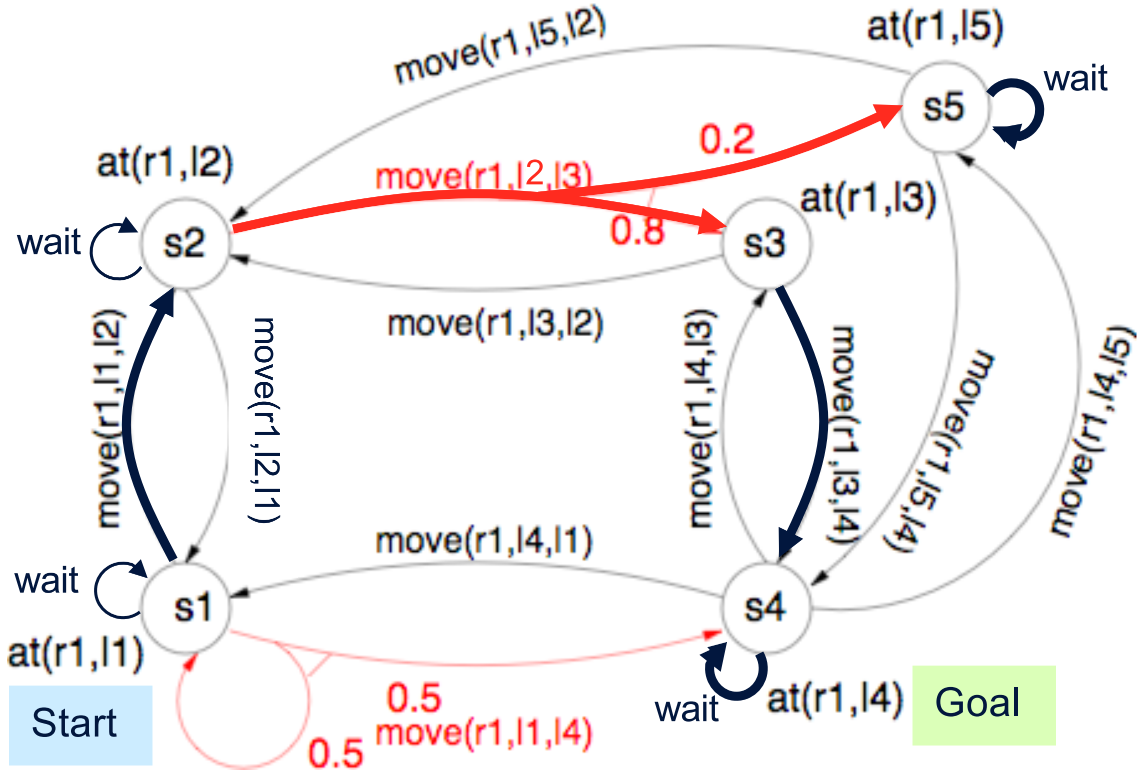
Example of an MDP

- A = {wait, move(r1,l1,l2), move(r1,l2,l1), move(r1,l4,l1), move(r1,l1,l4), move(r1,l3,l2), move(r1,l2,l3), move(r1,l5,l2), move(r1,l4,l3), move(r1,l3,l4), move(r1,l5,l4), move(r1,l4,l5)}
- S = {s1, s2, s3, s4, s5}
- t(s1,move(r1,l1,l4),s4)= t(s1,move(r1,l1,l4),s1)= 0.5; t(s2,move(r1,l2,l3),s3)=0.8; t(s2,move(r1,l2,l3),s5)=0.2; All others t(.) have a value of 1.
- r(s1,wait)= r(s2,wait)= -1; r(s4,wait)= 0; r(s5,wait)= -100; r(s4)=100;

r(s1,move(r1,l1,l2)) = r(s2,move(r1,l2,l1)) = r(s3,move(r1,l3,l4)) = -100;
r(s4,move(r1,l4,l3)) = r(s4,move(r1,l4,l5)) = r(s5,move(r1,l5,l4)) = -100;
r(s1,move(r1,l1,l4)) = r(s4,move(r1,l4,l1)) = r(s2,move(r1,l2,l3)) = -1;
r(s3,move(r1,l3,l2)) = r(s5,move(r1,l5,l2))= -1; r(s1)=r(s2)=r(s3)=r(s5)=0

Example

π_1 = {(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, wait)}



goal

$h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$
$h_2 = \langle$s1, s2, s5, s5 … $\rangle$

$P(h_1 \mid \pi_1) = 1 \times 1 \times 0.8 \times 1 \times \ldots = 0.8$

$P(h_2 \mid \pi_1) = 1 \times 1 \times 0.2 \times 1 \times \ldots = 0.2$

$P(h \mid \pi_1) = 0$ for all other h

Returns: we want to <u>maximize</u> the **expected return**

- **Discounted return**:

$$R_t = r_t + \gamma r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

where $\gamma, 0 \leq \gamma \leq 1$, is **the discount rate**.

- γ describes the preference of an agent for <u>current</u> reinforcements over <u>future</u> reinforcements.

Value Function V(s)

- The **value of a state** $V^{\pi}(s)$ under a policy π is the expected return when starting in that state s and following the policy π from that state onwards:

$$V^{\pi}(s) = E_{\pi}[R_t | s_t = s] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right]$$

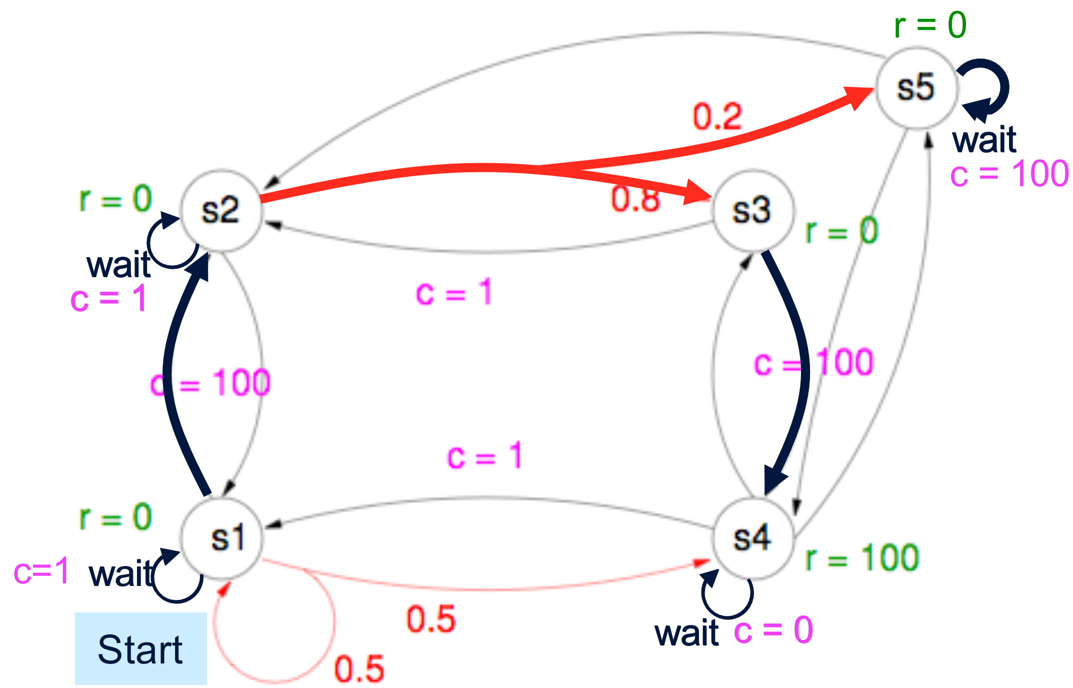
Value Function Q(s,a)

- The **value of a state-action pair** $Q^{\pi}(s, a)$ is the expected return starting from that state s, taking that action a, and thereafter following policy π:

$$Q^{\pi}(s, a) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a\right]$$

Example

π_1 = {(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, wait)}



$\gamma = 0.9$

$h_1 = \langle$**s1, s2, s3, s4, s4, …** $\rangle$

$V_{\pi 1}(h_1) = .9^0(0 - 100) + .9^1(0 - 1) + .9^2(0 - 100) + .9^3\,100 + .9^4\,100 + \ldots = \mathbf{547.9}$

$h_2 = \langle$**s1, s2, s5, s5 …** $\rangle$

$V_{\pi 1}(h_2) = .9^0(0 - 100) + .9^1(0 - 1) + .9^2(-100) + .9^3(-100) + \ldots = \mathbf{-910.1}$

$\boldsymbol{E[V_{\pi 1}(h)] = 0.8 \times 547.9 + 0.2 \times (-910.1) = 256.3}$

Optimal Value Functions

- Optimal state-value function:

$$V^*(\mathrm{s}) = \max_{\pi} V^{\pi}(s), \forall s \in S$$

- Optimal action-value function:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \forall s \in S, \forall a \in A$$

$$\boldsymbol{V^*(s) = \max_{a} Q^*(s,a)}$$

MDP – Solution: an optimal policy

- **Solution:**

$$\boxed{\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a)}$$

- **Q*(s,a) = r(s,a) + γ V*(s')**, $\forall s, s' \in S, \forall a \in A$

Policy

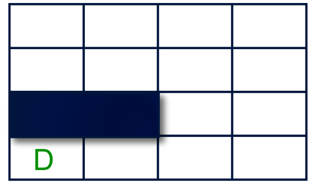
- A policy is the agent's behavior.
  - **Deterministic policy**

$$\pi: S \rightarrow A \Rightarrow \pi(s) = a$$

  - **Stochastic policy**

$$\pi: S \times A \rightarrow [0,1] \Rightarrow \pi(a|s) = P(a|s)$$

Example

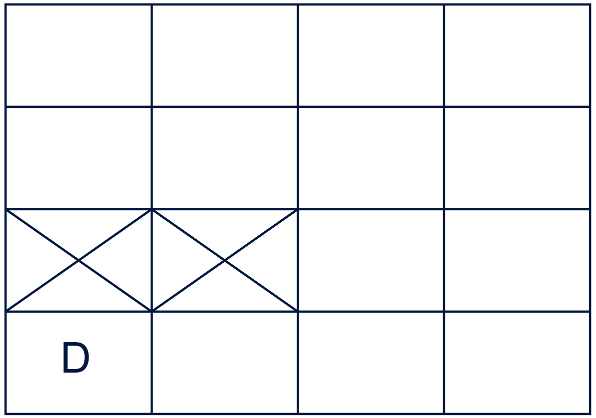


- Discreet 4x4 environment with obstacles.
- Agent must reach destination D from anywhere in the environment.
- D is an **absorbing state**: $V^*(D) = 0$, $A(D)=\{ \}$
- Actions: Up, Down, Left, Right
- Penalty for performing an action (any) = −1 (reward)
  - **Better** policy => **shorter** path
- Example for $\gamma = 1$ and **deterministic** MDP

Example

D			

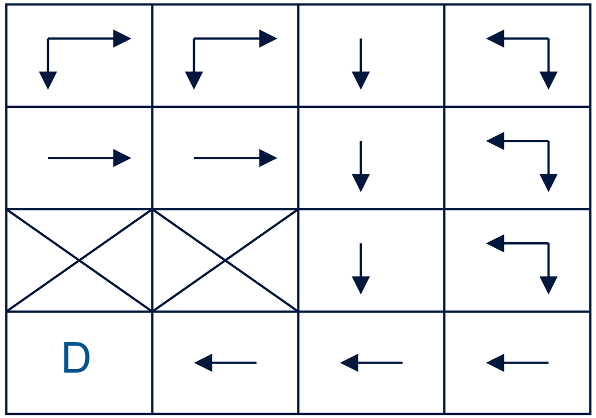
Example



Environment

-7	-6	-5	-6
-6	-5	-4	-5
		-3	-4
D 0	-1	-2	-3

Optimal value function: indicates the expected penalties until reaching the destination, following an optimal policy.



Optimal policies

Reinforcement Learning

RL Method

In ***Reinforcement Learning*** (RL), we would like an agent to **learn** to behave well in an MDP world, but **without knowing** anything about **T** or **R** when it starts out

We have to sample experiences and learn by trial and error.

RL elements

- **Policy π**: decision on what action to do in state s
- **Reward (or reinforcement) function**: defines goal, and good and bad (immediate) experience for learner
- **Value function**: estimate of total future reward
- **Model of the environment**: maps states and actions onto states

RL Method

- **Reward function**
- **Environmental model**

fixed <u>external</u> to agent

- **Policy**
- **Value function**
- **Estimate of model**

adjusted <u>during</u> learning

Design of an RL Agent

- When designing an RL agent, we need to:
  - define the reward function, which gives indications of what one wants
  - choose an RL method:
    - Model-free × Model-based
    - Off-policy × On-policy
    - Value-based × Policy-based

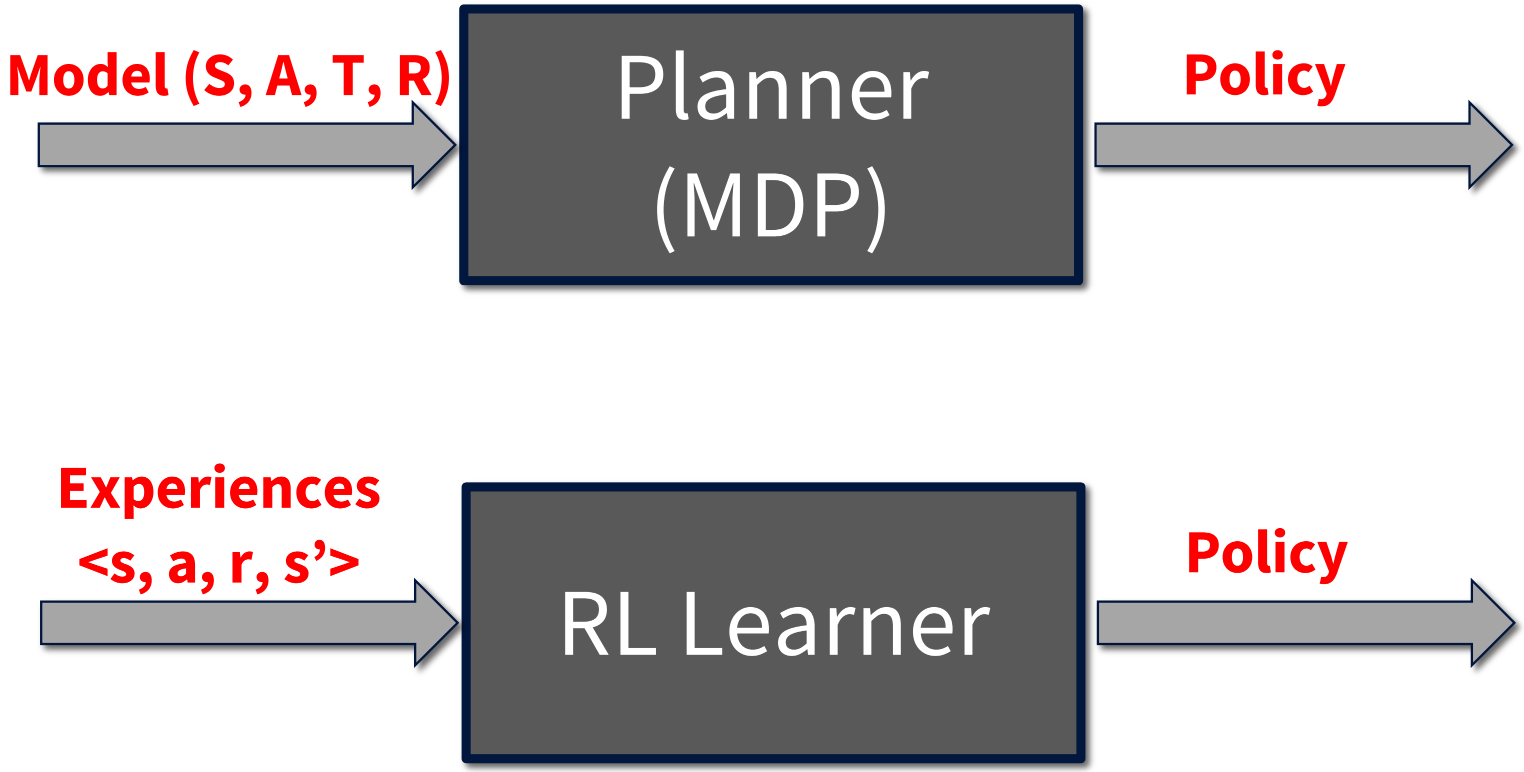
Reward Design

We need rewards to guide the agent to achieve its goal

- **Option 1**: Hand-designed reward functions
  - This requires a lot of experience and sensitivity (and luck!)
- **Option 2**: Learn rewards from demonstrations
  - Instead of having a human expert tune a system to achieve the desired behavior, the expert can demonstrate desired behavior and the agent can tune itself to match the demonstration

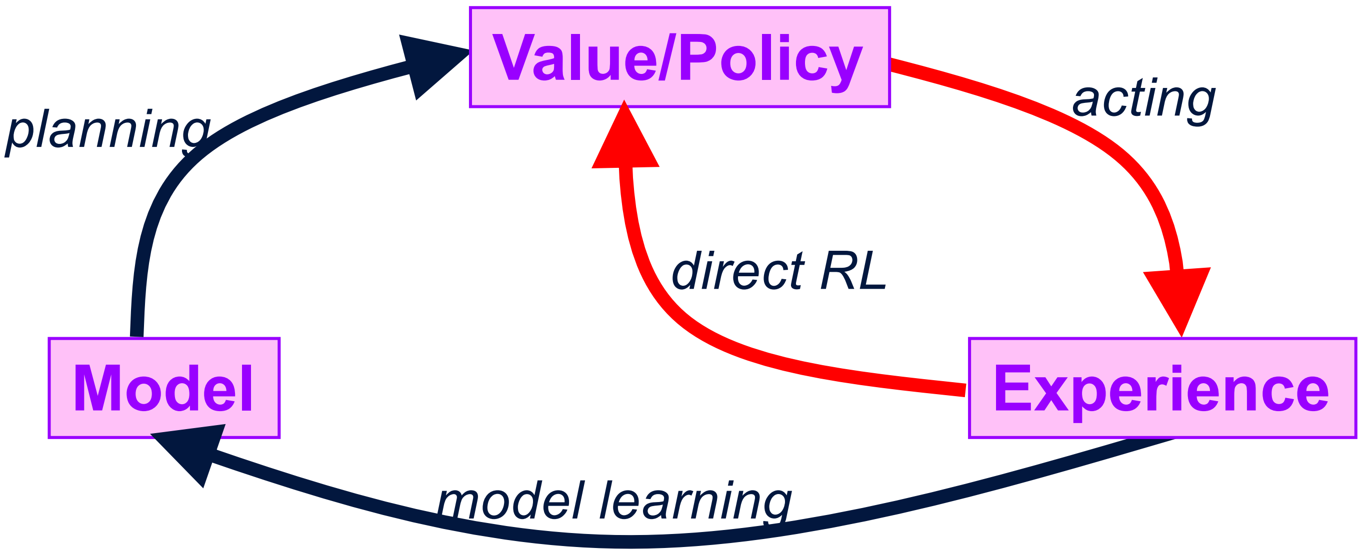
Model-free versus Model-based

- A model of the environment allows inferences to be made about **how** the environment will behave
- Model-based methods use models and planning.
  - Models are used for planning, which means deciding on a course of action by considering possible future situations before they are experienced
- Model-free methods learn exclusively from trial-and-error


Model (S, A, T, R)
Planner
(MDP)
Policy
Experiences
<s, a, r, s’>
RL Learner
Policy

RL and Planning

- Planning in RL interleaves cycles of learning based on experience in the world and experience gained via using the model to predict what will happen.



On-policy versus Off-policy

- An **on-policy** agent learns only about the policy that it is executing

- An **off-policy** agent learns about a policy or policies different from the one that it is executing

RL Methods

- **Model-based RL**
  - Build a model of the environment
  - Then plan using model
- **Value-based RL**
  - Estimate the optimal value function $Q^*(s, a)$
  - Then calculate the optimal policy π^* using Q^*
- **Policy-based RL**
  - Search directly for the optimal policy π^*

Credit Assignment Problem

- Given a sequence of states and actions, and the final sum of time-discounted future rewards, how do we infer which actions were effective at producing lots of reward and which actions were not effective?
- How do we assign credit for the observed rewards given a sequence of actions over time?

- **Every RL algorithm must address this problem**

Q-Learning

A simple algorithm: Q-learning

- **Key idea:**
  - Update the action-value function Q(s,a) **using the experience sequences**
  - Then use Q(s,a) to estimate $\boldsymbol{\pi}$

- **Characteristics:**

  Model-free, value-based, off-policy

Q-learning

```
Initialize Q(s,a) arbitrarily
Observe the current state s_t
do forever
   select an action a_t and execute it in s_t
   receive immediate reward r(s_t,a_t)
   observe the new state s_t+1
   update Q(s,a) as follows:
```

$$Q_{t+1}(s_t,a_t) \leftarrow (1-\alpha)Q_t(s_t,a_t) + \alpha[r(s_t,a_t) + \gamma \max_a Q_t(s_{t+1},a)]$$

$s_t \leftarrow s_{t+1}$

Q-learning

```
Initialize Q(s,a) arbitrarily
Observe the current state s_t
do forever
   select an action a_t and execute it in s_t
   receive immediate reward r(s_t,a_t)
   observe the new state s_t+1
   update Q(s,a) as follows:
```

$$Q_{t+1}(s_t,a_t) \leftarrow (1-\alpha)Q_t(s_t,a_t) + \alpha[r(s_t,a_t) + \gamma \max_a Q_t(s_{t+1},a)]$$

$s_t \leftarrow s_{t+1}$

Learning rate α

- The basic form of the update looks like this:

$$\mathbf{X}_{t+1} \leftarrow (1 - \alpha)\, \mathbf{X}_t + \alpha\, \boldsymbol{New}_t$$

  - We are updating our estimate of **X** to be mostly like our old value of **X** but adding in a new term ***New.***
  - It is a running <u>average</u> of the new terms received on each step.
  - It is quite typical (and, in fact, required for convergence), to <u>start with a large α</u>, and then <u>decrease it</u> over time.

Q-Learning

- There are **two iterative processes** going on:
  - One is the usual kind of **averaging** we do, when we collect a lot of samples and try to estimate their mean (using the learning rate).
  - The other is the **dynamic programming iteration** done by value iteration, <u>updating the value of a state based on the estimated values of its successors</u>.

Trade-off: Exploration vs. Exploitation

Which experimentation strategy produces most effective learning?

- The agent has to **exploit** what it has already experienced in order to obtain reward, but it also has to **explore** in order to gather new information.
- The dilemma is that neither exploration nor exploitation can be performed exclusively without failing at the task.

Trade-off: Exploration vs. Exploitation

- **ε-Greedy**:

$$a_t = \begin{cases} a_t^* & \text{with probability } 1 - \varepsilon \\ \text{random action} & \text{with probability } \varepsilon \end{cases}$$

exploitation

exploration

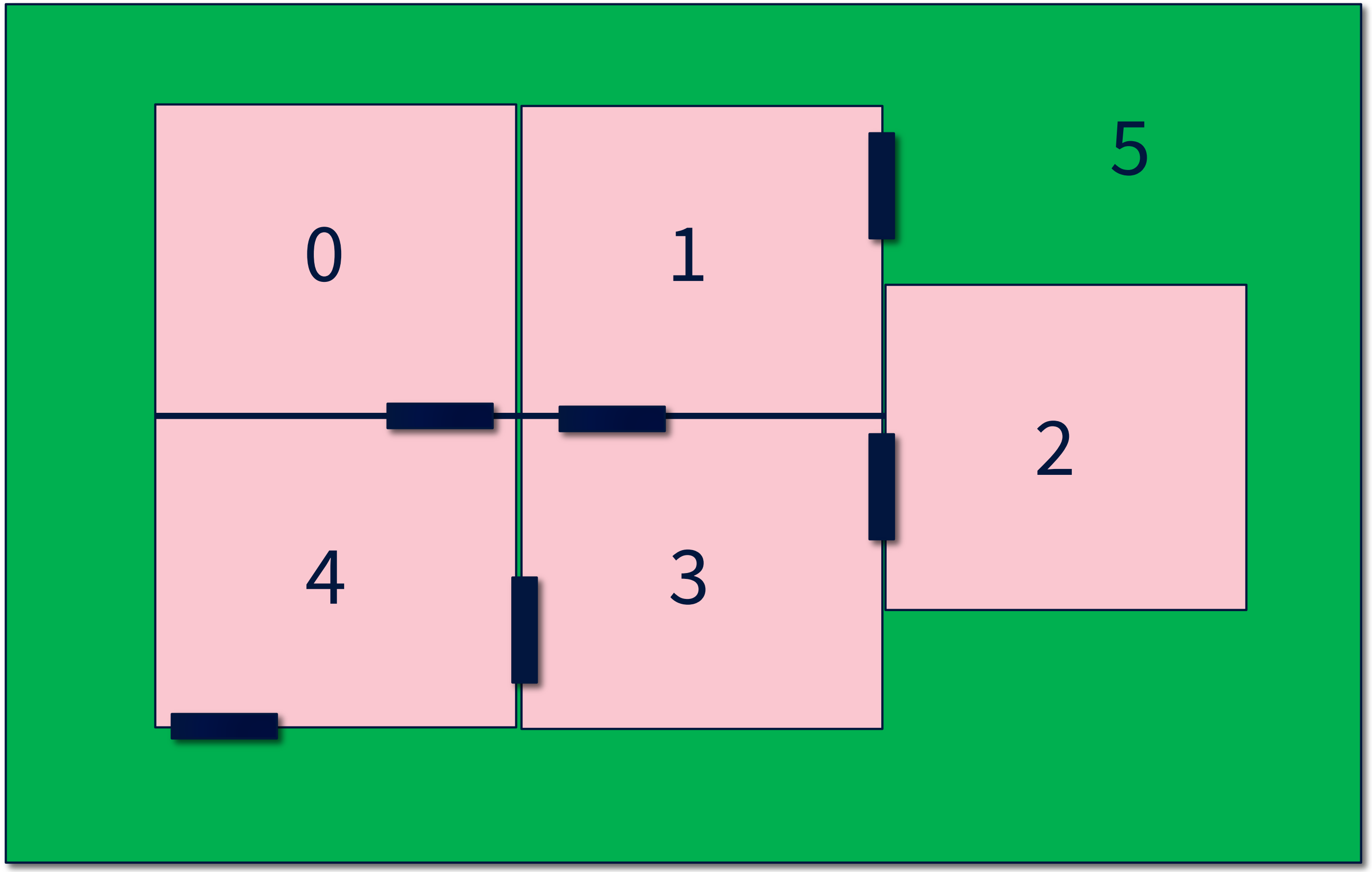
- **Softmax** action selection (Boltzmann distribution):

Choose *a* on *t* with probability:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}},$$

τ is the “computational temperature”.

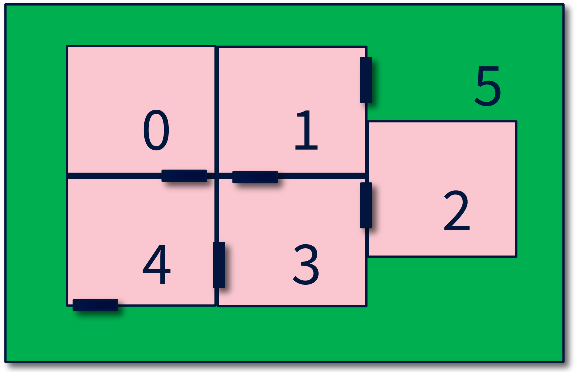
A Simple Example of Q-Learning



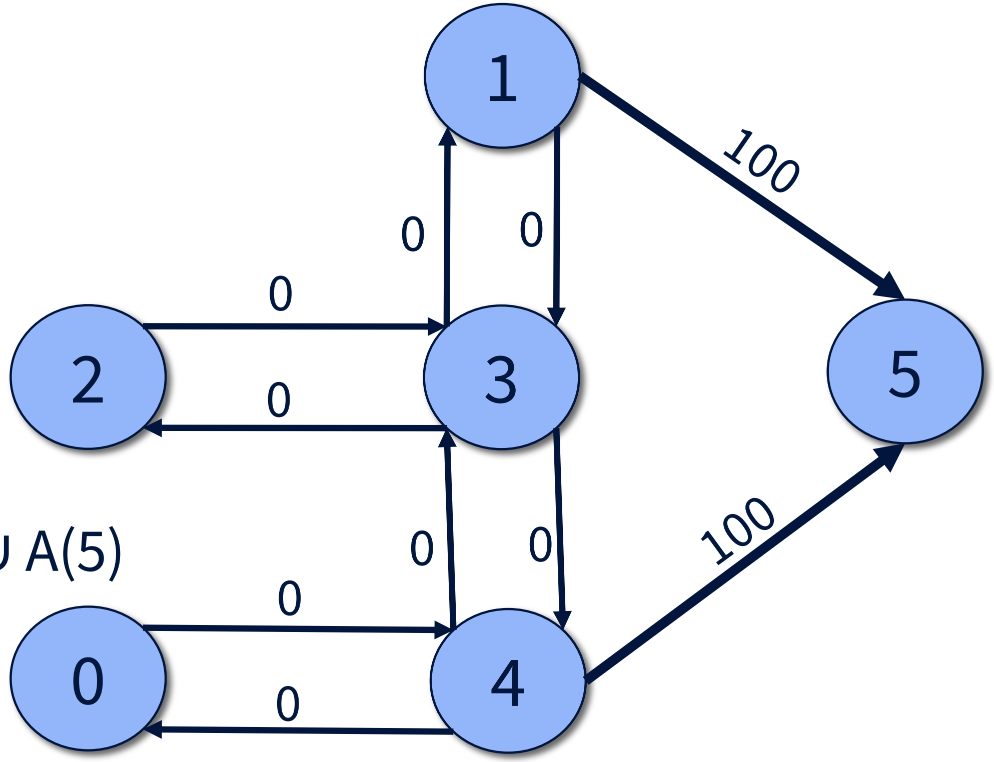
Deterministic world

Goal: to go outside the building

A Simple Example of Q-Learning



$\equiv$



MDP:

$S = \{0, 1, 2, 3, 4, 5\}$

$A = A(0) \cup A(1) \cup A(2) \cup A(3) \cup A(4) \cup A(5)$

T: $P(s'|s,a) = 1 \ \forall s, s' \in S, \forall a \in A(s)$

R: as indicated in the graph

A Simple Example of Q-Learning

- "Room" 5 is the goal – an absorbing state
- r(0,.,4) = r(1,.,3) = r(2,.,3) = r(3,.,1) = r(3,.,2) = 0;
  r(3,.,4) = r(4,.,0) = r(4,.,3) = r(5,.,5) = 0
  r(1,.,5) = r(4,.,5) = 100
- A(0) = {go to 4}; A(1) = {go to 3, go to 5};
  A(2) = {go to 3}; A(3) = {go to 1, go to 2, go to 4}
  A(4) = {go to 0, go to 3, go to 5}; A(5) = { }
- $\gamma = 0.8$; $\alpha = 1$ $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})_{t+1} = \boldsymbol{r}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \boldsymbol{\gamma} \max_{\boldsymbol{a}'} \boldsymbol{Q}(\boldsymbol{s}',\boldsymbol{a}')$

A Simple Example of Q-Learning

Suppose: **initial state = 1**
A(1) = {go to 3, go to 5}

Initial Q-table

	0	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

By random selection: **a = go to 5**
Experience = **(1, go to 5, 100, 5)**

$$Q(1,5) = 100 + 0.8 \max(Q(5,.))$$
$$= 100 + 0.8 * 0 = 100$$

	0	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	0	0	0	0	**100**
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

A Simple Example of Q-Learning

New episode: **state = 3**

A(3) = {go to 1, go to 2, go to 4}

Random selection: **a = go to 1**

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

Experience = **(3, go to 1, 0, 1)**

A(1) = {go to 3, go to 5}

Q(3,1) = 0 + 0.8 max(Q(1,3),Q(1,5))
= 0 + 0.8 * 100 = 80

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	**80**	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

A Simple Example of Q-Learning

Now **state = 1**

A(1) = {go to 3, go to 5}

ϵ-greedy: **a = go to 5**

	0	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

Experience = **(1, go to 5, 100, 5)**

A(5) = { }

Q(1,5) = 100 + 0.8 max(Q(5,.))
= 100 + 0.8 * 0 = 100

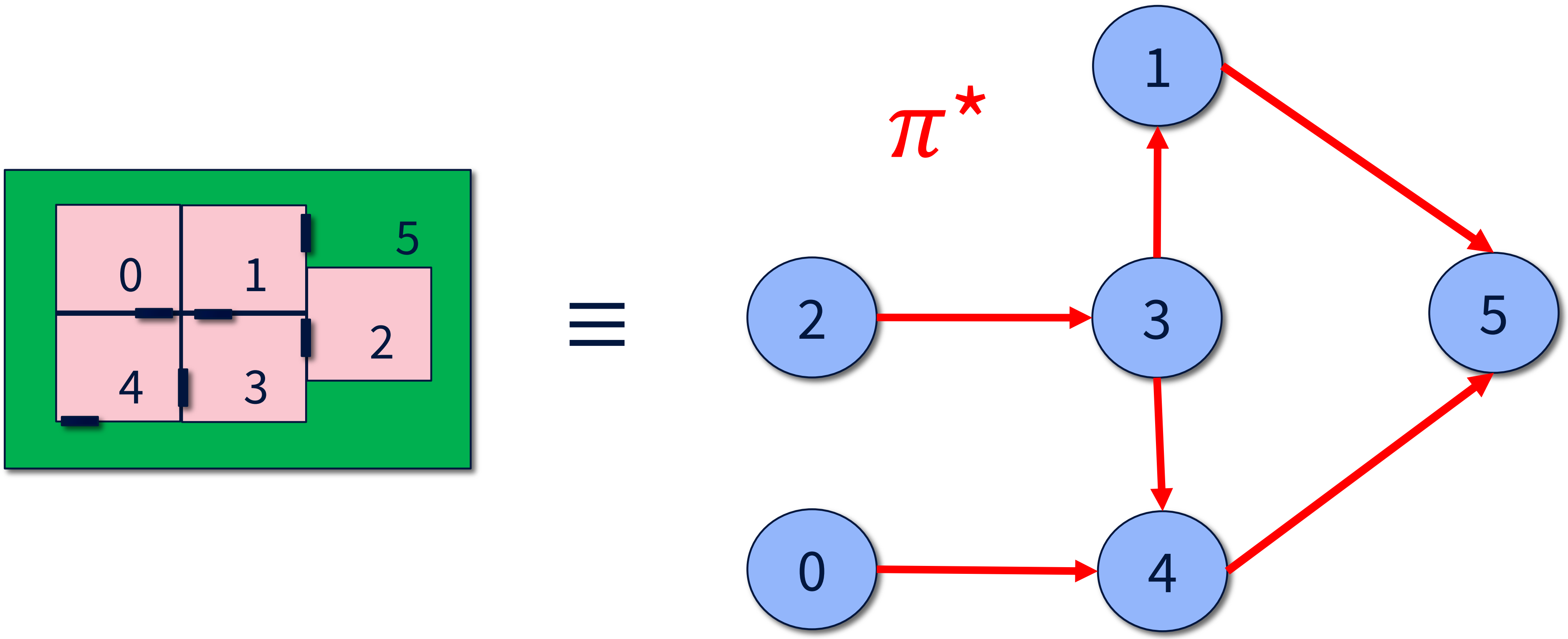
	0	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	0	0	0	0	**100**
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

A Simple Example of Q-Learning

If our agent learns more through further episodes, it will finally reach convergence values in Q-table like:

s \ a	0	1	2	3	4	5		
0	0	0	0	0	80	0	$V^*(0)=80$	$\pi^*(0)$ = go to 4
1	0	0	0	64	0	100	$V^*(1)=100$	$\pi^*(1)$ = go to 5
2	0	0	0	64	0	0	$V^*(2)=64$	$\pi^*(2)$ = go to 3
3	0	80	51	0	80	0	$V^*(3)=80$	$\pi^*(3)$ = go to 4 or 1
4	64	0	0	64	0	100	$V^*(4)=100$	$\pi^*(4)$ = go to 5
5	0	0	0	0	0	0	$V^*(5)=0$	$\pi^*(5)$ = { } (goal)

A Simple Example of Q-Learning



Problems with tabular Q-Learning

- In realistic situations we cannot learn about each state!
  - Too many states to visit them all in training (**slow convergence)**
  - Too many space to hold the Q-tables in memory (demand for **memory resources)**
- Instead, we want to <u>generalize</u>:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states

Q-learning Algorithm: Possible solutions

- **If we know a model:**
  1. We use the known model to build a **simulation**.
  2. Using Q-learning **plus a function approximation technique**, we learn to behave in the simulated environment, which yields a good control policy for the original problem

Hot topics: Deep RL, batch-RL, transfer learning!

Deep Reinforcement Learning