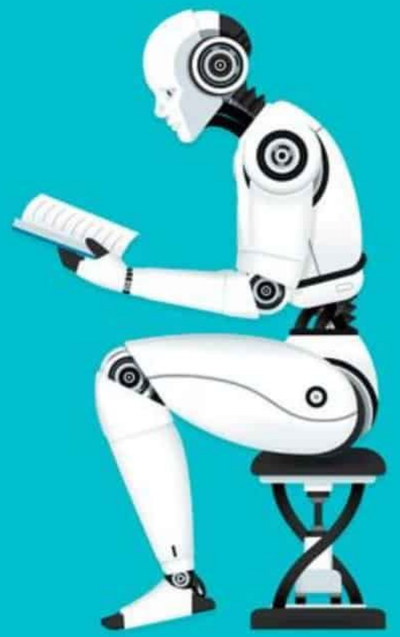
AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

RSE Ricerca Sistema Energetico | we move rsearch

Reinforcement Learning

A gentle introduction



Davide Salaorni - 13 marzo 2024

Who am (A)I?

Since 1996

Davide Salaorni
28 yo - Legnago (VR)

2015 - 2021

B.Sc. Ingegneria Informatica @ Polimi
M.Sc. Computer Science and Engineering @ Polimi

TODAY

Ph.D. student @ POLIMI - Scholarship by ✨RSE✨
Supervised by Marcello Restelli and Francesco Trovò

REINFORCEMENT LEARNING

ARTIFICIAL INTELLIGENCE
REINFORCEMENT LEARNING
APPLICATIONS OF RL
RESOURCES AND

Agenda

1. Artificial Intelligence

- History and definition

2. Reinforcement Learning

- History and achievements
- Mathematical formulation
- Taxonomy
- Comparison with MPC

3. Applications in Energy Field

- Smart Grid
- Water Distribution Systems

4. Do try this at home…

- Python Libraries
- Books and Courses

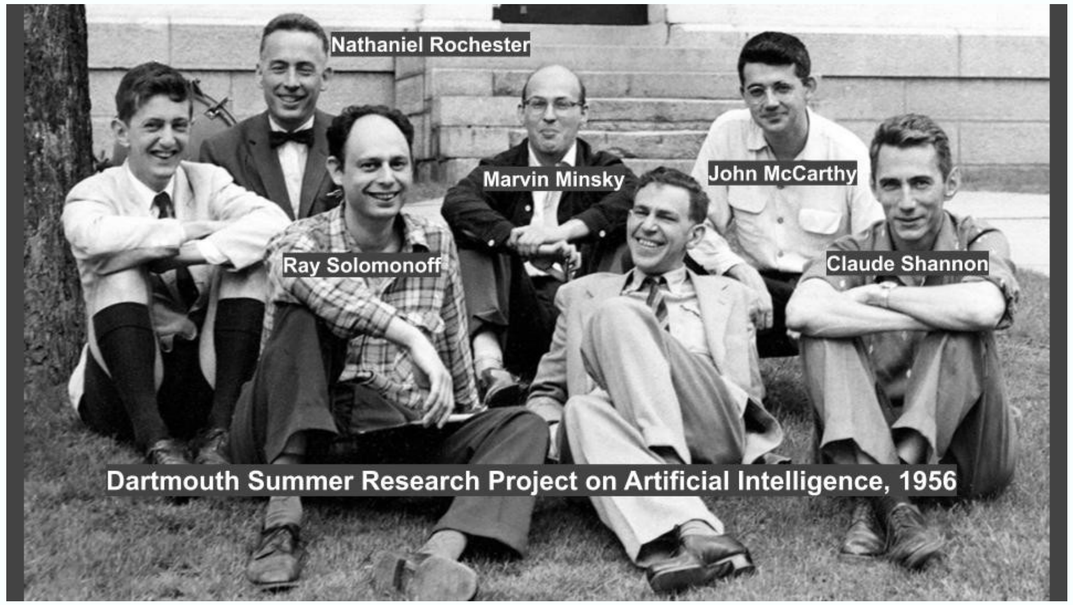
AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

RSE Ricerca Sistema Energetico | we move rsearch

Artificial Intelligence

What is AI?

In 1950 Alan Turing asks “*whether if it is possible for a machinery to show intelligent behaviour*”

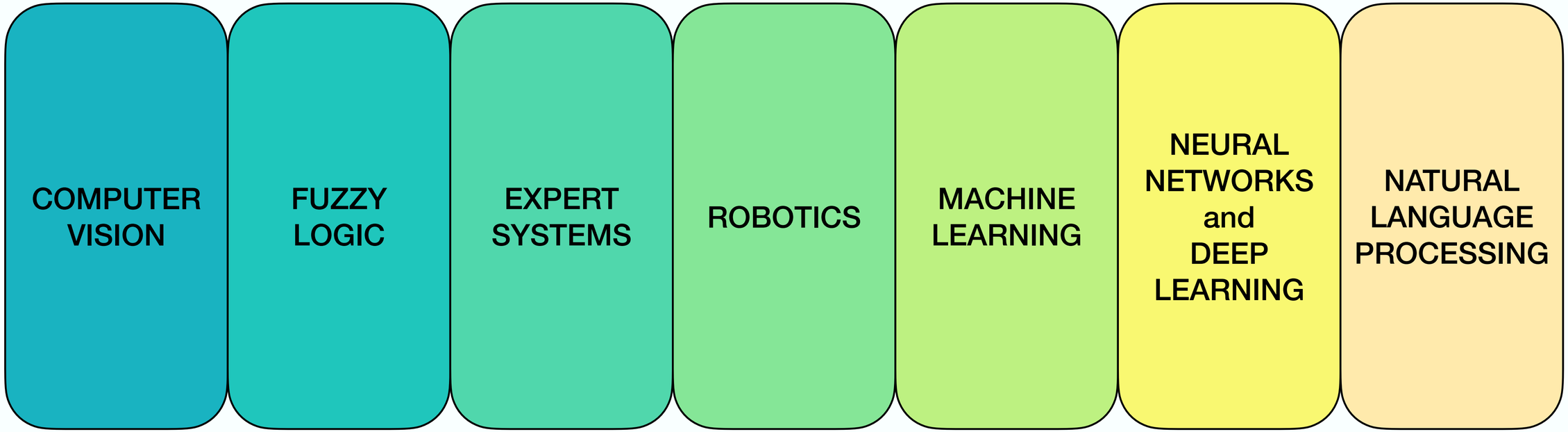


Russell and Norvig approch the definition by distinguishing between 4 “schools of thoughts”

	Human-like behavior	**Rational behavior**
Reasoning	Systems that think humanly	Systems that think rational
Acting	Systems that act humanly	Systems that act rational

“The intelligence is the computational part of the ability to achieve goals in the world” (McCarthy)

What is AI?


COMPUTER VISION
FUZZY LOGIC
EXPERT SYSTEMS
ROBOTICS
MACHINE LEARNING
NEURAL NETWORKS and DEEP LEARNING
NATURAL LANGUAGE PROCESSING

What is AI?

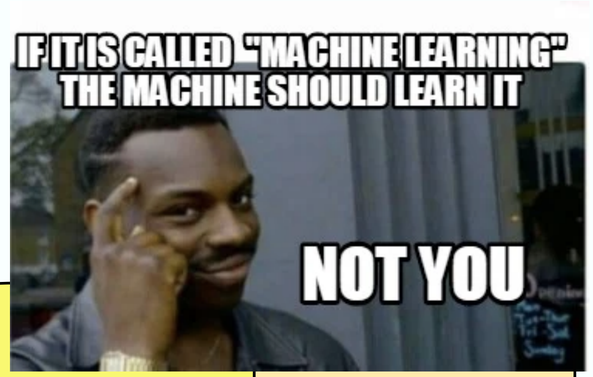
COMPUTER VISION

FUZZY LOGIC

EXPERT SYSTEMS

ROBOTICS

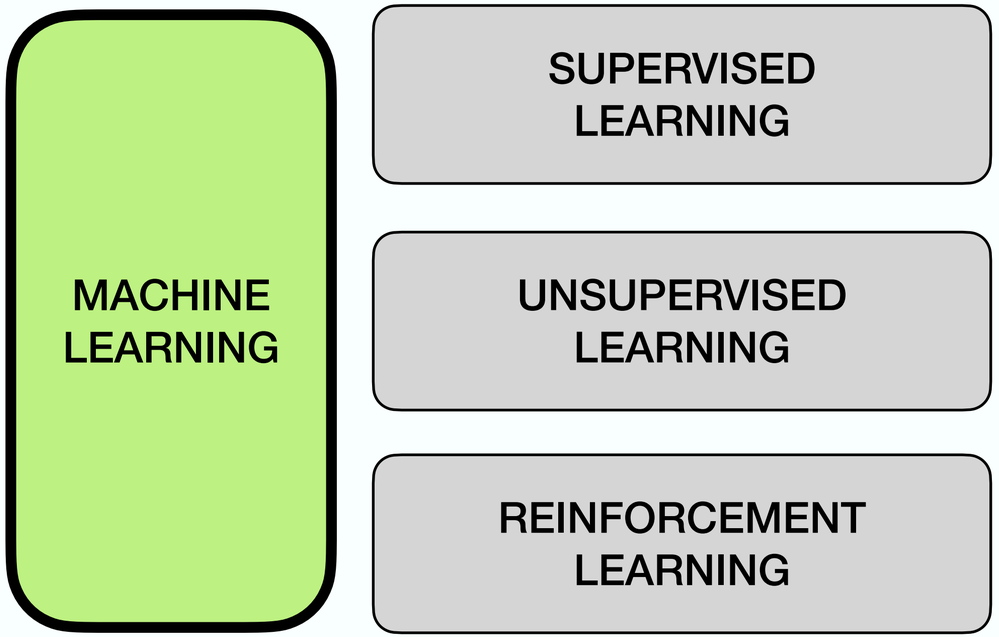
MACHINE LEARNING



NEURAL NETWORKS and DEEP LEARNING

NATURAL LANGUAGE PROCESSING

Machine Learning in a nutshell



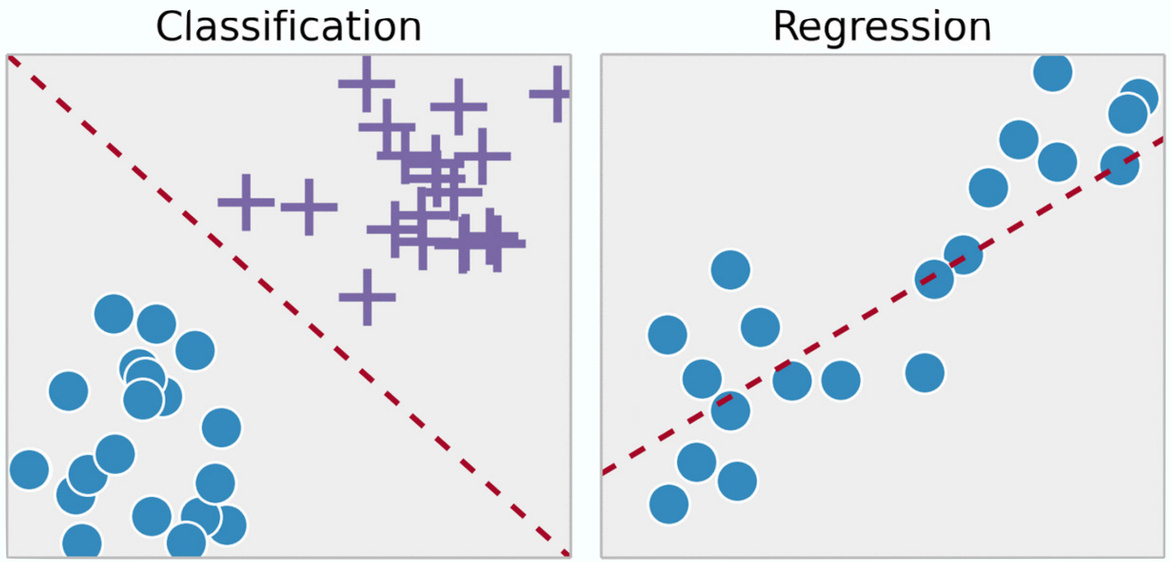
Machine Learning in a nutshell

MACHINE LEARNING

SUPERVISED LEARNING

UNSUPERVISED LEARNING

REINFORCEMENT LEARNING



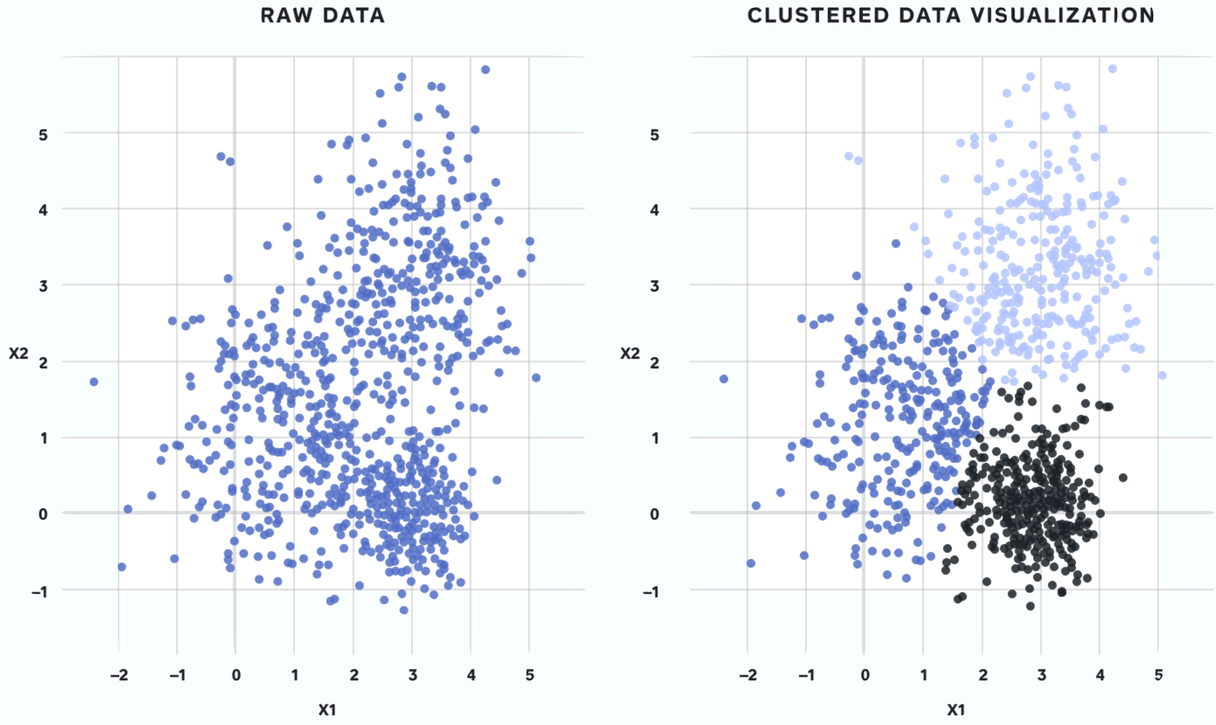
Machine Learning in a nutshell

MACHINE LEARNING

SUPERVISED LEARNING

UNSUPERVISED LEARNING

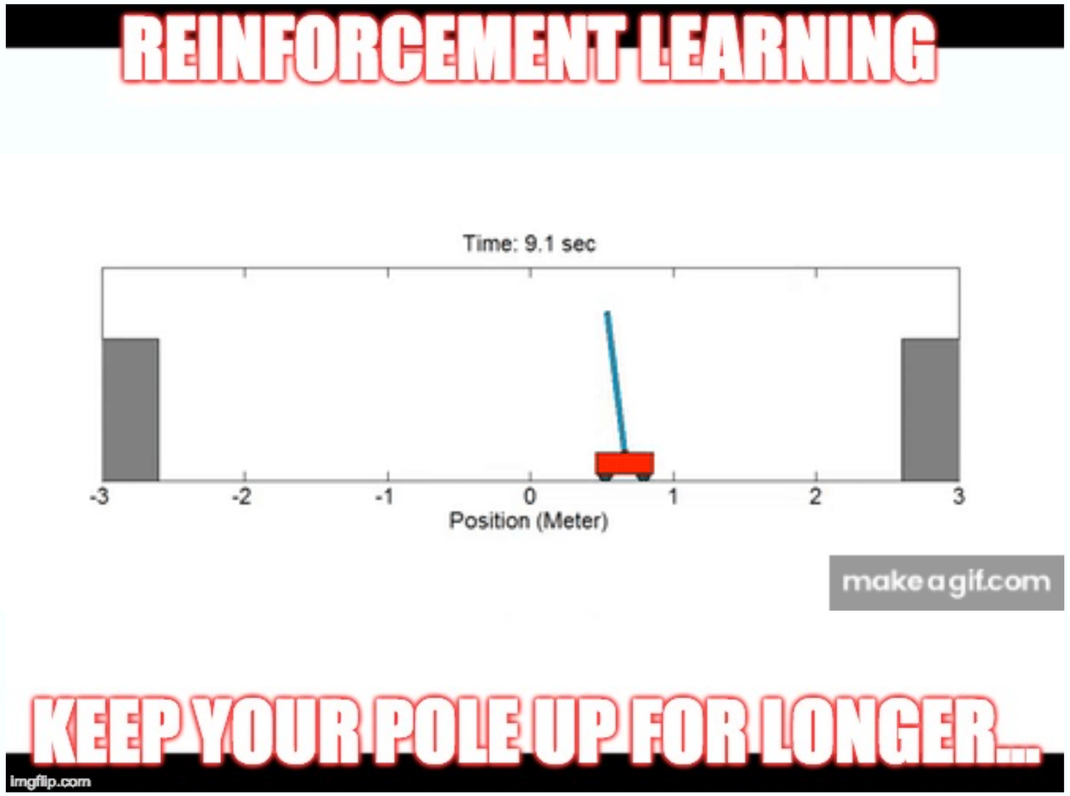
REINFORCEMENT LEARNING



Machine Learning in a nutshell

MACHINE LEARNING

- SUPERVISED LEARNING
- UNSUPERVISED LEARNING
- REINFORCEMENT LEARNING

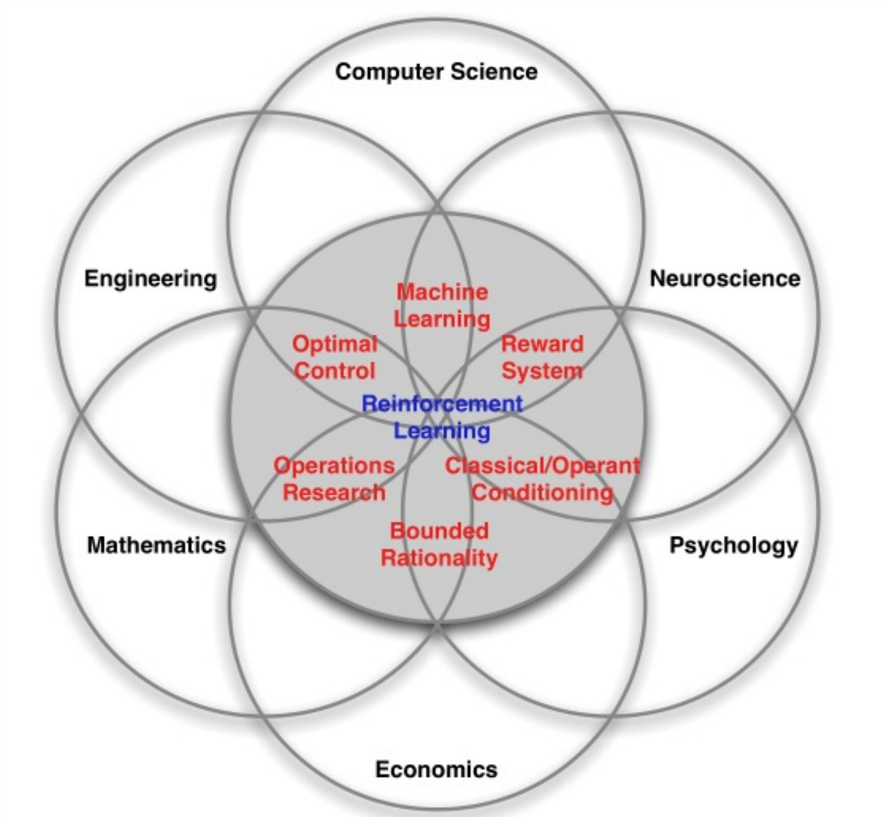


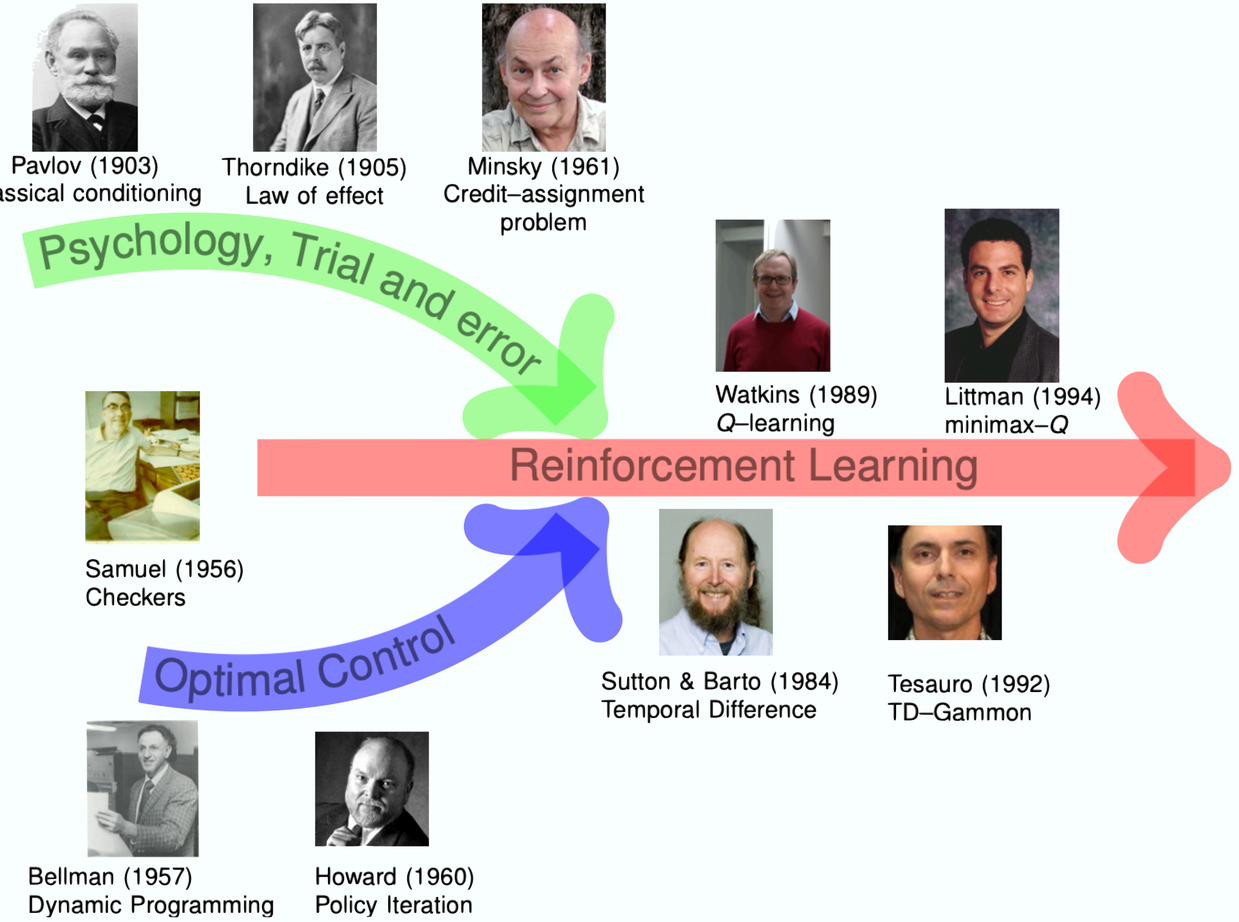
AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

RSE Ricerca Sistema Energetico | we move rsearch

Reinforcement Learning

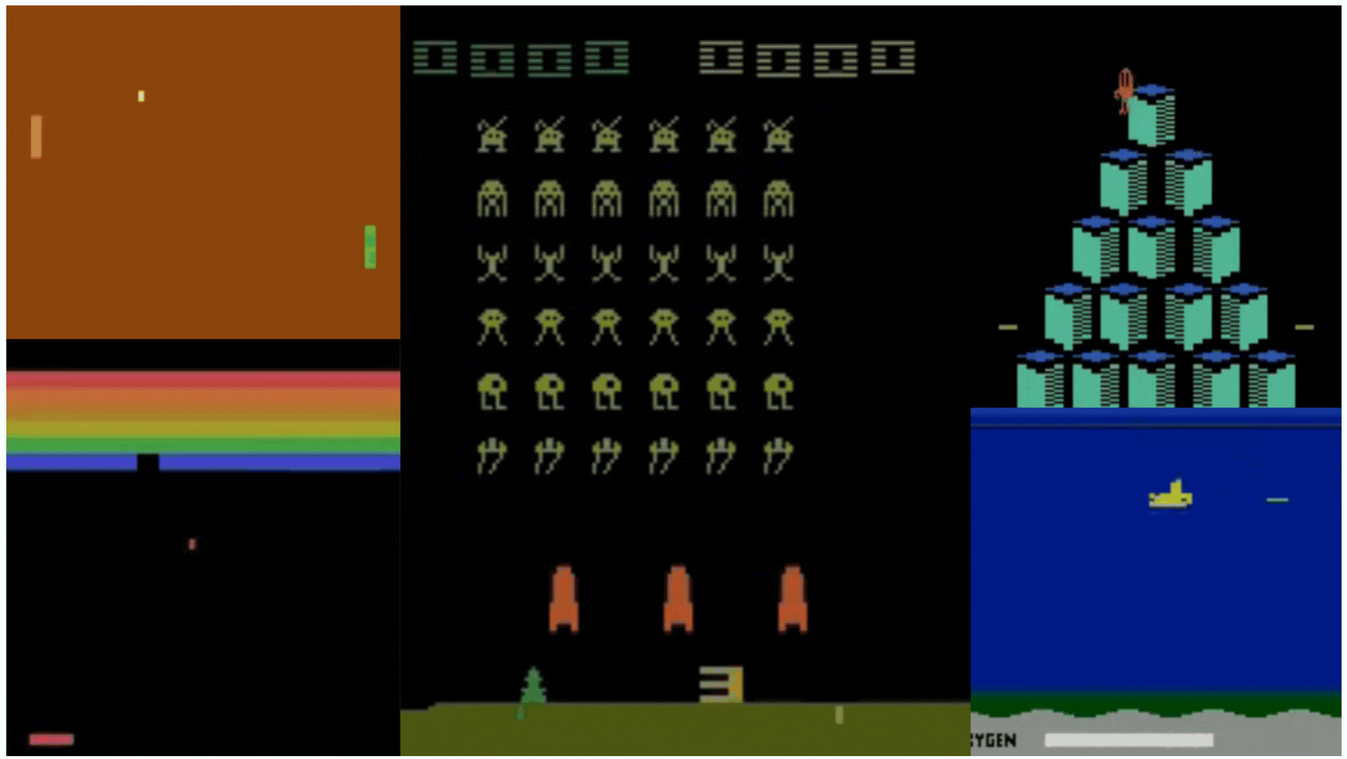
Where RL comes from?

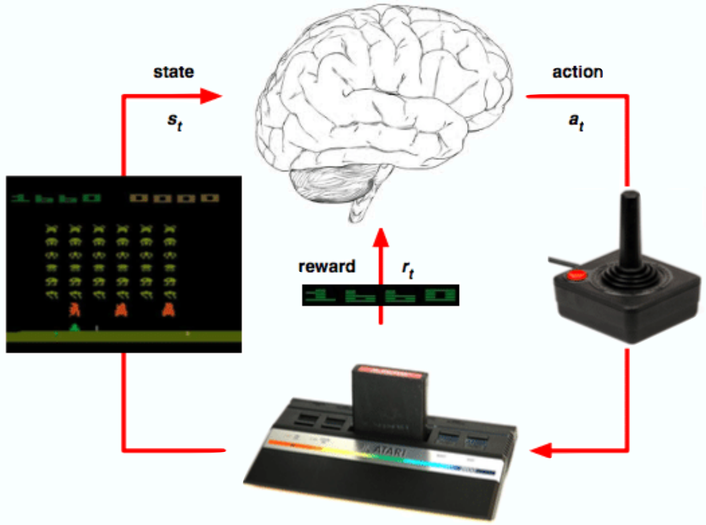




Where RL beat humans?

2013: playing Atari games





Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing Atari with Deep Reinforcement Learning

Where RL beat humans?

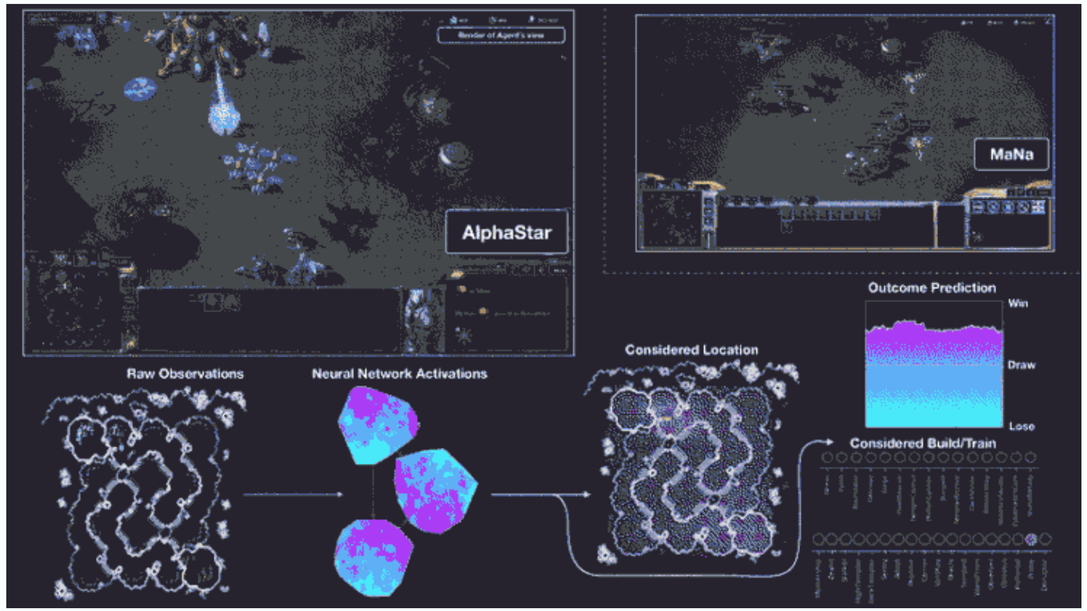
2016: AlphaGo vs Lee Sedol

Final result: 4 - 1

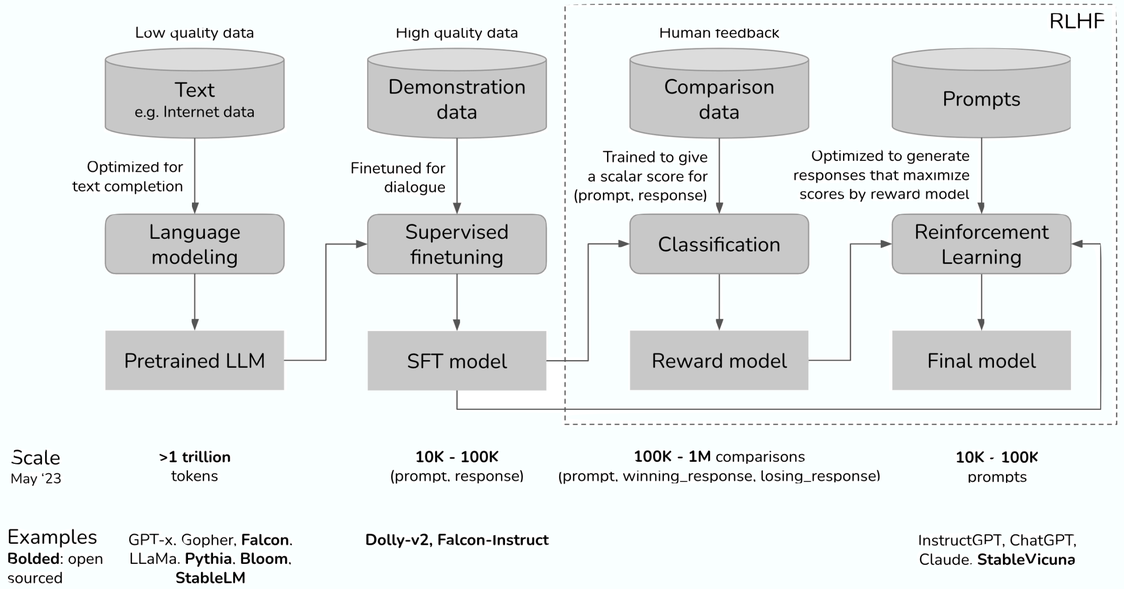
Silver, D., Huang, A., Maddison, C. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016)

Where RL beat humans?

2019: AlphaStar playing StarCraft II

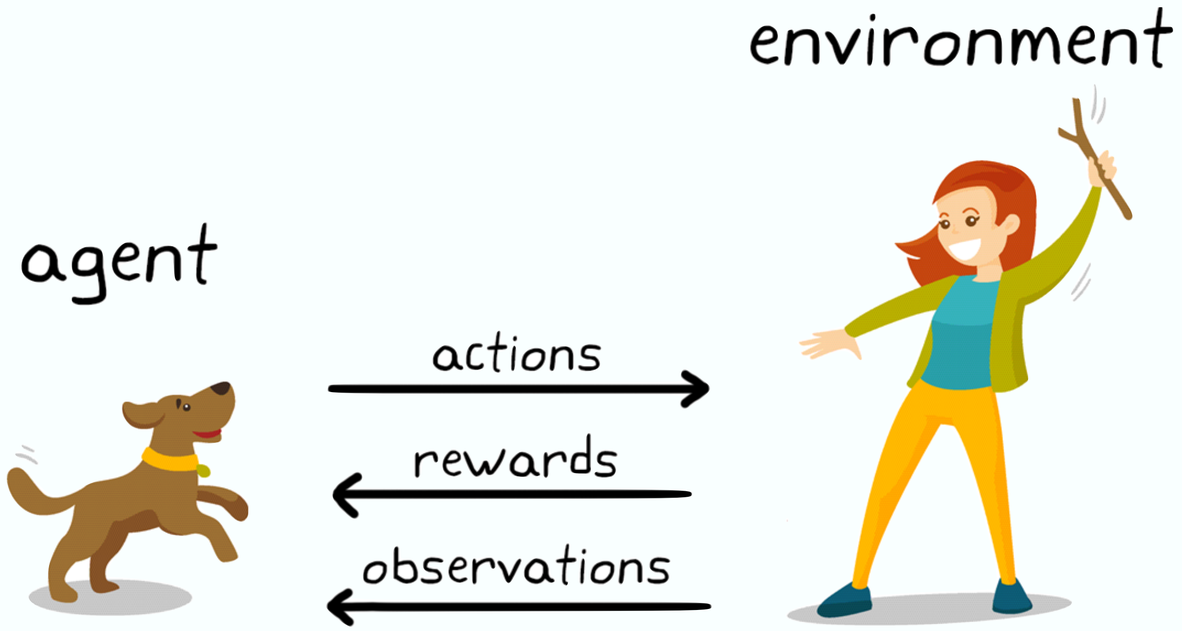


Mathieu M, Ozair S, Srinivasan S, Gulcehre C, Zhang S, Jiang R, Paine TL, Powell R, Żołna K, Schrittwieser J, Choi D, Georgiev P, Toyama D, Huang A, Ring R, Babuschkin I, Ewalds T, Bordbar M, Henderson S, Colmenarejo SG, van den Oord A, Czarnecki WM, de Freitas N, Vinyals O (2023) AlphaStar Unplugged: Large-Scale Offline Reinforcement Learning



2022: RL with Human Feedback in ChatGPT

The power of reinforcements



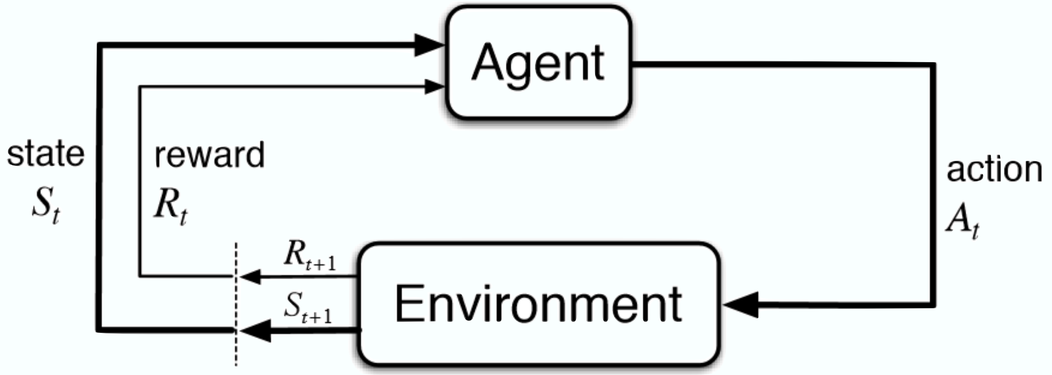
Observations: woman and stick movement

Actions: movements towards the stick and to take it back to master

Reward: dog biscuits

Goal: get a lot of dog biscuits

Sequential Decision Making



The **history** of the process is the sequence of observations, actions and rewards:

$$h_t = \{a_1, o_1, r_1, a_2, o_2, r_2, \ldots, a_t, o_t, r_t\}$$

The **state** is the information used to determine how to behave depending on the past experience:

$$s_t = f(h_t) = f(a_1, o_1, r_1, \ldots, a_t, o_t, r_t)$$

Example 1: pole balancing

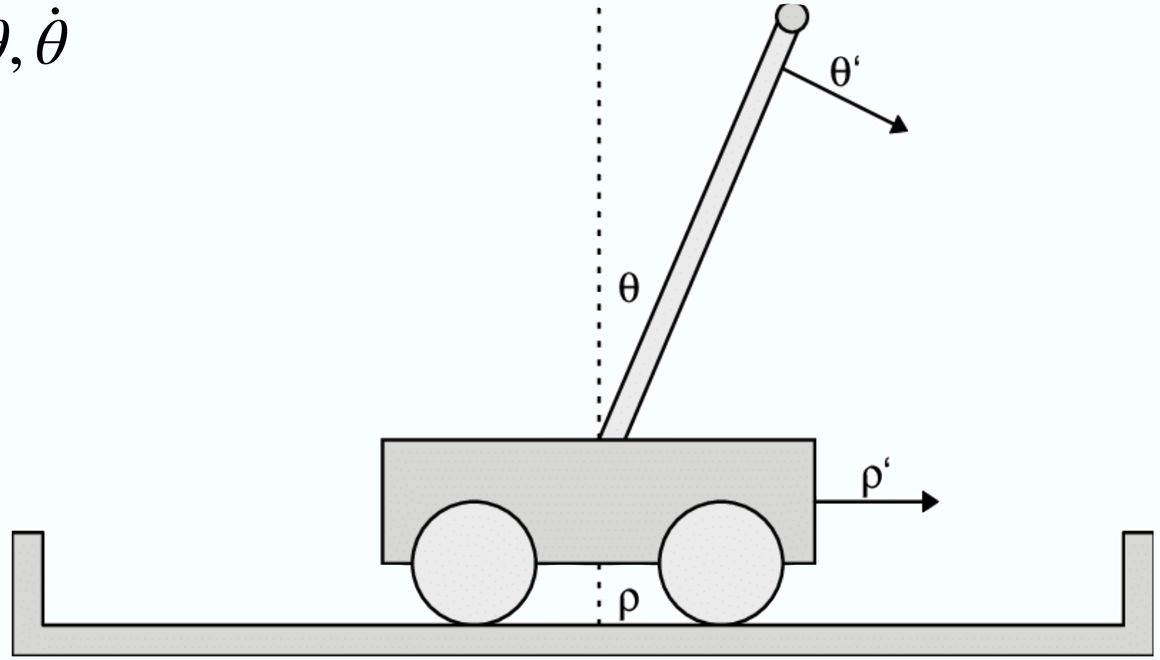
State space: 4 continuous state variables $\rho, \dot{\rho}, \theta, \dot{\theta}$

Actions: 2 actions $\{-N, N\}$

State transitions: deterministic

Reward:

- 0 in the goal region
- -1 outside the goal region
- -100 unfeasible region



Example 2: blackjack

State space: ≈ 800

Actions: from 2 to 4 depending on the state

State transitions: stochastic

Reward: 0 each step, {-2, -1, 0, 1, 1.5, 2}
at the end

Example 3: chess

State space: $\approx 10^{47}$

Actions: from 0 to 218

State transitions: deterministic

Reward: 0 each step, {-1, 0, 1} at the end

Size of the game tree: 10^{123}

How do we model the system?

The system can usually be described as a **stochastic process**, i.e. an indexed collection of **random variables** $\{X_t\}$.

The index represents:

- the instant of time in the **continuous case,** or
- the *t-th* configuration of the environment among a finite number of mutually exclusive and exhaustive labeled states in the **discrete case**

Generally, the probability to reach a particular state s at time t+1 is:

$$p_s = \mathbb{P}(X_{t+1} = s \,|\, h_t) = \mathbb{P}(X_{t+1} = s \,|\, X_t = x_t\,, X_{t-1} = x_{t-1}\,, \dots\,, X_2 = x_2\,, X_1 = x_1)$$

Markov Process

Reinforcement Learning problems are usually modeled as Markovian processes, which respect the **Markov assumption**:

"The future is independent from the past given the present"

This means that each state captures all the information from the history and we can just throw away the past:

$$\mathbb{P}(X_{t+1} = s \,|\, X_t = x_t, X_{t-1} = x_{t-1}, \dots, X_2 = x_2, X_1 = x_1) = \mathbb{P}(X_{t+1} = s \,|\, X_t = x_t)$$

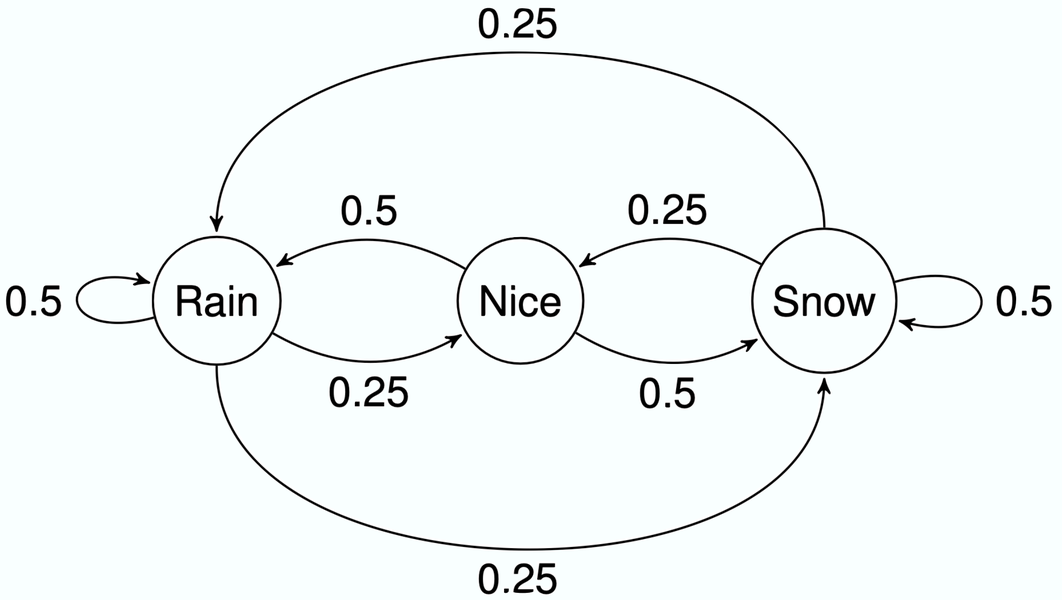
If the process is **stationary** (time invariant), we can write:

$$p_{i \to j} = \mathbb{P}(X_{t+1} = j \,|\, X_t = i) = \mathbb{P}(X_1 = j \,|\, X_0 = i)$$

Markov Process

A Markov Process is a **memoryless** stochastic process described by the tuple $\langle S, P, \mu \rangle$:

- S is a (finite) set of states
- P is a state transition probability matrix, so that $P_{ss'} = \mathbb{P}(s' \mid s)$
- μ is the set of initial probabilities, where $\mu_i^0 = \mathbb{P}(X_0 = i)$



$$P = \begin{array}{c} \\ \textit{Rain} \\ \textit{Nice} \\ \textit{Snow} \end{array} \begin{bmatrix} 0.5 & 0.25 & 0.25 \\ 0.5 & 0 & 0.5 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}$$

Markov Decision Process

A Markov Decision Process (MDP) is a random process described by the tuple $\langle S, A, P, R, \gamma, \mu \rangle$:

- S is a (finite) set of states
- A is a (finite) set of actions
- P is a state transition probability matrix, so that $P_{ss'} = \mathbb{P}(s' \mid s, a)$
- R is a reward function $R(s, a) = \mathbb{E}[r \mid s, a]$
- γ is a discount factor such that $\gamma \in [0, 1]$
- μ is the set of initial probabilities, where $\mu_i^0 = \mathbb{P}(X_0 = i)$

Stay away from easy rewards…

The interaction agent-environment is a sequence of $\{S_0, A_0, R_0, S_1, A_1, \dots\}$, but the objective is not the mere reward.

The agent's goal is the **maximization of the expected return** G_t defined as

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

where T is the **finite time-horizon** and the task is an **episodic task**, meaning that the sequence of agent-environment interactions always reaches a *terminal state*.

A different objective has to be considered with **infinite time-horizon**, because the summation of step rewards may diverge.

That's what γ is made for…

…and prefer discounted ones!

With infinite time-horizon (*continuing tasks*), where $T = \infty$, the return has to be expressed as the summation of the expected discounted rewards

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

With the discount factor $\gamma \in [0, 1)$ we can reproduce

- a **“myopic”** strategy with $\gamma = 0$
- a **“far-sighted”** strategy with $\gamma \approx 1$

Financial and behavioral meaning: *a reward today may be more interesting than a delayed one.*

The policy

A policy π is a distribution over actions given the state:

$$\pi(a \mid s) = \mathbb{P}[a \mid s]$$

The policy defines the **behavior** (or strategy) of the agent and can be:

- markovian or history-dependent
- deterministic or stochastic
- stationary or non-stationary

MDP => **markovian** and **stationary** policy

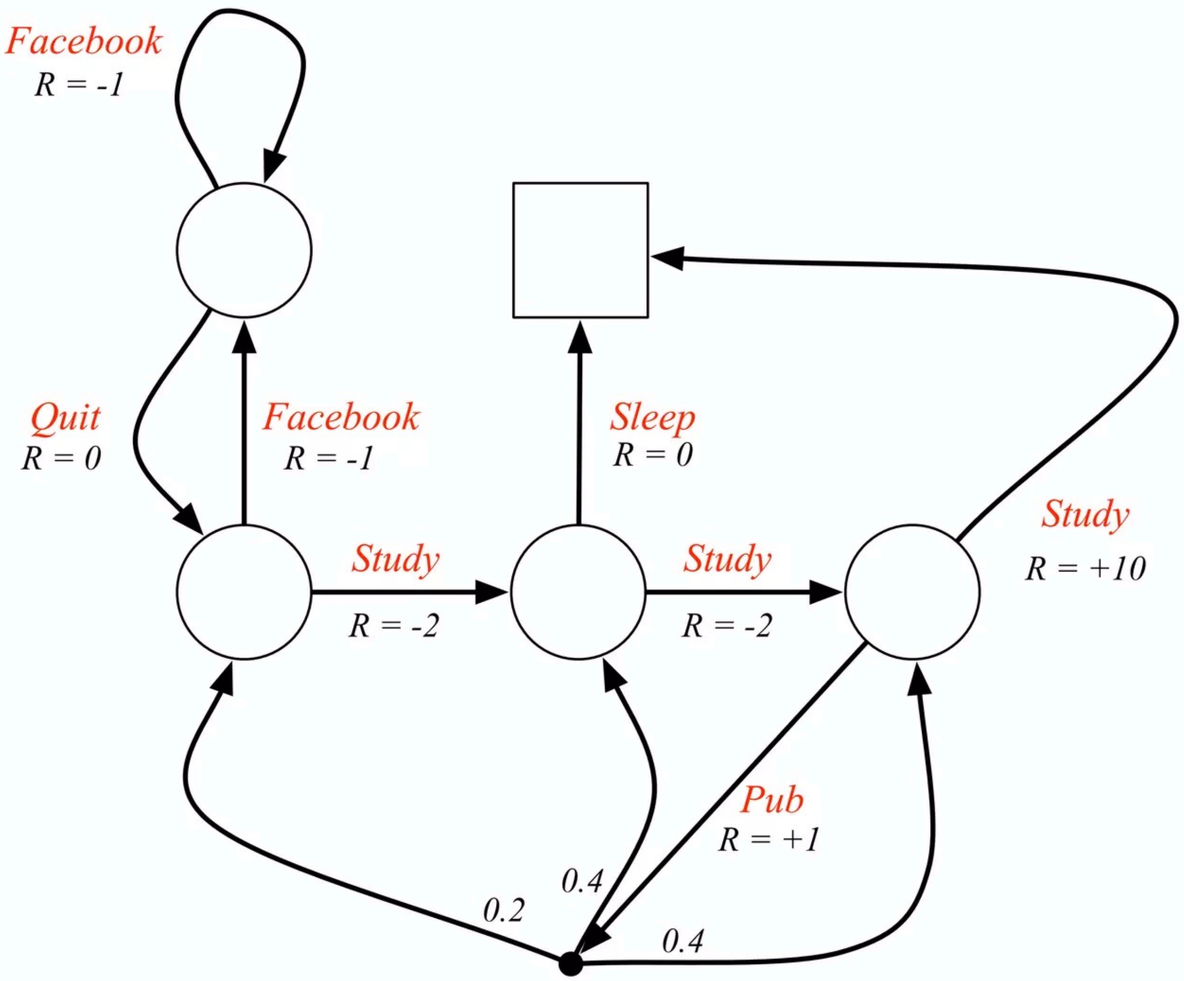
In RL problems, finding the (best) policy is the end goal of our efforts.

Student MDP

Initial state

Non-terminal state

Terminal state



Policy Value Functions

The **state-value function** $V^{\pi}(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t \,|\, s_t = s]$$

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

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t \,|\, s_t = s, a_t = a]$$

The value functions come from dynamic programming and they satisfy a recursive relationship between the value of the state and the values of its successor states.
They are also known as **Bellman equations**.

Bellman Expectation Equations

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}[G_t \,|\, s_t = s] \\
&= \mathbb{E}_{\pi}[R_{t+1} + \gamma\, V^{\pi}(s_{t+1}) \,|\, s_t = s] \\
&= \sum_{a \in A} \pi(a \,|\, s) \left(R(s,a) + \gamma \sum_{s' \in S} P(s' \,|\, s, a)\, V^{\pi}(s') \right)
\end{aligned}
$$

$$
\begin{aligned}
Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[G_t \,|\, s_t = s, a_t = a] \\
&= \mathbb{E}_{\pi}[R_{t+1} + \gamma\, Q^{\pi}(s_{t+1}, a_{t+1}) \,|\, s_t = s, a_t = a] \\
&= R(s,a) + \gamma \sum_{s' \in S} P(s' \,|\, s, a)\, V^{\pi}(s') \\
&= R(s,a) + \gamma \sum_{s' \in S} P(s' \,|\, s, a) \sum_{a' \in A} \pi(a' \,|\, s')\, Q^{\pi}(s', a')
\end{aligned}
$$

Both used to evaluate the goodness (or **utility**) of a policy.

The $Q^{\pi}(s,a)$ is used more for **control purposes** to evaluate the played action.

Variation of Bellman equations take into consideration problem of finding *optimal* policies.

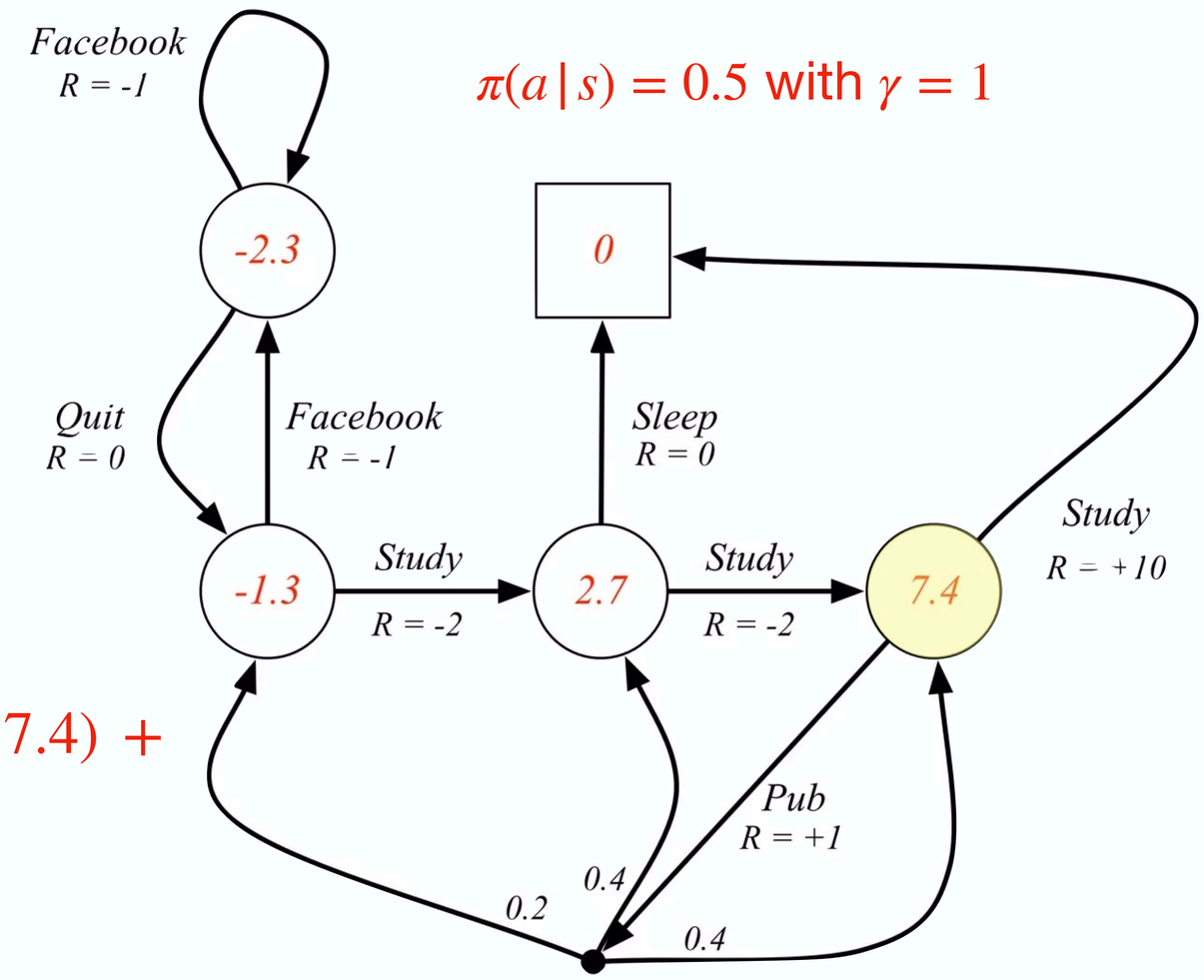
Student MDP
Policy Evaluation

Keeping in mind the Bellman equation

$$V^{\pi}(s) = \sum_{a \in A} \pi(a \mid s) \left(R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a)\, V^{\pi}(s') \right)$$

We can compute the state-value function

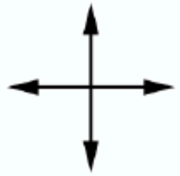
$$\begin{aligned} 7.4 &= 0.5 \cdot \text{"Pub"} + 0.5 \cdot \text{"Study"} \\ &= 0.5 \cdot (1 + 0.2 \cdot (-1.3) + 0.4 \cdot 2.7 + 0.4 \cdot 7.4) \; + \\ &\quad + 0.5 \cdot 10 \end{aligned}$$



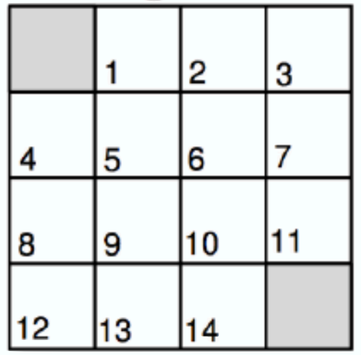
Gridworld: Iterative Policy Evaluation

To avoid the tedious computation of a system of $|S|$ equations in $|S|$ unknowns with the recursive approach, we can go for the **iterative** way.

$$V_0 \rightarrow V_1 \rightarrow \ldots \rightarrow V_k \rightarrow V_{k+1} \rightarrow \ldots V^{\pi}$$



actions

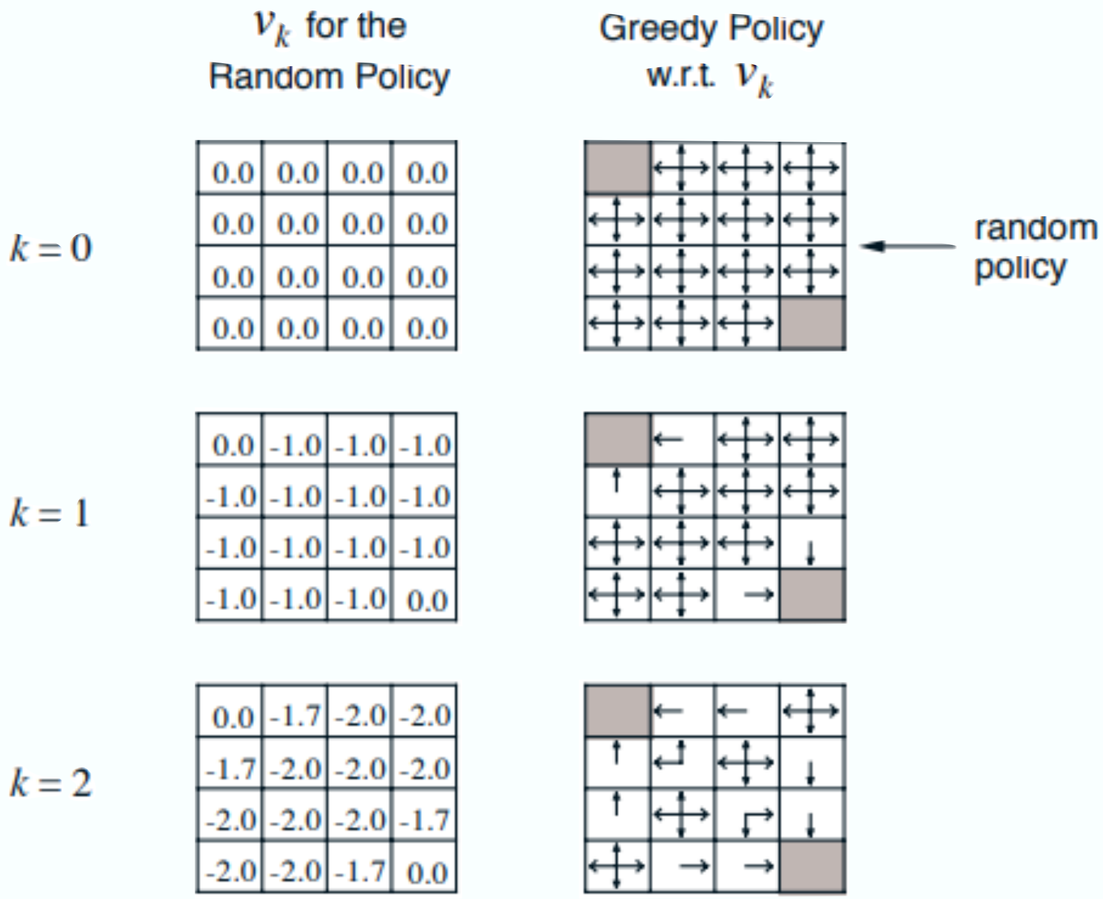


$r = -1$
on all transitions

At each iteration $k+1$, for each state s, we update the state-value function with the state-value function compute in s' at the previous step $V_k(s')$ with the following **backup**:

$$V_{k+1}(s) \leftarrow \sum_{a \in A} \pi(a \mid s) \left(R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s, a)\, V_k(s') \right)$$

Gridworld: Iterative Policy Evaluation



Gridworld: Iterative Policy Evaluation

$k = 3$

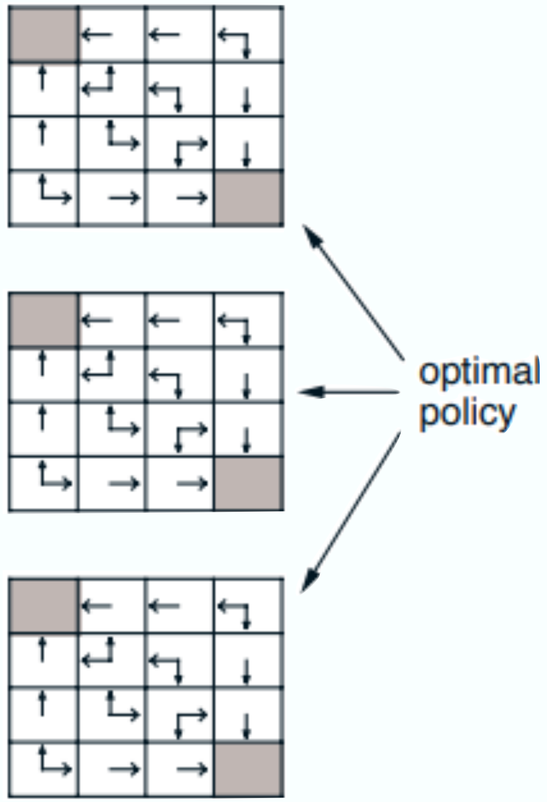
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



optimal policy

Optimal Policy

Value functions define a **partial ordering** over policies: $\pi \geq \pi'$ if $V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in S$

For any MDP:

- there exists an **optimal policy** π^* that is better than or equal to all other policies $\pi^* > \pi, \forall \pi$
- all optimal policies achieve the **optimal value function** $V^{\pi^*}(s) = V^*(s)$
- all optimal policies achieve the **optimal action-value function** $Q^{\pi^*}(s, a) = Q^*(s, a)$
- there is always a **deterministic optimal policy**, that corresponds to

$$\pi^*(a \mid s) = \begin{cases} 1 \text{ if } a = \text{argmax}_{a \in A} Q^*(s, a) \\ 0 \text{ otherwise} \end{cases}$$

Bellman Optimality Equations

$$V^*(s) = \max_a Q^*(s,a)$$
$$= \max_a \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a)\, V^*(s') \right\}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a)\, V^*(s')$$
$$= R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

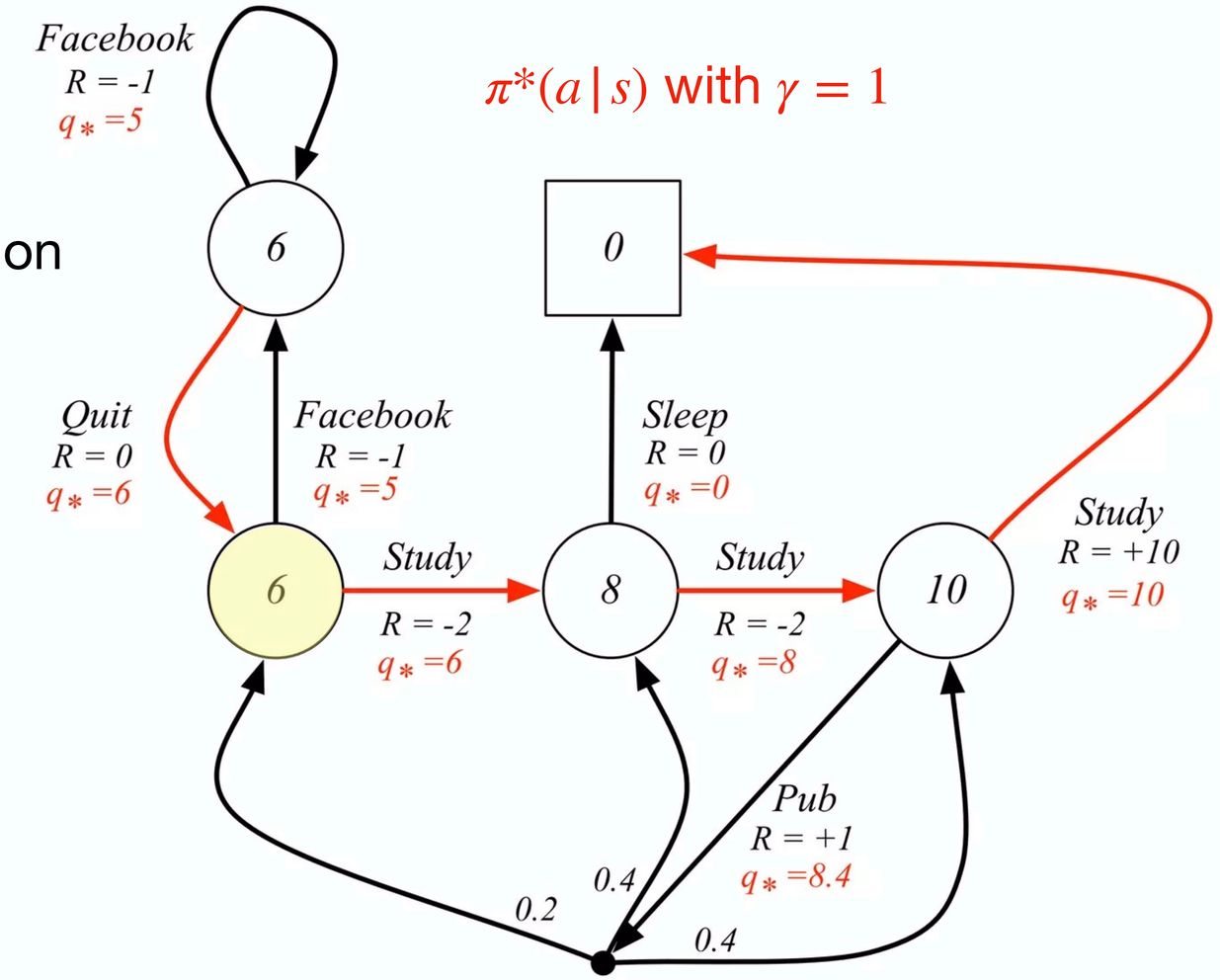
Student MDP
Optimal Policy

Keeping in mind the Bellman optimality equation for the action-value function

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

We can compute the optimal action-value function

$$6 = \max\{\text{"Study"}, \text{"Facebook"}\}$$
$$= \max\{-2+8, -1+6\}$$



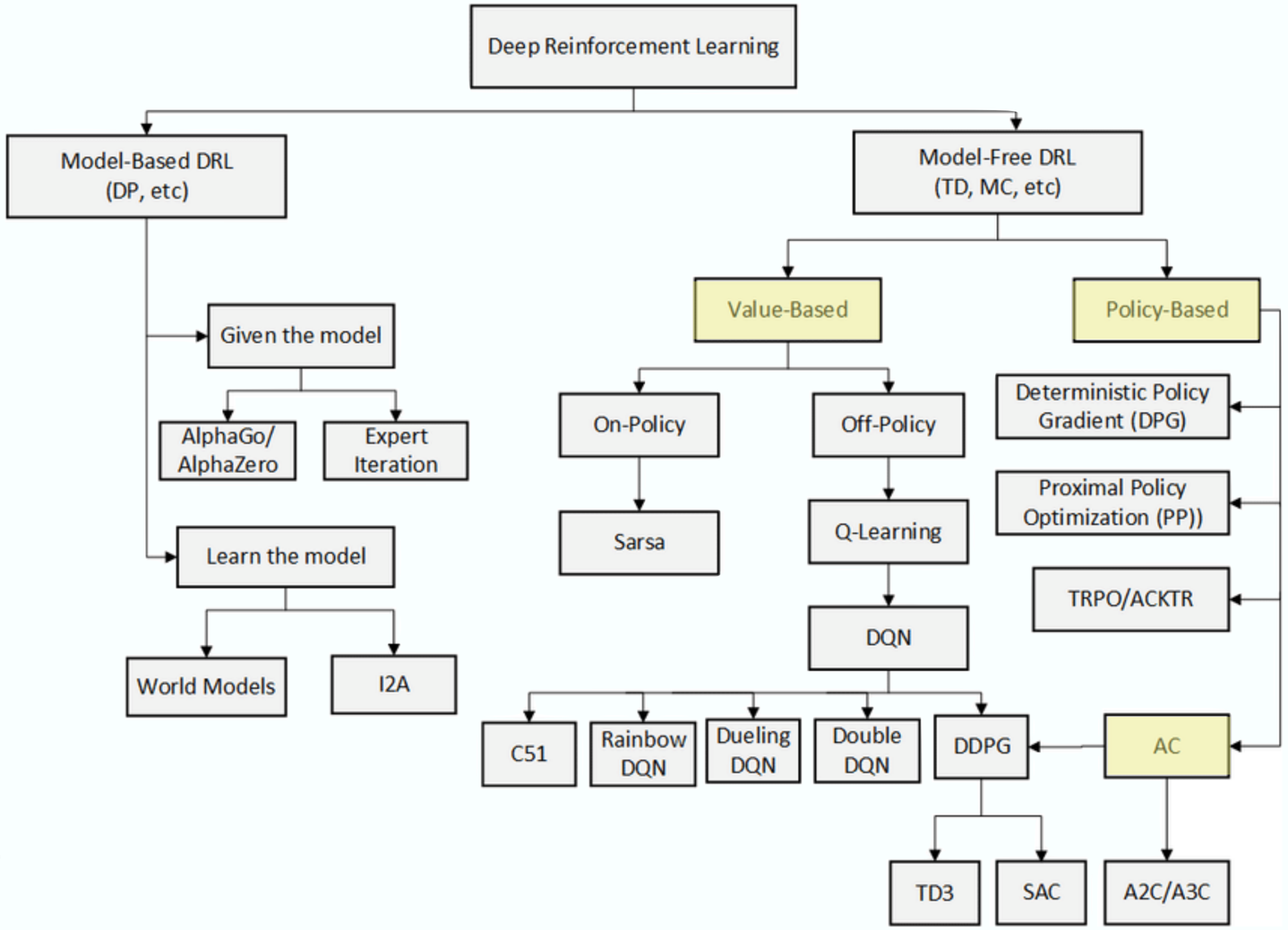
RL Taxonomy

To find the policy we typically have 3 kind of algorithms:

Value based
Learn the state/action-value function of the policy (implicit policy)

Policy based
Learn directly the policy with a parametrized function

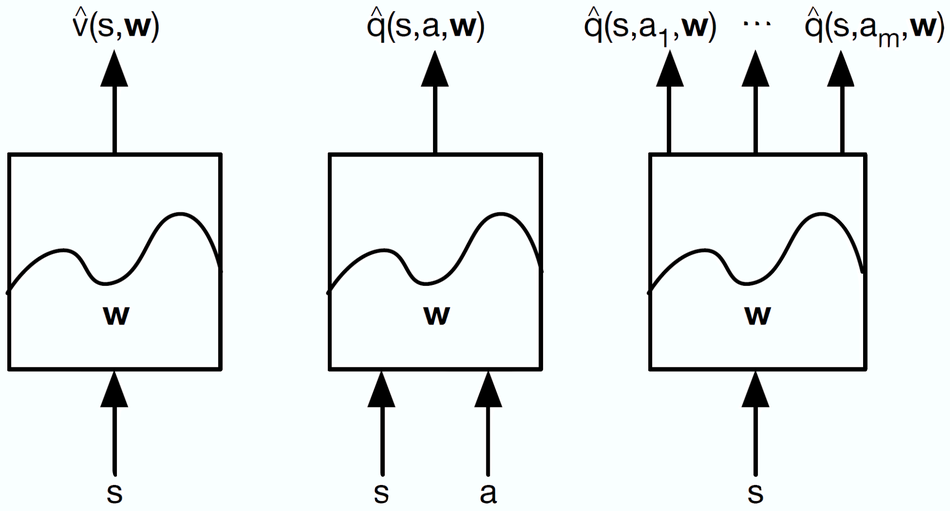
Actor Critic
Hybrid approach that learns both the value function and the policy



To infinity… and beyond!

Large/continuous **state space** S
Large/continuous **action space** A
Continuous Time

Tabular representation that we have used until now cannot be employed



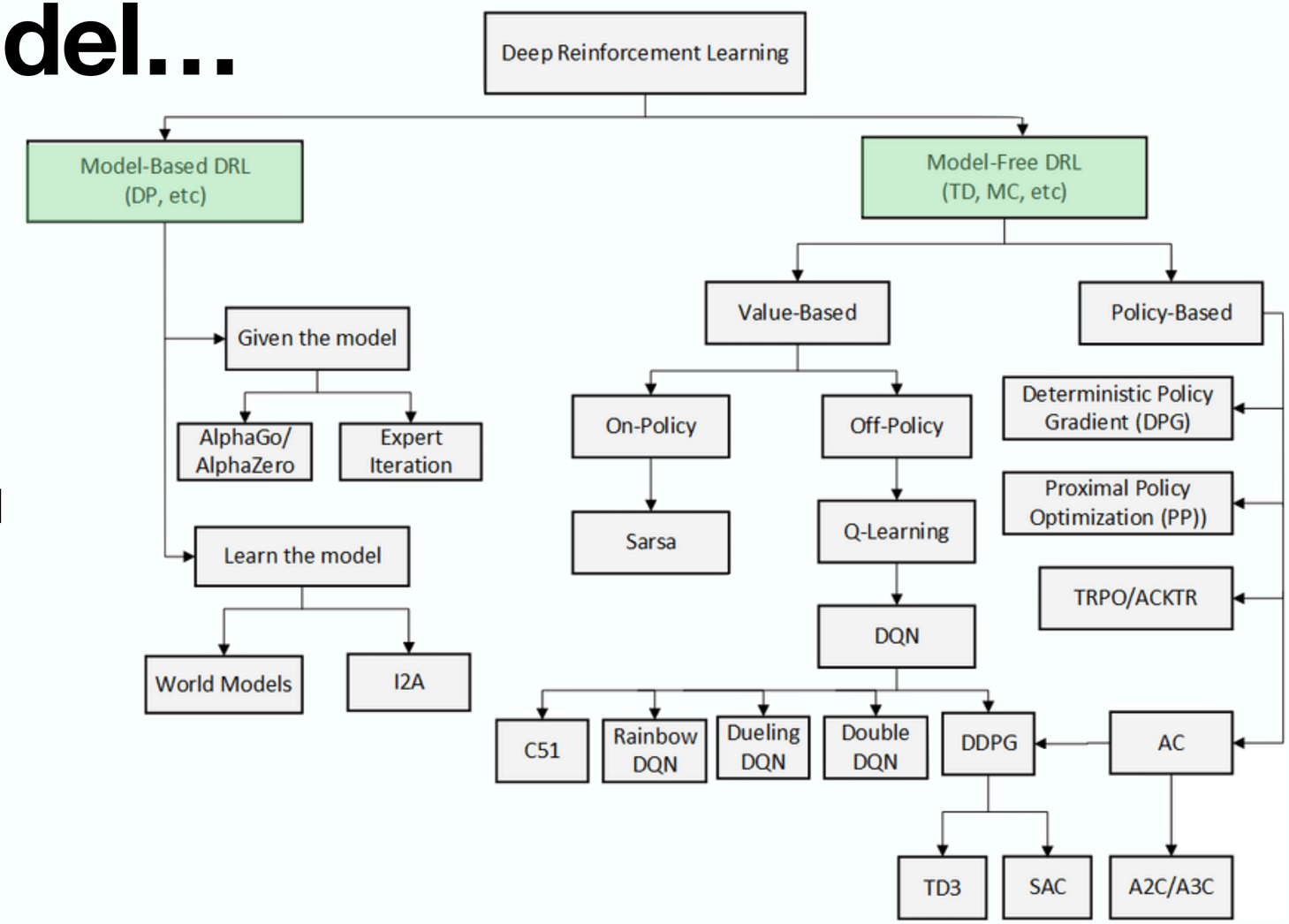
We can estimate the value function with **function approximations** $v_w(s)$ and $q_w(s,a)$ through:

- linear models
- (deep) neural network
- decision trees
- nearest neighbors
- coarse coding, tile coding
- …

Model or not model…

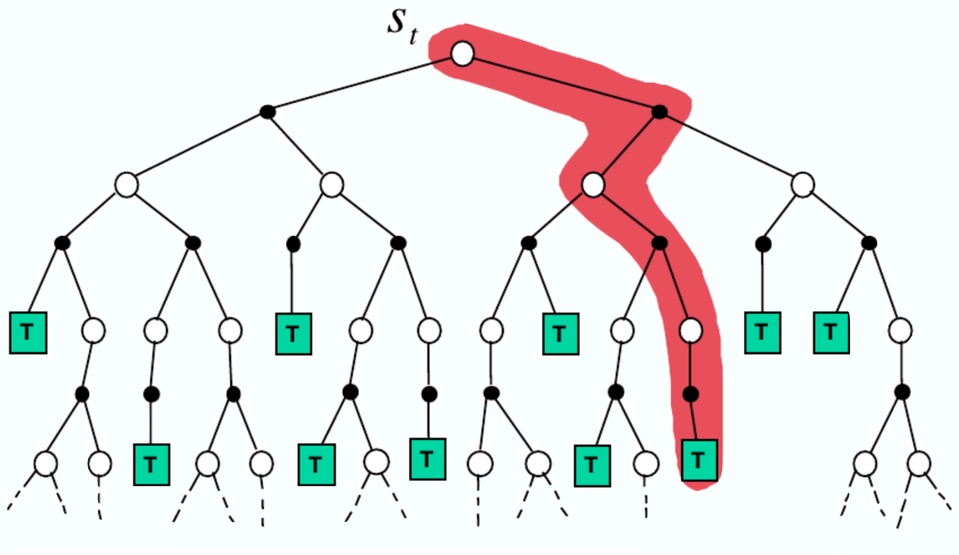
In RL with **model** we mean the model of the environment, i.e. anything that an agent can use to predict how the environment will respond to its actions.

Looking back to the MDP the model is the **transition function** $P(s' \mid s, a)$.

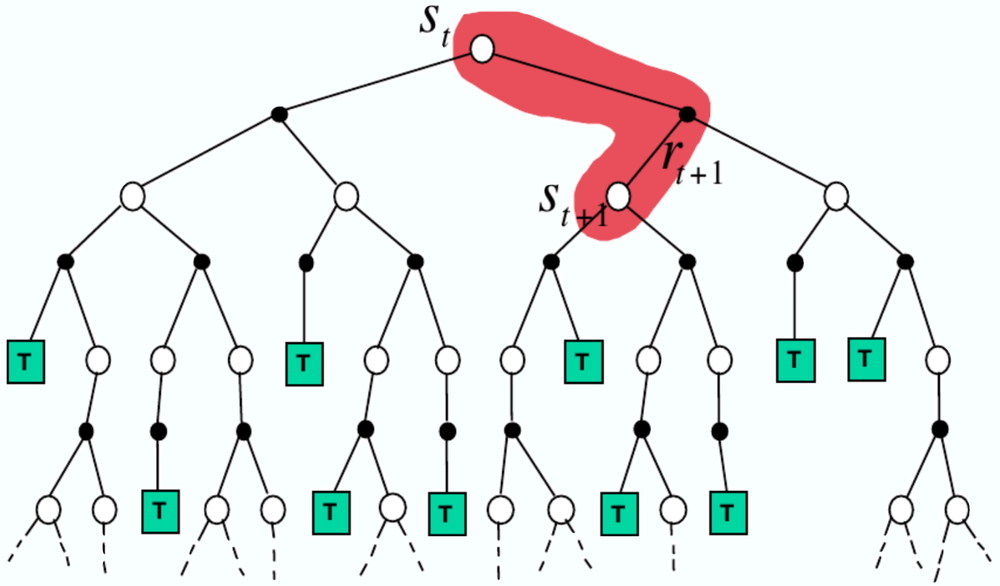


Model-free RL

We don’t know the model of the system (environment’s dynamics) and we learn the value function from *actual experience*.



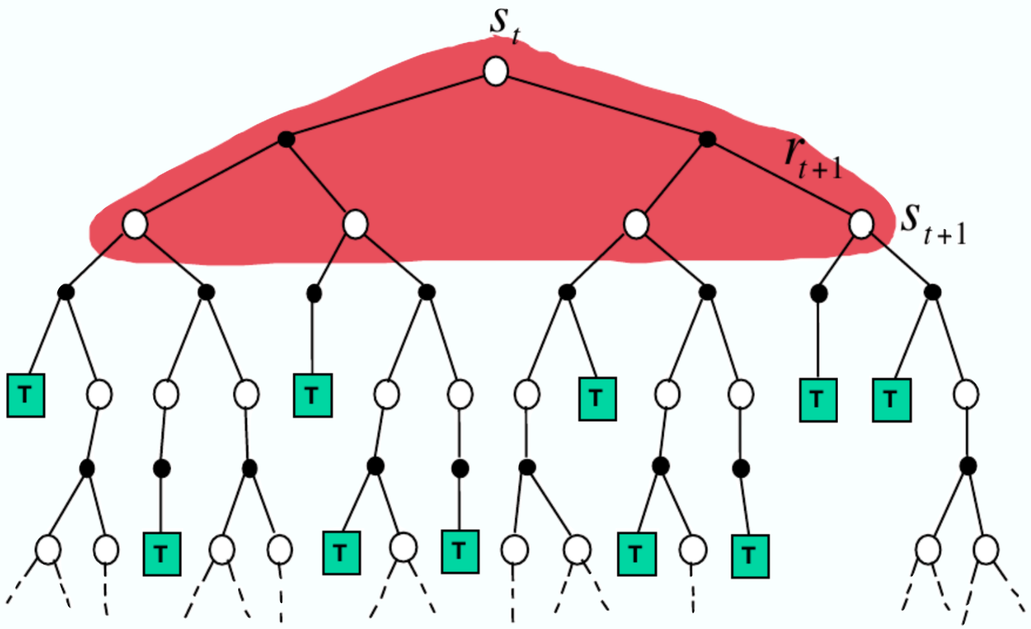
Monte-Carlo Prediction



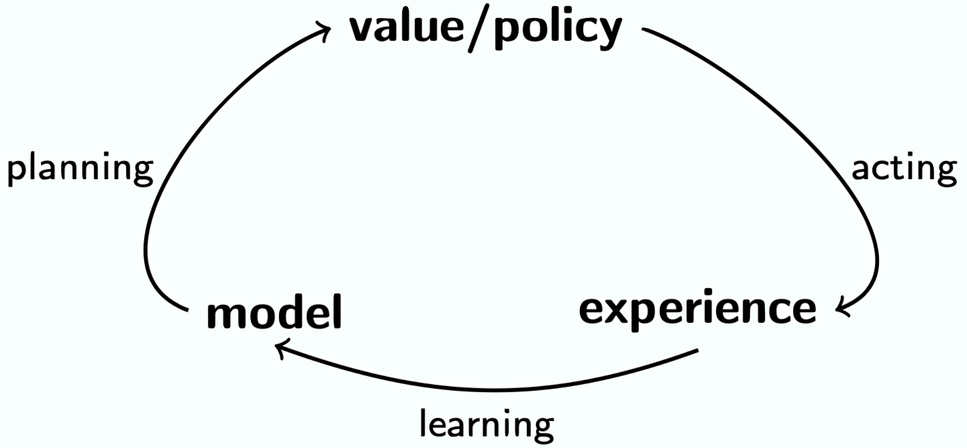
Temporal Difference Prediction

Model-based RL

We have **knowledge** of the environment's dynamics, so we can compute the expected return by considering all the possible actions in every state.



Dynamic Programming



Or we can **learn a model from experience** and then plan the value function v and/or the policy π.

Model Predictive Control

Model Predictive Control is a framework that use a **model** of the process to predict its future evolution and choose the “best” control action.

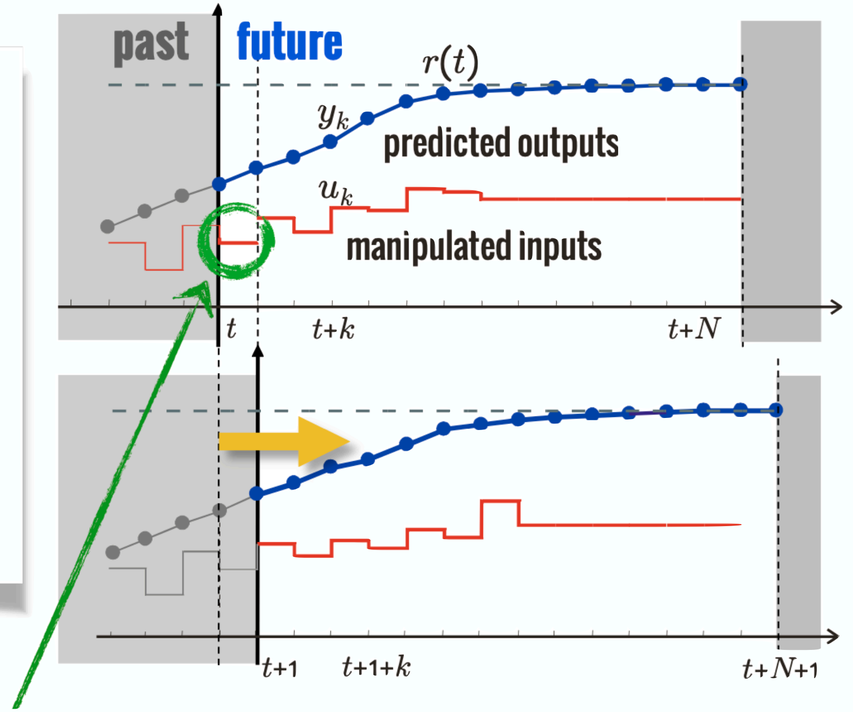
$$\min \sum_{k=0}^{N-1} \|W^y(y_k - r(t))\|_2^2 + \|W^u(u_k - u_r(t))\|_2^2$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k) \qquad \text{prediction model}$$
$$y_k = g(x_k)$$

$$u_{\min} \le u_k \le u_{\max} \qquad \text{constraints}$$
$$y_{\min} \le y_k \le y_{\max}$$

$$x_0 = x(t) \qquad \text{state feedback}$$

→ numerical optimization problem



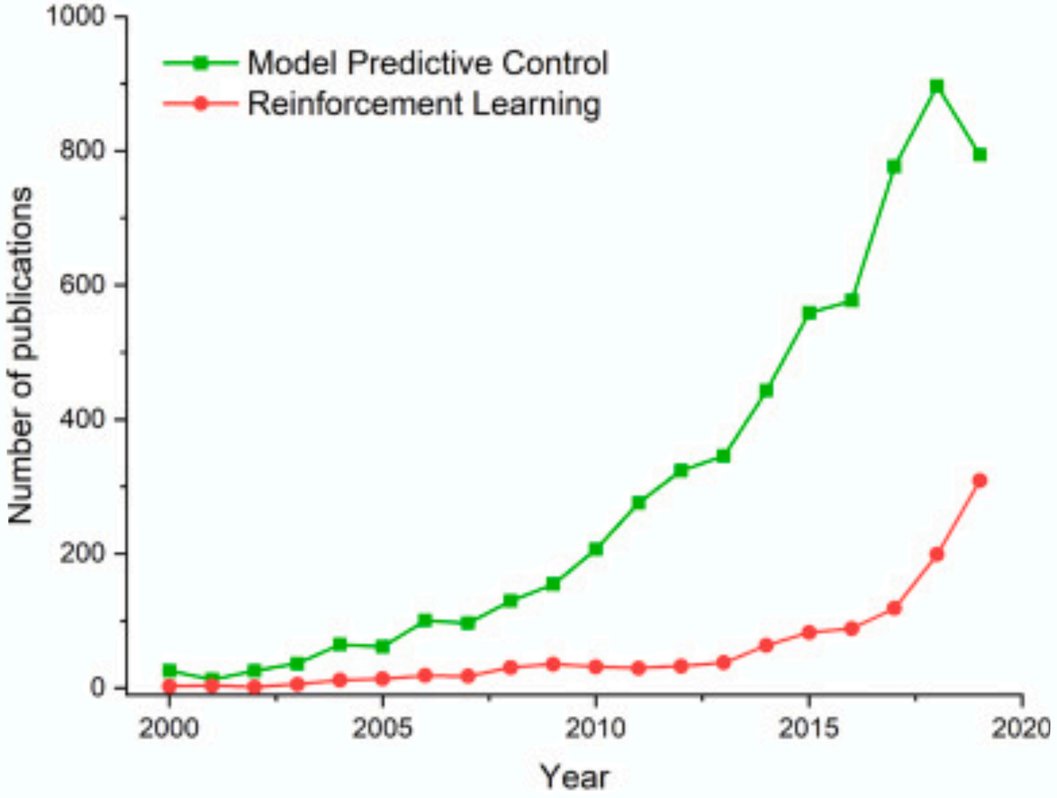
RL vs MPC

The increasing **complexity** of energy systems, **uncertaintes**, and **security** problems, are difficult to control entirely using model-based approaches.

MPC needs the **knowledge of the model** and, for specific families of models, provides **strong theoretical guarantees**.

RL allows to be **more flexible** in the problem formulation and relies on a **learnt approximate model** or entirely on **experience**.

Comparison between MPC and RL approaches in energy sector



A.T.D. Perera, Parameswaran Kamalaruban, Applications of reinforcement learning in energy systems, *Renewable and Sustainable Energy Reviews,* Volume 137 (2021)

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

RSE Ricerca Sistema Energetico | we move rsearch

Applications in Energy Field

Application in Energy Field

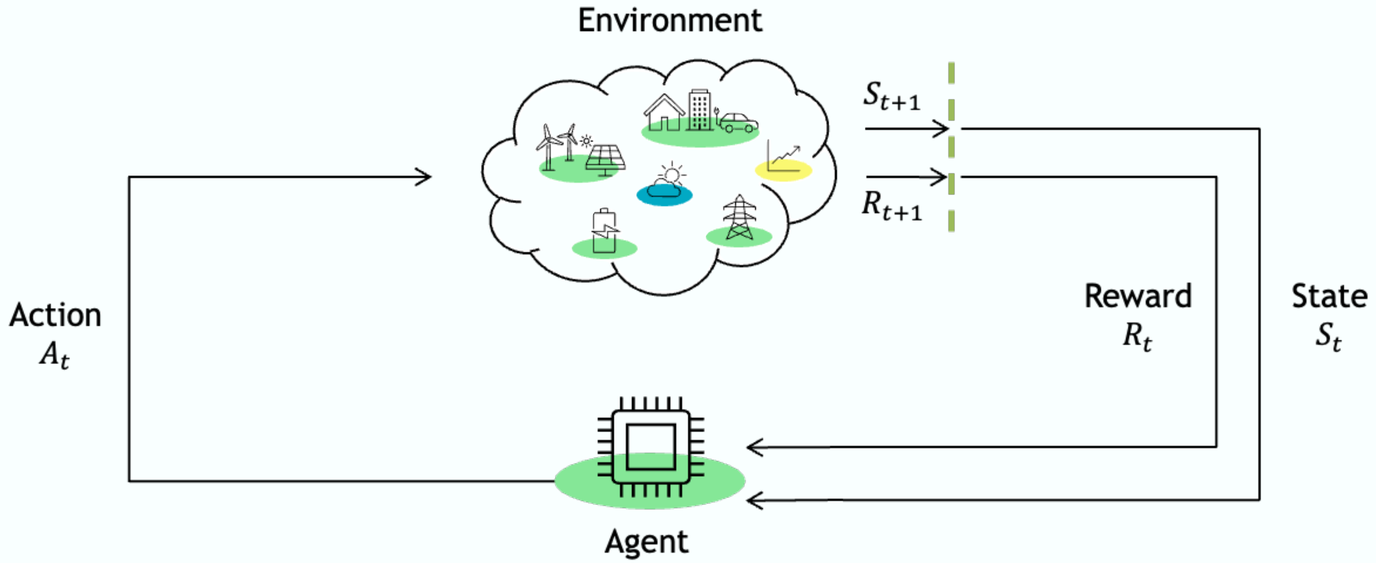
Smart Grids and Energy Storage Systems

Distributed and decentralized scenario conceived as a control problem involving the collaboration/competition of **multiple agents** (*prosumers*).

Many variables and uncertaintes:

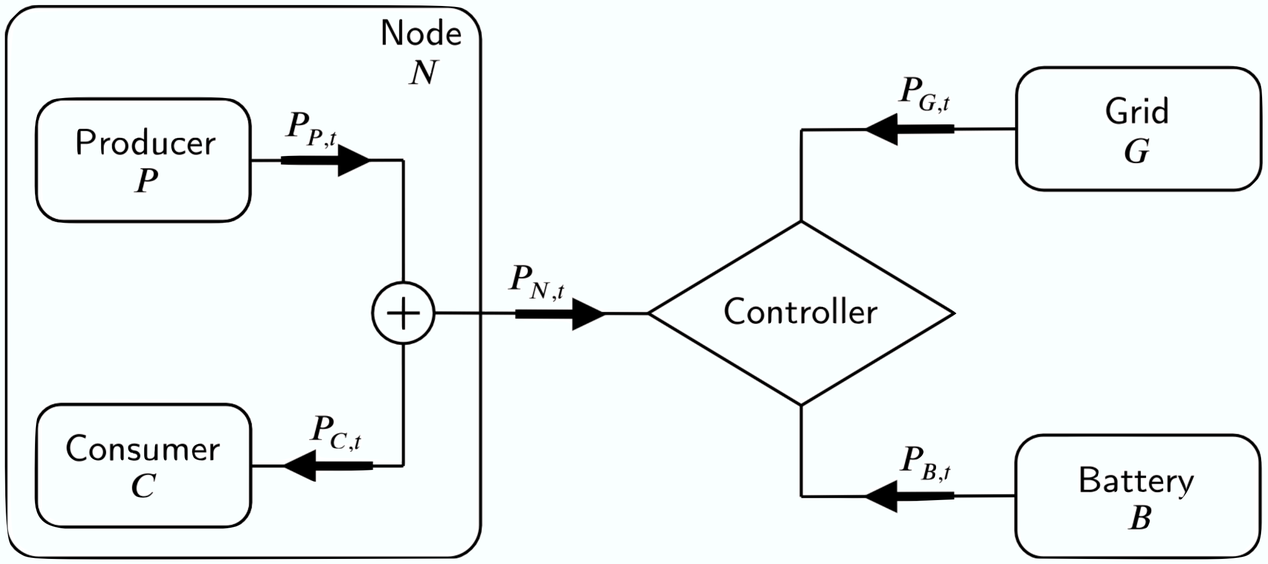
- consumers' demand
- energy generation
- battery degradation
- electrical market trend

The environment's dynamics could be too complex to be modeled.



Application in Energy Field

Smart Grids and Energy Storage Systems



Marco Mussi, Luigi Pellegrino, Oscar Francesco Pindaro, Marcello Restelli, Francesco Trovò, A Reinforcement Learning controller optimizing costs and battery State of Health in smart grids, *Journal of Energy Storage*, 1, Volume 82 (2024)

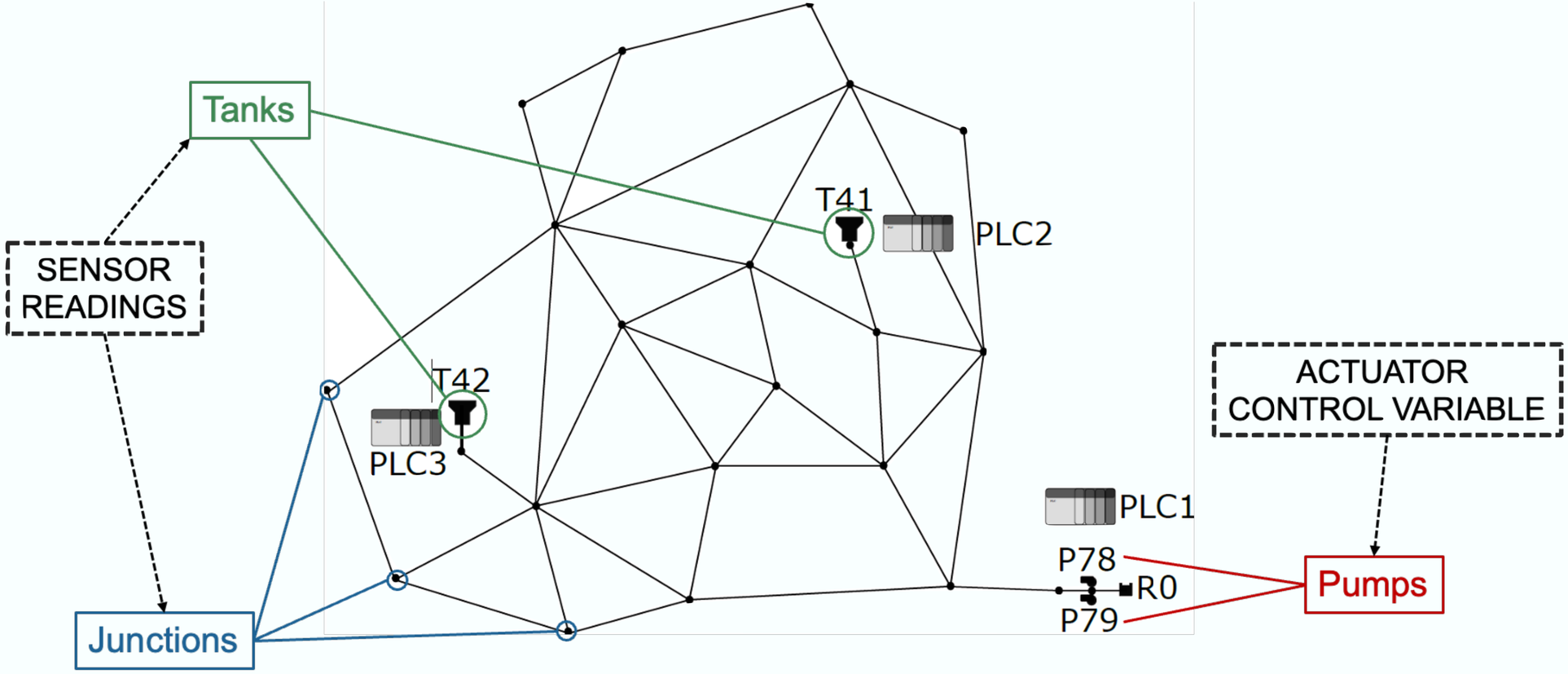
The **goal** consists in minimizing the cost of the system and maximizing profits selling energy to the grid.

The **state space** include information on SoC, temperature, exchanged power and time.

The **action space** is a finite set of possible fractions of power stored by the battery.

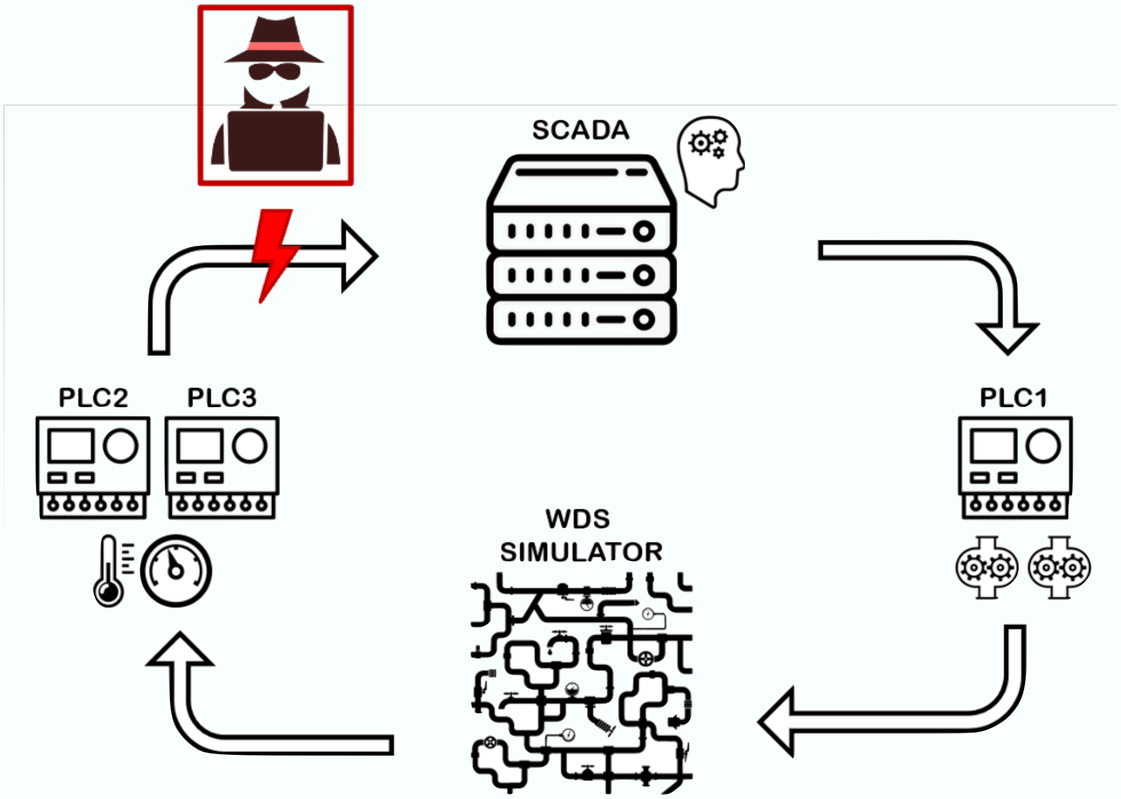
Application in Distribution Networks

Water Distribution Systems undergoing Cyber-Attacks



Application in Distribution Networks

Water Distribution Systems undergoing Cyber-Attacks



The **goal** consists in make the system resilient to cyber-attacks and avoid interruptions of the network's normal operations.

To do so we have adopted a deep-RL algorithm (DQN) together with an attack detection system.

The tests have been conducted on a suitable Digital Twin (DHALSIM).

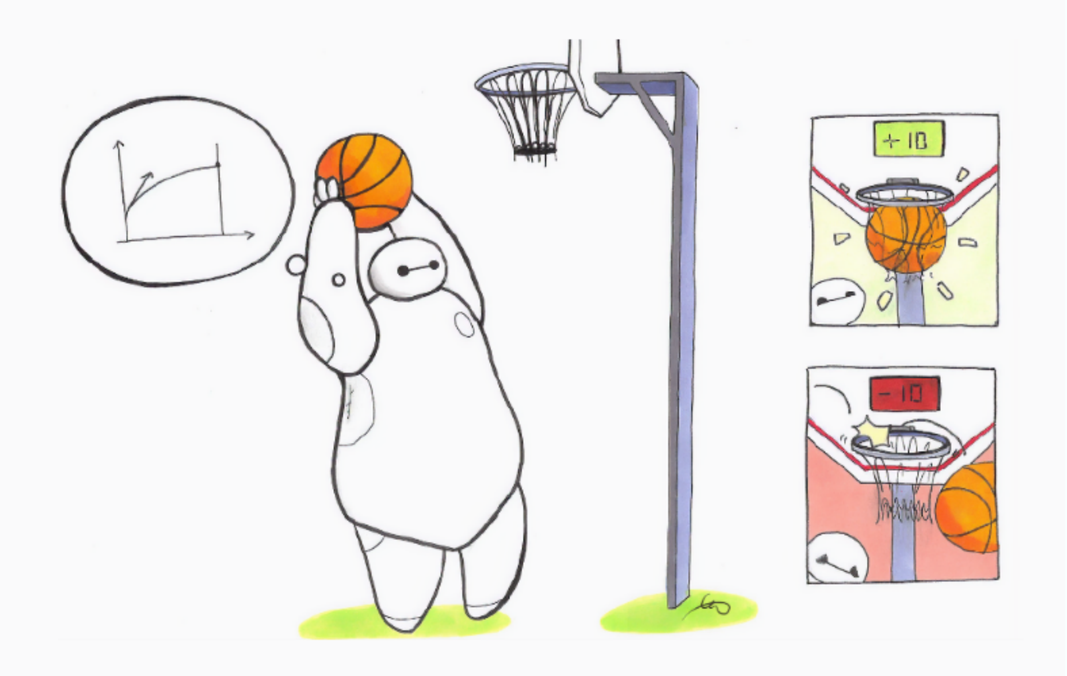
AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

RSE Ricerca Sistema Energetico | we move rsearch

Do try this at home…

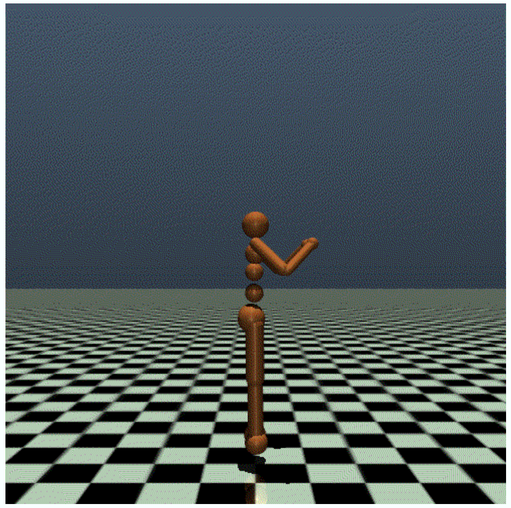
Python Libraries for Algorithms

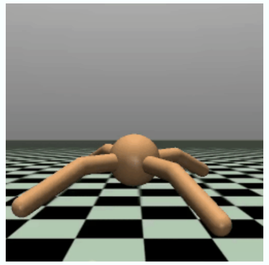
- Stable Baselines 3
- RL Lib
- Mushroom-RL

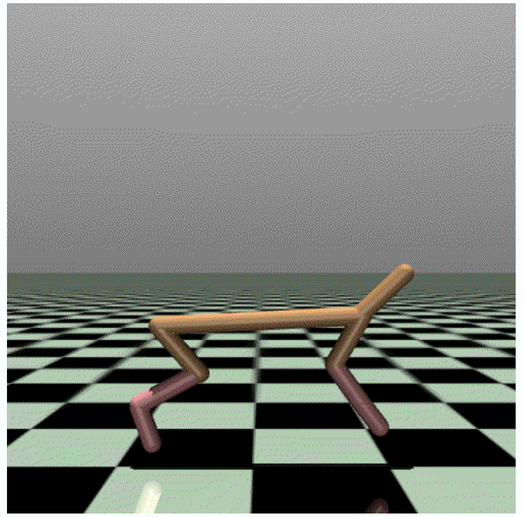


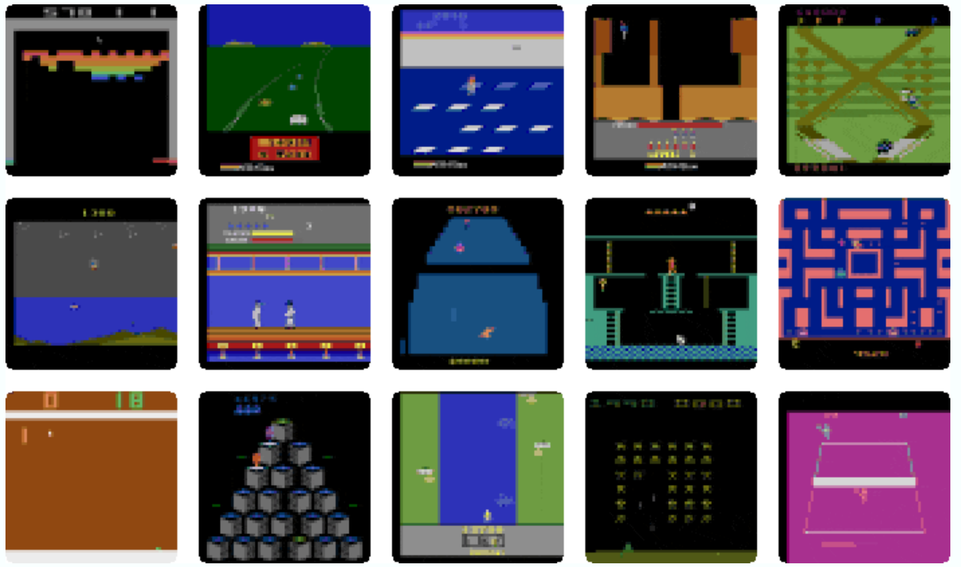
Python Libraries for Environments

- Gym/Gymnasium
- Atari-RL
- Mujoco
- SMAC









Gymnasium

Books and Courses

- Sutton and Barto, “*Reinforcement Learning: an Introduction*”, MIT Press, 2018. http://incompleteideas.net/book/the-book-2nd.html
- Bertsekas and Tsitsiklis, “*Neuro–Dynamic Programming*”, Athena Scientific, 1996.
- Szepesvari, “*Algorithms for Reinforcement Learning*”, Morgan and Claypool, 2010.
- Agarwal, Jiang, Kakade, and Sun, “*Reinforcement Learning: Theory and Algorithms*”, 2021. https://rltheorybook.github.io/
- Bertsekas, “*Dynamic Programming and Optimal Control, Vol.II, 4th Edition: Approximate Dynamic Programming*”. Athena Scientific, 2012.

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

RSE Ricerca Sistema Energetico | we move rsearch

Thank you for listening!