Introduction to Reinforcement Learning

Reinforcement learning – LM Artificial Iintelligence
(2022-23)

Alberto Castellini
University of Verona

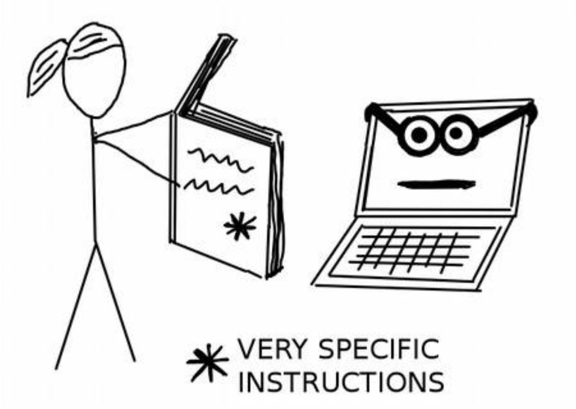
Summary

- What is Reinforcement Learning
- Examples and Applications of Reinforcement Learning
- Elements of Reinforcement Learning
- An extended example: Tic-Tac-Toe
- History of Reinforcement Learning

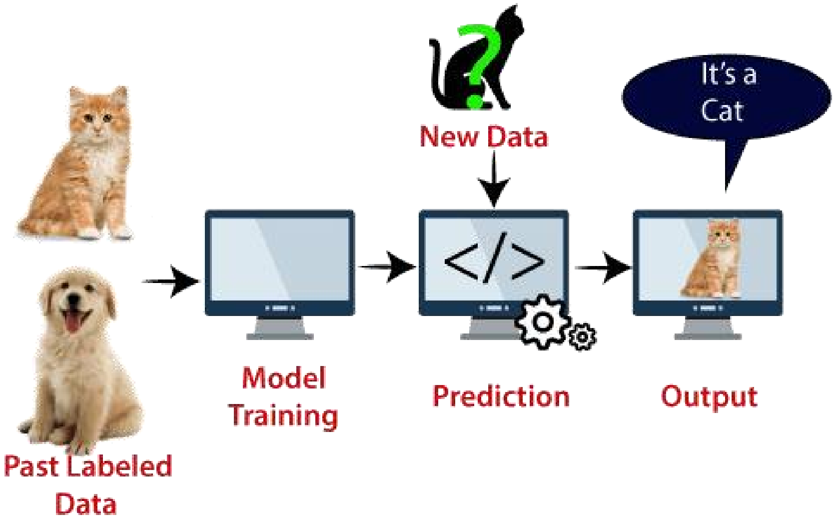
What is Reinforcement Learning

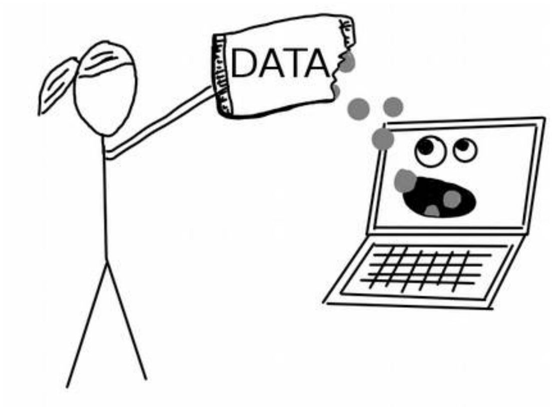
From traditional computer science to machine learning

- **Traditional computer science:** computers are programmed for every task they have to perform (notice: a program is a function)



- **Machine learning paradigm:** **examples** are provided to machines and machines learn to perform tasks based on examples (notice: a model is a function)





The nature of learning

- **Nature of learning: we learn by interacting with our environment**

- Infants have **no explicit teacher** but direct **sensorimotor connection** to the environment

- Exercising this connection produces information about
  - **cause-effect relationships** (consequences of actions)
  - what to do to achieve **goals**

Learning from interaction

Learning to drive a car

Learning to hold a conversation

- We are **aware** of **how our environment responds** to what we do and we seek to **influence what happens** through our **behaviour**

- **Learning from interaction:** foundational idea underlying nearly all theories of learning and intelligence

We explore
Reinforcement Learning
a computational approach to **learning from interaction**

- We evaluate the effectiveness of various **learning methods**
- We adopt the perspective of **artificial intelligence**
- We explore **designs for machines** that are effective in solving learning problems

Reinforcement learning: definitions and features

Reinforcement learning:

- **Learning what to do** so as to **maximize** a numerical **reward signal**
- Learn how to **map situations** to **actions**

Two most important features of RL:

1) Trial-and-error search

The learner is **not** told which **action** to take in each **situation** (as in **supervised learning**)

but

it must **discover** which action yields the **most reward by trying** them

2) Delayed reward:

Actions may affect **not** only the **immediate reward**

but

also the next situation and, through that, all **subsequent rewards**

Reinforcement learning: problem, solution methods, research topic

RL is simultaneously:

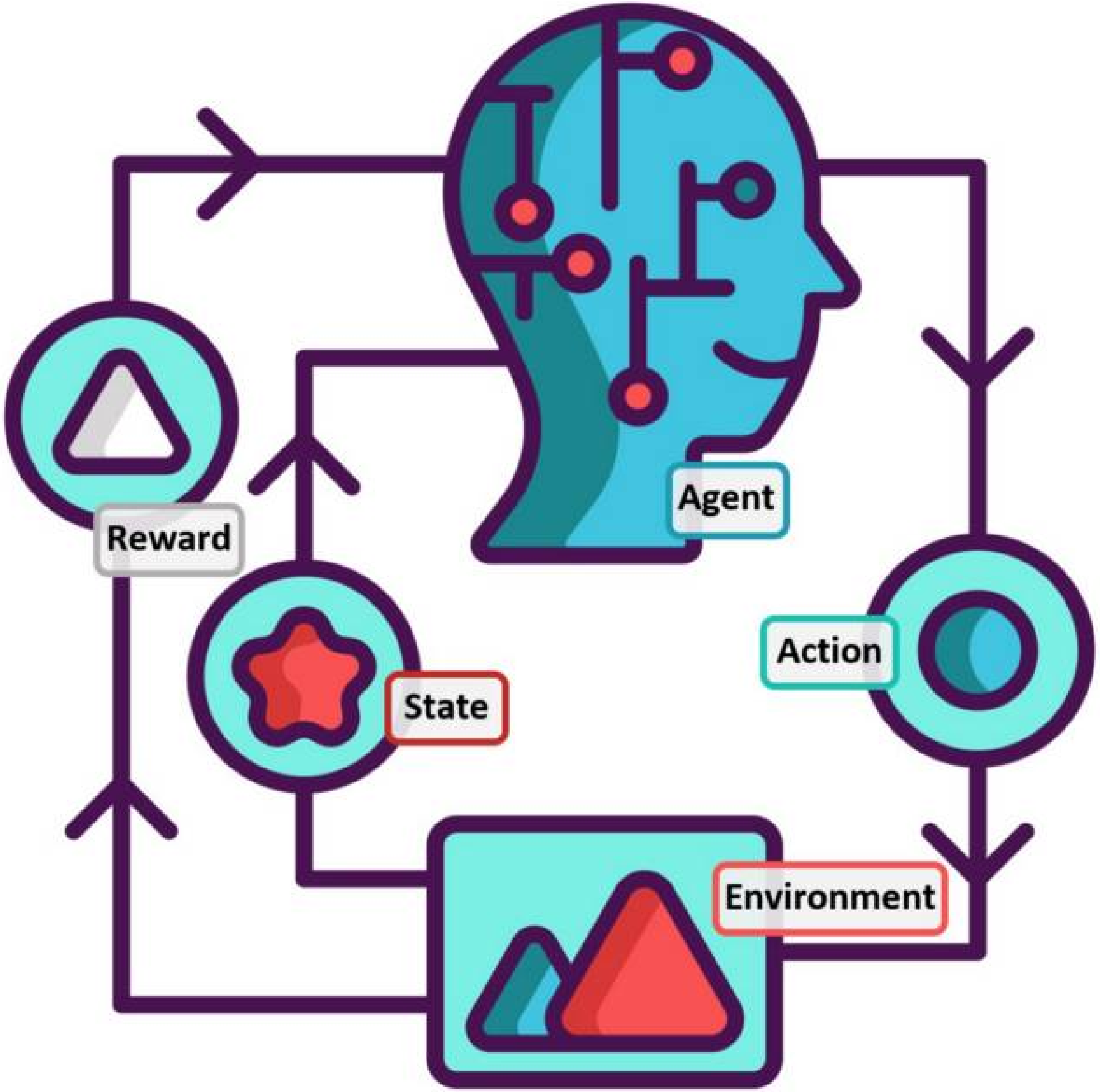
- a **problem**
- a **class of solution methods**
- a **research field** that studies this problem and its solution methods

- It is important to **distinguish** the three to avoid confusion

Problem of reinforcement learning. Ideas from:

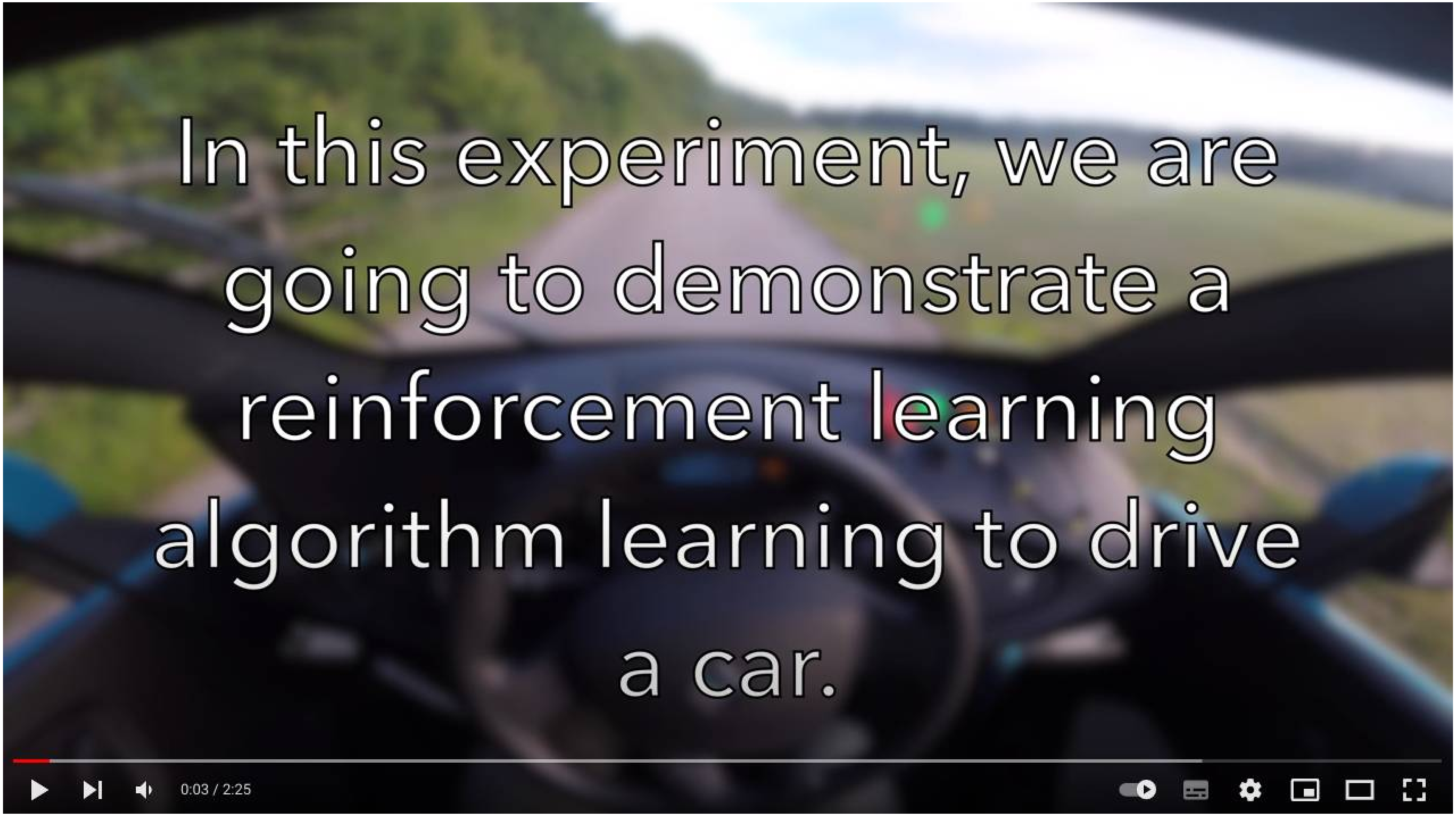
- Dynamical systems theory
- Optimal control of incompletely-known Markov Decision Processes

Reinforcement learning problem: main ingredients



All these elements are present in **Markov Decision Processes (MDP)** which we introduce in the next lectures

An example: learning to drive



https://youtu.be/eRwTbRtnT1I

Reinforcement Learning vs Supervised Learning

Reinforcement Learning is different from **Supervised Learning**

- **Supervised learning:** learning from a training set of **labeled samples** (external supervisor)

- Each **sample**:

  situation → correct action (label/category)

- **Object** of supervised learning:

  To generate an agent able to **generalize** its **responses** to act correctly in **situations not present** in the **training set**

- **Not adequate** for learning from **interactions**
- In interactive problems it is **impractical to get informative training sets.** The agent should learn from its own experience

Reinforcement Learning vs Unsupervised Learning

Reinforcement Learning is different from **Unsupervised Learning**

- **Unsupervised learning:** finding structure hidden in collections of unlabelled data

- **Reinforcement learning** tries to **maximize a reward** signal instead of trying to find a hidden structure in the dataset

- Discovering a **structure** in an agent’s experience is **useful** but it does **not** address the problem of **maximizing reward signal**

Reinforcement learning is a third machine learning paradigm alongside supervised learning and unsupervised learning

Exploration-exploitation trade-off in RL

- A **key challenge in RL** (not present in other kinds of learning): optimization of the **trade-off** between **exploration** and **exploitation**

- **Exploration:** select **actions never tried** to a situation to learn what happens (i.e., how much reward they provide)
  - Risky in terms of reward acquisition
  - Informative about environment dynamics and reward acquisition

- **Exploitation:** select **actions already tried** in the past and found to be effective in producing reward
  - Safe in terms of reward acquisition
  - Not informative

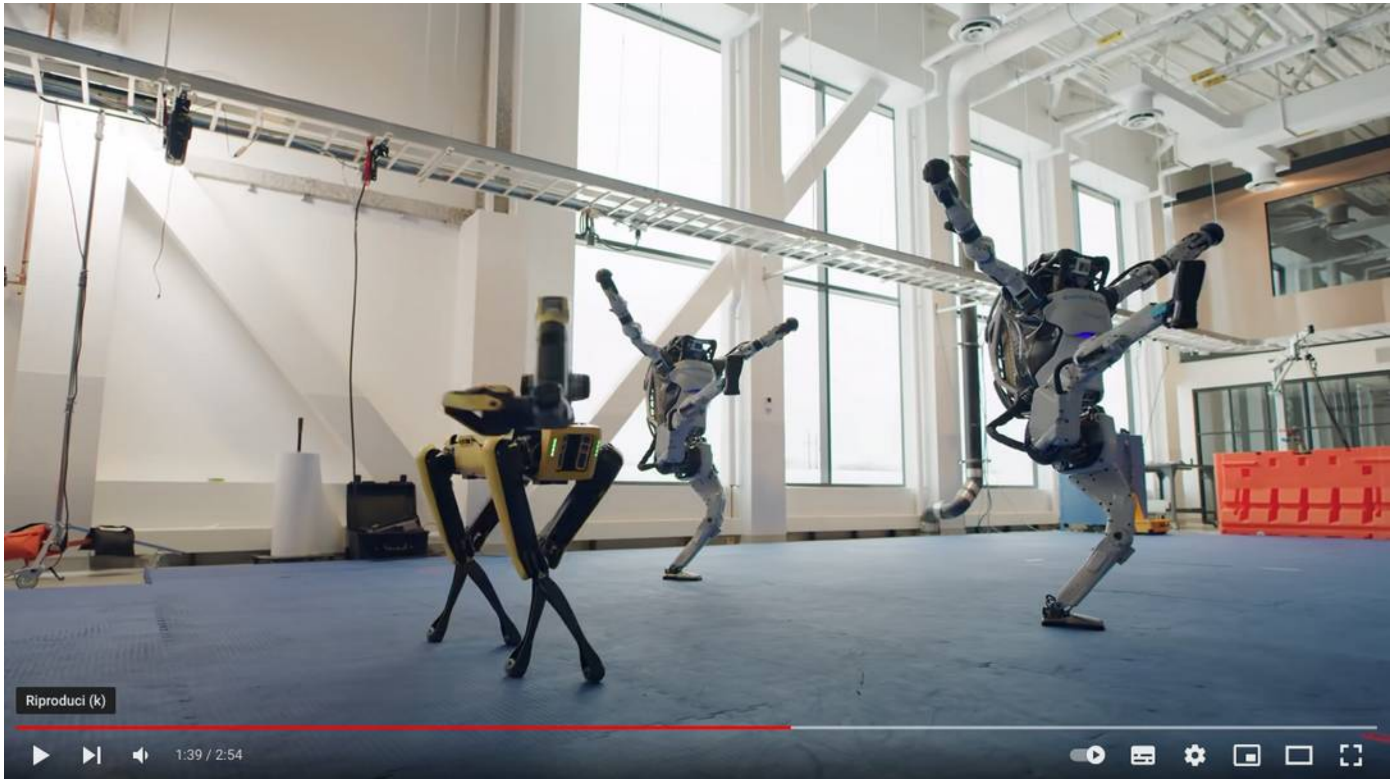
Exploration-exploitation dilemma

- **Exploration-exploitation dilemma: should I explore or exploit?**
- The agent must **try** a variety of actions and progressively **favour** those that appears to be the best
- On **stochasic tasks** each action must be tried **many times** to gain a reliable estimate of its expected reward
- The exploration-exploitation dilemma is still **unsolved**
- The exploration-exploitation dilemma is not present in supervised and unsupervised learning

RL agents as components of larger systems

- An RL **agent** can be also a **component of a larger system** (e.g., agent that monitors the **charge level** of **robot's battery**)

- In this case the agent's environment is the rest of the robot together with the robot's environment



https://youtu.be/fn3KWM1kuAw

https://youtu.be/tF4DML7FIWk

Interaction between RL and other disciplines

- RL has substantive and fruitful **interactions** with other **scientific and engineering disciplines**

- Some examples

Artificial Intelligence	**Operations reseach**
Machine learning	**Control theory**
Statistics	**Psycology**
Optimization	**Neuroscience**

Examples and applications of RL

Examples and applications of RL

- A master **check** player makes a move (IBM's Deep Blue vs Kasparov - 1997)

- **AlphaGo** reached superhuman performance in the game of Go (2016)

- **Game** playing (Atari, Backgammon, Blackjack, Tic-tac-toe,...)

Situations? Actions? Rewards?

Examples and applications of RL

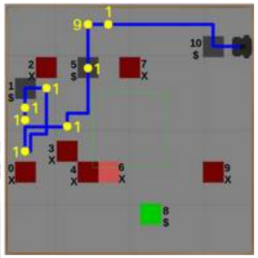
- An adaptive controller **adjusts parameters** of a **petroleum refinery's operation** in real time (control of cyber-physical systems)

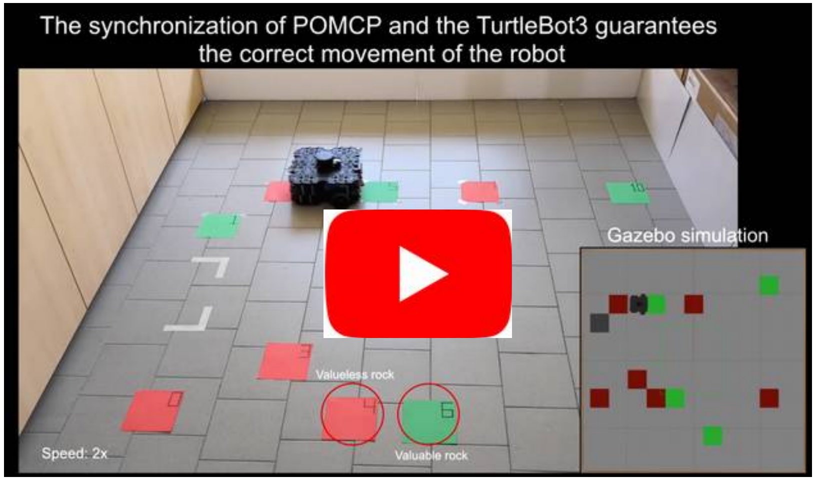
- A **mobile robot** decides whether to enter a new room in **search of more trash** to collect or to move back to a **battery recharging** station

Situations? Actions? Rewards?

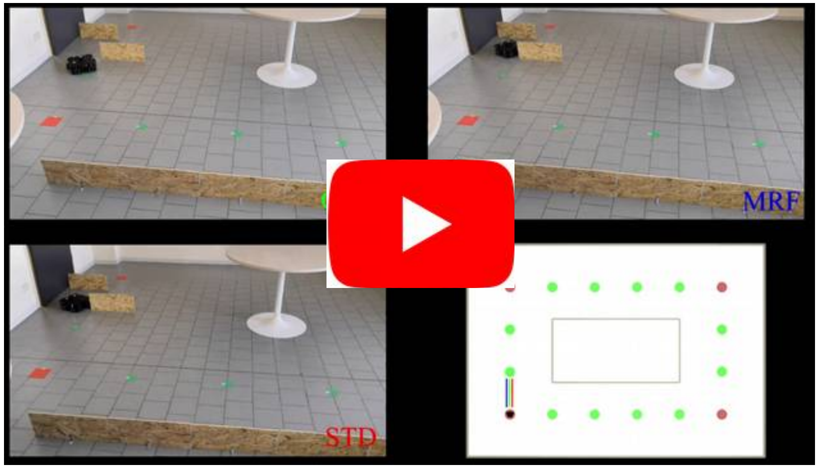
- **Robot planning/control** in robotic/industial environments (e.g., **Projects @ISLa**)

Intelligent System Lab





Rocksample with a Turtlebot



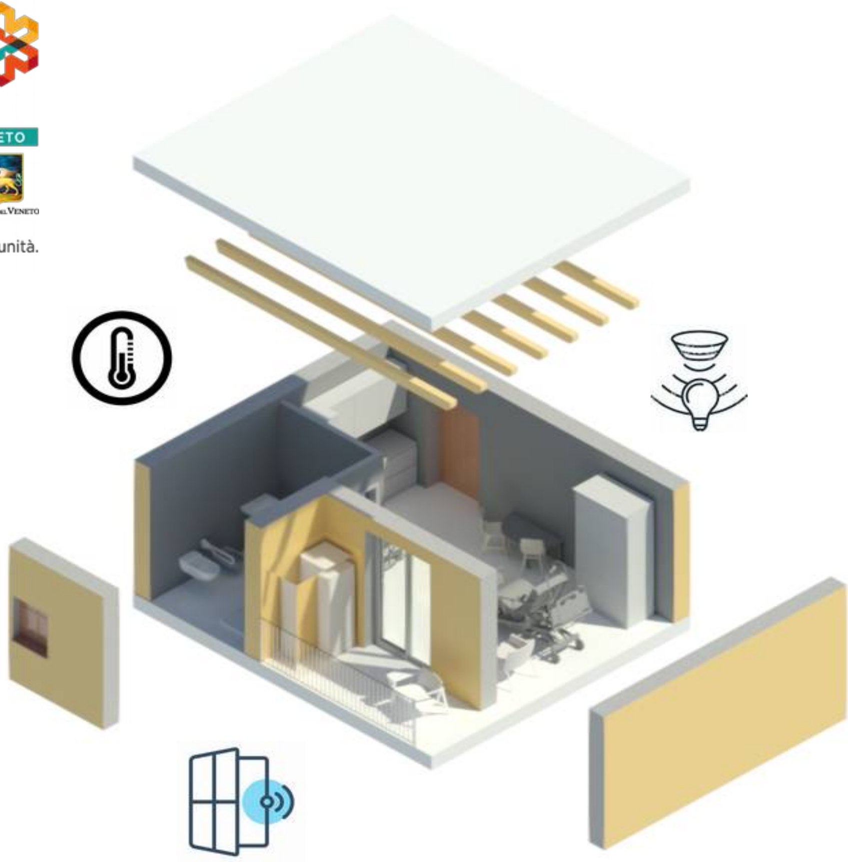
Velocity regulation of a Turtlebot

Pick up and delivery with a Kairos in the ICE lab

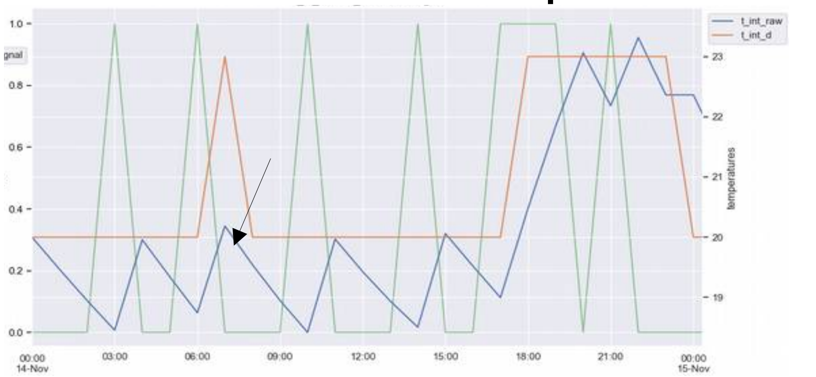
- An autonomous agent **controls air quality** and thermal comfort in a **smart building** (e.g., **Ghotem** and **Safe Place project @ISLa**)

Intelligent System Lab

Un moltiplicatore di opportunità.
Da non lasciarsi sfuggire.



Before environment adaptation



After environment adaptation

Examples and applications of RL

- Control of autonomous surface vehicles (UAV)

(e.g., **Incatch project @ISLa**)

- Autonomous cars

https://youtu.be/GYwPNMAgF-Q

- Helicopter control (UAV)

https://youtu.be/0JL04JJjocc

- Quadcopter control (UAV)

https://youtu.be/w2itwFJCgFQ

Situations? Actions? Rewards?

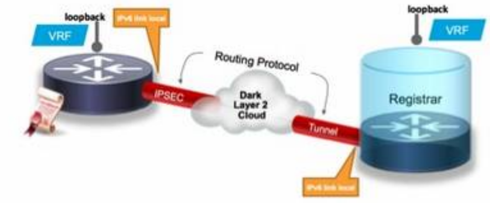
Examples and applications of RL

- **Operations research** (pricing, vehicle routing)

- **Spoken dialog systems** (e.g., chatbot)

- Data center **energy optimization**

- Self-managing **network systems**



- Computational **finance**

Situations? Actions? Rewards?

Features shared among examples

- **Interaction** between agent and environment
- The agent has a **goal**
- **Uncertainty** about the environment (effects of actions cannot be fully predicted)
- Actions performed by the agent affect the **future** state of the environment
- Presence of indirect and **delayed consequences** of actions
- The agent can use its experience to improve its performance over time (adjusting behaviour) → **Adaptation**

Elements of RL

Elements of RL: policy, reward signal, value function, model

1) Policy: defines the agent's way of behaving at a given time and in a given situation

Policy Function: state → action

It may be implemented as

- a function
- a lookup table
- a search process

It may be **deterministic** or **stochastic**

Its generation is the **target of Reinforcement Learning**

Elements of RL: policy, reward signal, value function, model

2) Reward signal: defines the goal of the RL problem

At each step the environment sends to the agent a single number called reward (i.e., **immediate pleasure/pain**)

The agent’s objective is to **maximize the total reward over long runs**

Reward signals may be **deterministic** or **stochastic** functions of the **state** and the **action**

Reward Function: state, action $\rightarrow$ reward

Elements of RL: policy, reward signal, value function, model

3) Value function: specifies **what is good in the long run** (while the reward focuses on what is good **immediately**)

The **value** of a **state** is the **total amount of reward** an agent can expect to accumulate over the future, starting from that state

State Value Function: state → value

The **value** of a **state-action** pair is the **total amount of reward** an agent can expect to accumulate over the future, performing the action from that state

State-Action Value Function: state, action → value

It is much harder to determine values than rewards. Hence, **methods for efficiently estimating values are key elements of RL algorithms**

Elements of RL: policy, reward signal, value function, model

4) Model of the environment: is a mathematical model that mimics the behaviour of the environment and allows to infer it

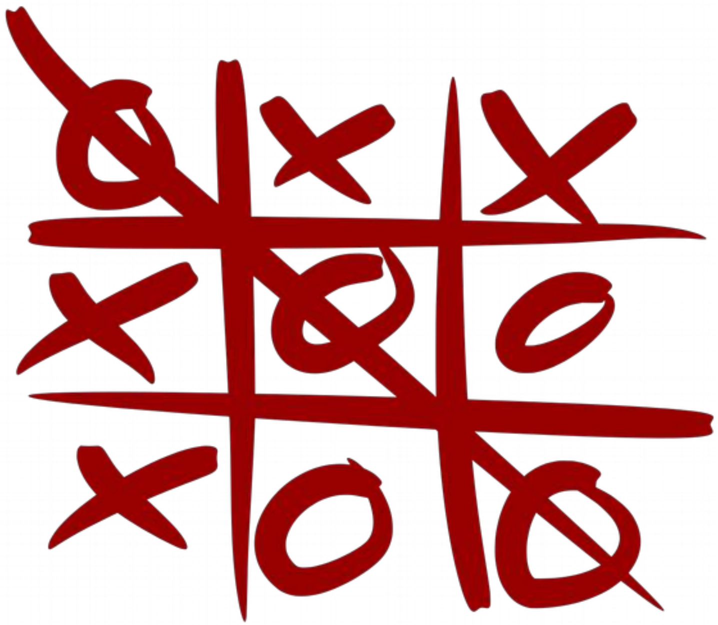
Environment model: state, action → next state/reward

Models are used for **planning**, i.e., chosing immediate actions considering possible future situations

- **Model-based RL methods:** use an explicit representation of the model of the environment to select the best actions
- **Model-free RL methods:** do not use the model of the environment but are explicit trial-and-error learners

An extended example: Tic-Tac-Toe

An extended example: Tic-Tac-Toe



- Assume to play against an **imperfect player**
- We want to **construct a player** that **finds the imperfections** in its opponent and learns how to **maximize** its **chances of winning**

An extended example: Tic-Tac-Toe

- Problem: classical techniques
  - **minimax** solutions from game theory
  - classical optimization methods (e.g., **dynamic programming**)

  need a complete specification of the opponent to work

- **This information can be estimated from experience,** by playing many games against the opponent

- Idea: **learn the model of the opponent's behaviour** from experience, then apply **dynamic programming** to compute an optimal solution (not that different from what RL does)

An extended example: Tic-Tac-Toe

Evolutionary methods would search the space of all possible policies.

- For each policy considered in the population: winning probability is computed by playing some games against the opponent
- Better policies are selected during the evolution
- Hill-climbing in policy space

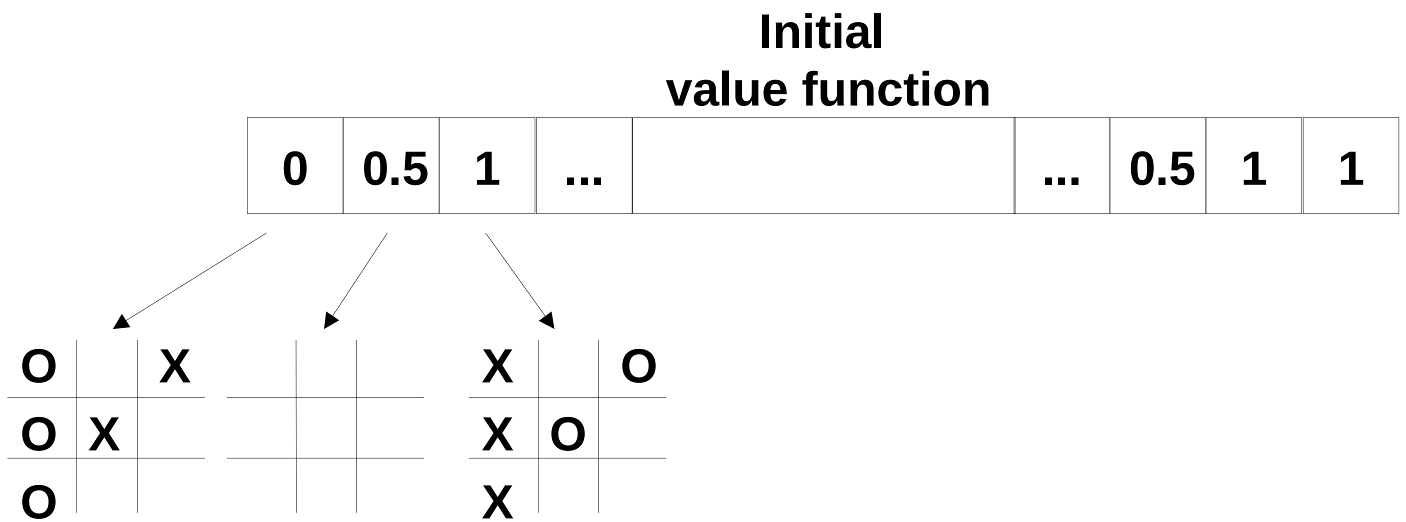
This method could **find the best policy** but it is often **inefficient**

An extended example: Tic-Tac-Toe

A method using **value function**

- We set up a **table** with a number (value) **for each state**

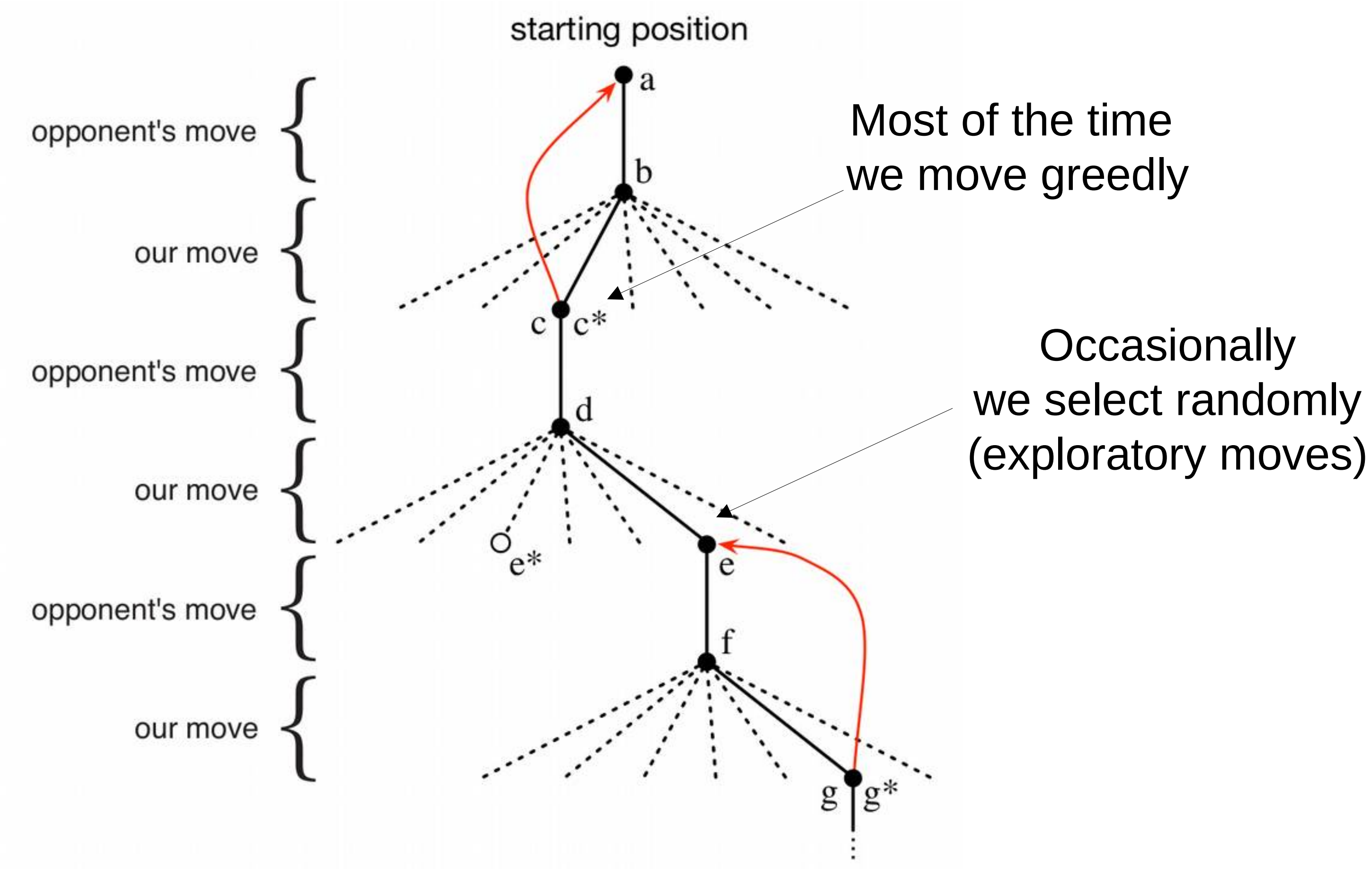
- The number is the latest estimate of the probability of winning from that state (0 if state=loss, 1 if state=win, 0.5 otherwise)



An extended example: Tic-Tac-Toe

- We play many games against the opponent

- To **select** our **moves** we examine the states that would result from each possible move and **look up** their current value in the table



An extended example: Tic-Tac-Toe

- While playing we **change the values of states** in which we find ourselves making them more accurate

- To this aim, we **back up** the value of the **state after each greedy move** (red arrows in the slide before)

- **Value update rule: The current value of the earlier state is updated to be closer to the value of the later state. In particular, we move the earlier state's value a fraction on the way toward the value of the later state**

- Let S_t be the state before the greedy move, S_{t+1} the state after the greedy move, $V(S_t)$ the value of state S_t and α a small fraction (*step size*), then the update rule is:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[V(S_{t+1}) - V(S_t)\Big]$$

Example of temporal-difference learning rule

An extended example: Tic-Tac-Toe

- This method performs well

- **If the step-size parameter is reduced properly over time this method converges, for any fixed opponent, to the true probabilities of winning from each state given optimal play by our player**

- Namely, the method **converges to an optimal policy** for playing against the **specific opponent**

- If the step-size parameter is not reduced to zero over time the policy can play well also against opponents that slowly change

- Notice: the tic-tac-toe player is **model-free**

An extended example: Tic-Tac-Toe

Both evolutionary and value function methods search the space of policies but:

- **Evolutionary methods** use a **fixed policy** for several games to evaluate it in an unbiased way. What happens **during the games is ignored**
- **Value function methods**, in contrast, allow **individual states to be evaluated**, hence they **take advantage of information available during the course of play**

This makes value function methods **more efficient** in several cases

General Applicability of RL

- RL methods work also in problems with **no external opponent** (game against the nature/**environment**)

- RL methods are applicable also to **non-episodic** problems

- RL methods are applicable also to **continuous time** problems

- RL methods can be used also in problems with very **large or infinite state spaces** (e.g., backgammon, 10^{20} states (Tesauro, 1992) using RL with artificial neural networks → next semester)

- **Prior knowledge** can be incorporated into RL

- RL can be used also in problems in which the state is **partially observable**

Hystory of RL

Hystory of RL: a summary

Three main **threads** of RL in its early history:

1) Optimal control with value functions and dynamic programming

2) Learning by trial-and-error: psycology of animal learning

→ Brought to some of the earliest works in artificial intelligence

3) **Temporal-difference methods**

The three came together in the late **1980s** producing the modern **field** of Reinforcement Learning

Hystory of RL: Optimal control thread

- **1950:** the term **optimal control** came into use to describe the problem of designing a controller to minimize or maximize a measure of a dynamical system's behaviour over time

- **Mid-1950s: Richard Bellman** developed an approach based on **value functions** for this problem extending the nineteenth century theory of Hamilton and Jacobi (**Bellman Equation**)
  - **Dynamic programming** (1957)
  - **Markov Decision Processes** (MDPs)

- **1960:** Ronald Howard devised the **policy iteration** method for MDPs

- **Connections** between **optimal control** based on dynamic programming and **learning** were slow to be recognized, possibly because of the nature of dynamic programming (it procees backwards, it needs complete knowledge of the dynamics)

Hystory of RL: Optimal control thread

- **1977: Ian Witten**'s work combines learning and dynamic-programming ideas
- **1989: Chris Watkins**' work on **Q-learning** represents the first **full integration** of dynamic programming and online learning
- Since then, these relationships have been extensively developed by many researchers
- **1996: Dimitri Bertsekas** and **John Tsitsiklis** combined dynamic programming and **artificial neural networks** (neurodynamic programming, approximate dynamic programming)

Hystory of RL: Optimal control thread

- **Optimal control** is part of **reinforcement learning** althought dynamic programming (DP) needs complete knowledge of the environment

- Like learning methods **DP algorithm** gradually **synthesize** the **policy** through **successive approximations**

- Similarities are very strong

- The theories and solution methods for complete and incomplete knowledge are so closely related that they can be considered as part of the same subject matter

Hystory of RL: Trial-and-error learning thread

- **Late 1800s:** Alexander Bain and Conway Morgan first used the idea of trial-and-error learning in studies of animal behaviour

- **1927:** Edward Thorndike used the term “reinforcement” in the context of animal learning

- **1948: Alan Turing** described a “pleasure-pain system” representing the first idea to implement trial-and-error learning in a computer (**artificial intelligence**)

- Many **electro-mechanical machines** were constructed that demonstrated trial-and-error learning (see http://cyberneticzoo.com/cybernetic-time-line/)

Hystory of RL: Trial-and-error learning thread

- **1961: Marvin Minsky**'s paper "Step towards Artificial Intelligence" discussed issues relevant to trial-and-error learning:
  - prediction
  - expectation
  - Basic **credit-assignment problem** for complex reinforcement learning systems: how do you distribute credit for success among many decisions that may have been involved in producing it?

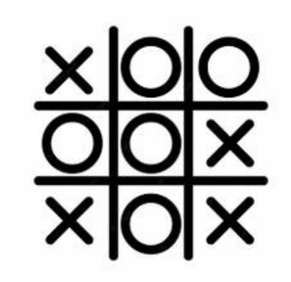
  → All methods discussed in this course are directed toward solving this problem

- **1961-1963: Donald Michie** described a simple trial-and-error learning system for learning how to play tic-tac-toe (MENACE: Matchbox Educable Naughts and Crosses Engine)

https://youtu.be/G-di38Fpgdw

Hystory of RL: Trial-and-error learning thread

- **1968: Michie and Chambers** described another **tic-tac-toe** reinforcement learner called **GLEE** and a reinforcement learning controller called **BOXES**. BOXES was applied to **balance a pole hinged to a movable cart**

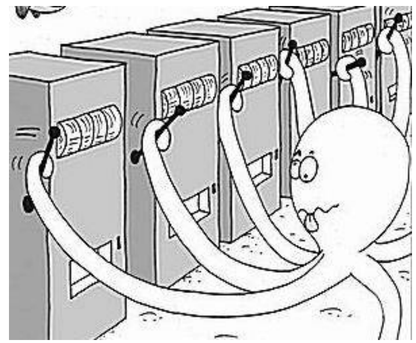


- **1973: Widrow, Gupta and Maitra** modified the **Least-Mean-Square (LMS)** algorithm to produce a reinforcement learning rule that could learn from success/failure signals instead of training examples (selective bootstrap adaptation, learning with a critic)

  → It can learn to play **blackjack**

Hystory of RL: Trial-and-error learning thread

- **1960s:** Research on **learning automata** directly influenced the trial-and-error thread. Methods for solving a nonassociative, purely selectional learning problem: **k-armed bandit** (analogy with a slot machine)



- 1970s: **Statistical learning** theories developed in **psycology** were adopted in **economics**, leading to a thread in that field devoted to RL
  → Reinforcement learning in the context of **game theory** (John Nash)

- **1975: John Holland** work on trial and error with **evolutionary methods**

Hystory of RL: Temporal difference thread

- **1950s:** origin of temporal-difference learning in animal learning psycology (secondary reinforcers)

- **1959: Arthur Samuel** first implemented a learning method based on temporal-difference ideas, as part of his **checkers-playing** program

  → Inspiration from **Claude Shannon's (1950)** sugestion that "*a computer can be programmed to use an evaluation function to play chess, and it might be able to improve its play by modifying the function online*"

- **1972. Klopf** brought **trial-and-error** learning together with **temporal-difference** learning to scale learning to large systems

- **1981: Sutton** and **Barto** developed the **actor-critic architecture**, which combines temporal-difference trial-and-error learning
- **1988: Sutton** separated temporal-difference **learning** from **control**

References

- R. S. Sutton, A. G. Barto. Reinforcement learning, An Introduction. Second edition. Chapter 1