Unit 1: Modelling an Intelligent agent.

Why, how, and for what is it interesting?

What is an agent?

An agent is **autonomous**: in some way, this sw is able to self-control its own behaviour.

This definition, although helps, is too wide open (a thermostat is also an agent!).

An **intelligent agent** is a sw able to do autonomous **flexible** actions in an environment

Flexible implies this sw to show **reactive**, **proactive** and **social** behaviour.

What means reactive?

To hold a continuous interaction with the environment, to answer back to the detected changes in a suitable way.

What means pro-active?

To pursue the sw designed goal, taking the initiative. To show an opportunistic behaviour.

What has to prevalence, a reactive or proactive behaviour?

What means to be social?

To be able to interact with other agents (or even humans) by using some kind of formal agent communication language.

Interactions are driven by the pro-activity of each involved agent, its design goals, and overall behavior.

What is an environment?

It is the (physical or virtual) medium common to all agents. Agents get events from the environment and they also act on the environment.

Environment classification:

- Accessible vs *not accessible.*
- Determinist vs *not determinist.*
- Episodic vs *not episodic.*
- Static vs *dynamic.*
- Discrete vs *continuous*.

How to design *Intelligent* agents?

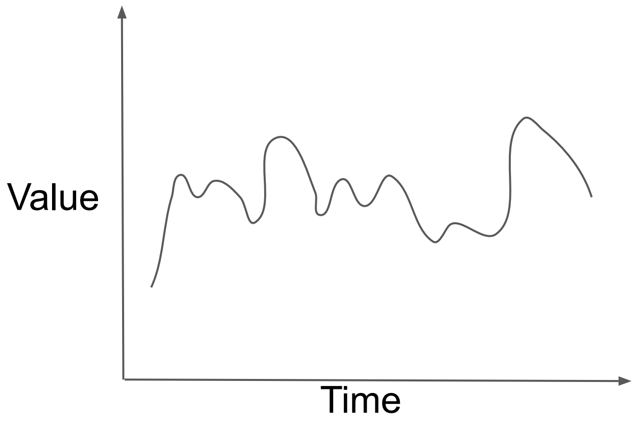
How *Intelligence* can be approached?

1. Intelligence is a consequence of a statistical model feed with an enormous quantity of pre-processed data (**machine learning**).
2. Intelligence **emerges** from the massive interaction of simple rules. It deals with intelligence as a kind of complex system (small modifications in the values of some variables provide very different behaviors).
3. Intelligence can be modeled by symbol manipulation procedures and mathematical logic. **Intentional systems**, from which BDI logic (and many extensions) is the most frequently used.
4. Intelligence is a hybrid of the previous models.

Machine learning

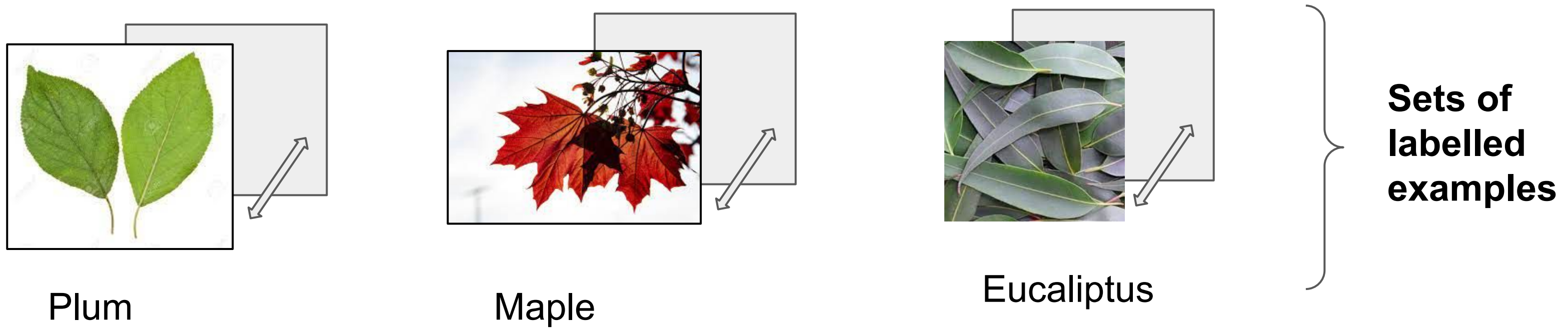
Machine learning is a discipline aims to design, understand and apply computer programs that learn from experience (i.e. data) for the purpose of modelling, prediction or control.

One of the main domains on using Machine learning is for *prediction problems*



What plates have these vehicles?

Supervised learning is one of the disciplines of machine learning. It assumes that some problems are very hard if we try to program “general” logical solutions (as traditional computer programs do). Alternatively, we can obtain more accurate “general” solutions by giving enough examples to the program about what it can be understood as good input data and outputs pairs.



The set of labelled examples are split into two sets:

- One for training the classifier or predictor (**training set**).
- Another one smaller for testing the trainer classifier or predictor (**validation set**).

Usually the set of possible classifiers is very very big. So, we have to look for one “enough” good.

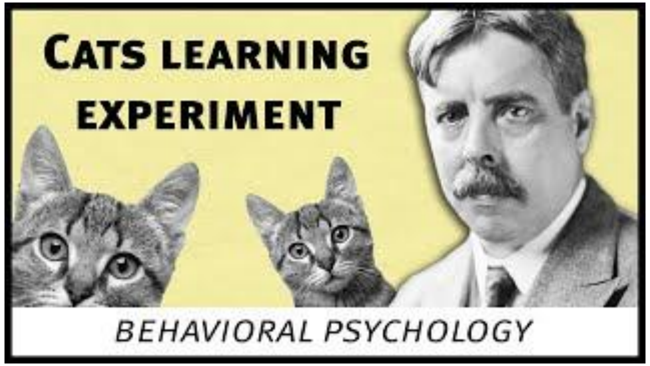
For measuring how good a classifier is the concepts of **error** and **generalization** are heavily applied.

Types of machine learning

- ❑ Supervised learning. Prediction based on examples of correct behavior.
- ❑ Unsupervised learning. Goal is discover a model behind provided data.
- ❑ Semi-supervised learning. A mix of the previous types.
- ❑ Active learning. Learn to query a user to label data with the desired outputs.
- ❑ Transfer learning. How to apply what was already learned to other agents.
- ❑ **Reinforcement learning**. Learning to act, not just predict; goal is to ***maximize the overall reward*** obtained while executing actions on a environment.
- ❑ And many more nowadays and to come …

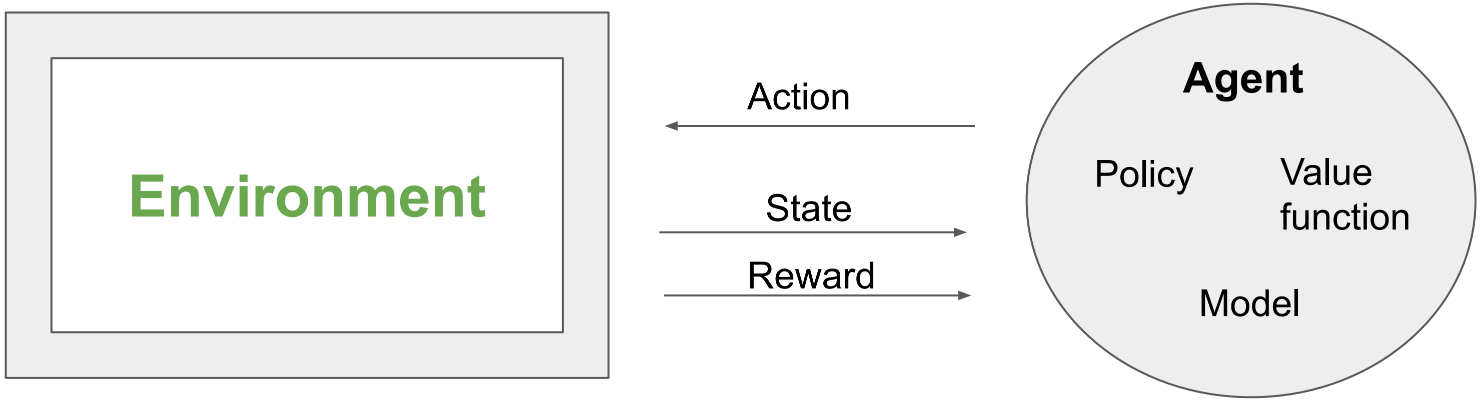
Reinforcement learning

Inspired by psychology studies with animals and infants



Which are the main elements common in these two examples?
Well, there is an agent (cat, infant), an action to perform by the agent (drawn from a set of available actions), an environment where the actions chosen by the agent are executed, and as consequence the environment changes, and also a reward (positive or negative) are given to the agent for each executed action.

Overall view of Reinforcement learning

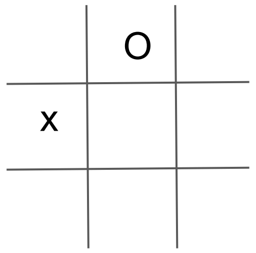


Policy: how to choose an action.
Action: performed by the agent on the environment.
Value function: to evaluate how well it is the current situation for the agent.
Model: an idealized representation of the real environment (should be useful for the goals of the agent).
State: the set of values that characterize the current situation in the environment.
Reward: the value returned by the environment to the agent in response to the consequences of the action performed.

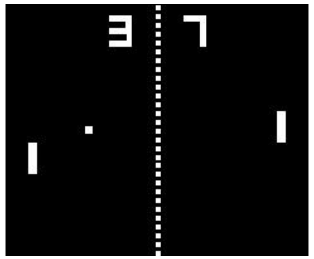
Overall view of Reinforcement learning

There are three great kinds of **States**:

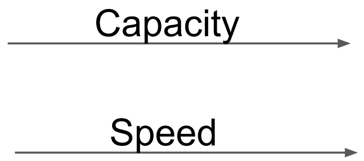
Discrete and small ones



Discrete and big ones



Continuous



Rewards:

- It uses to be a scalar value, strongly related to the goal or goals to be achieved by the agent.
- It is determined by the designer/programmer of the RL system.
- The notation r_t denotes the reward obtained at time t.

Overall view of Reinforcement learning

Actions: Can be

- Discrete (choose one among the N available actions)
- Continuous (an array of scalar values, e.g.)

Policy: A projection from the set of states to the set of actions. This projection can be:

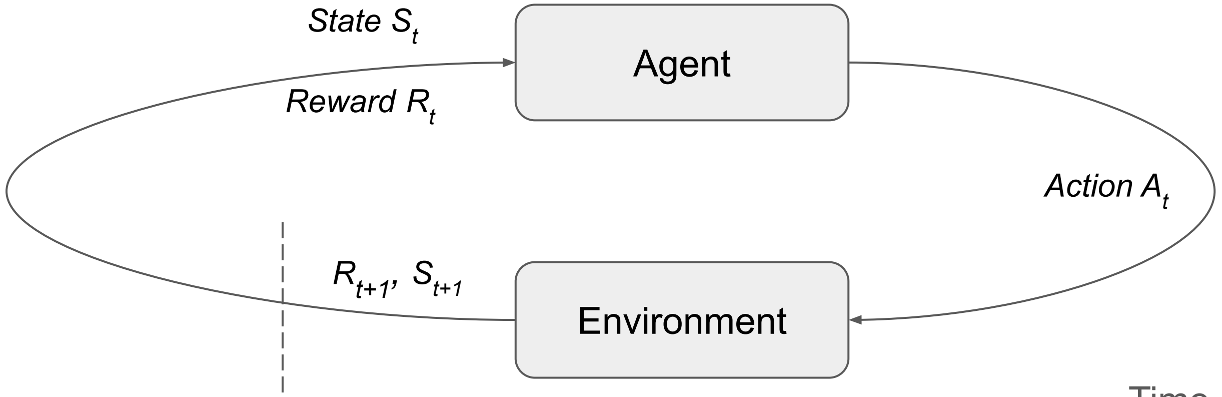
- Deterministic.
- Stochastic.

Value functions: Two types

- $V^{\pi}(s)$, the value that can be obtained accumulating the potential reward obtained from state s following policy π
- $Q^{\pi}(s,a)$, the value that can be obtained accumulating the potential reward obtained from state s, performing action a, and then, following policy π

The goal of RL is to learn a good policy without supervision.

Overall view of Reinforcement learning



Time t

$S_t \in S$ (The set of states)

$A_t \in A$ (The set of actions)

$R_t \in \mathbb{R}$ (The set of real numbers)

Returns

Is the long term accumulation of rewards starting from time step t $G_t = R_{t+1} + R_{t+2} + \ldots + R_T$

Another version, the discounted return, using a discount factor ϒ in [0,1] in continuous settings

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

There is an inherent recursive relationship between discounted returns in consecutive time steps:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \ldots) = R_{t+1} + \gamma G_{t+1}$$

Both cases can be notated with the same equation::

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \; with \; T = \infty \; xor \; \gamma = 1$$

Markov Decision Process (MDP)

Is a RL setting that satisfies the **Markov property**: the transition probabilities (at which state to go from state s if action a is executed) and rewards depend only on current state, and maybe also in the current action, and this fact remains unchanged *regardless of the history* that leads to the current state. If state representation is enough to figure out which action to take, then it is a Markov scenario (e.g. Tic-Tac-Toe)

Mathematically is a tuple of 4 elements: < S, A, T, R> where:

- S is the set of States,
- A is the set of actions available,
- T transition probabilities, T(s,a,s’) = P(s’|s,a), so that for each pair s and a, $\sum_{s' \in S} T(s,a,s') = 1$
- R is the reward function from (s,a,s’), (s,a) or s to a real number.

Example: Recycling robot

It can be approached as a **MDP:**

- **States**: [low, high] the level of the charge of the battery
- **Actions**: Described as available actions for each state
  - A(low) = {search, wait, recharge}
  - A(high) ={search, wait}
- **Rewards**: *1* for each can collected, *0* no can is collected, *-3* if stopped due to a lack of battery

And the transitions?

Example: Recycling robot

s	a	s’	p(s’|s,a)	r(s,a,s’)
high	seach	high	α	r_{search}
high	seach	low	$1-\alpha$	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	r_{wait}
low	seach	low	β	r_{search}
low	seach	high	$1-\beta$	-3
low	wait	high	0	r_{wait}
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	0

such that:

- α is the probability to go from high to high on search
- β is the probability to go from low to low on search
- r_{search} is the expected number of cans while searching
- r_{wait} is the expected number of cans while waiting

Exercise: Represent this table as a transition graph (to be solved in classroom)

How to solve a MDP? Value functions

They are one of the fundamental concepts in RL.
There are two types: as function of state or function on (state, action) pairs.
The goal is that these functions estimate how good is to be in a particular state, or take some chosen action in a particular state (by means of the expected return)
Value functions can be learned from experience

How to solve a MDP? Policies

They represent a mapping from state to probability of executing an action π(a|s)
RL methods show how policy changes on experience.

Value functions

Two types:

- State value function for policy π: $v_\pi(s) = \mathbb{E}[G_t | S_t = s]$
- State-Action value function for policy π: $q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$

Great!! They satisfy recursive relationships as G_t does.
For State value function this is:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \textstyle\sum_a \pi(a, s) \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma \mathbb{E}[G_{t+1} | S_{t+1} = s']] \\
&= \textstyle\sum_a \pi(a, s) \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

The value of a state is expressed in terms of the value of the successor states.

Optimal Value functions

What is the goal of a RL method? To find a policy that achieves a big reward in the long term.
But, by value functions, we can define a partial ordering over potential policies. How? Easy!

Policy π is defined to be better or equal to policy π' if $v_{\pi}(s) >= v_{\pi'}(s)$ for all possible s

For any task, there is **at least one** policy that is >= to all other policies. Therefore:

- This policy, π_*, is an optimal policy for the task.
- All optimal policies share the same optimal state value function

$$v_*(s) = \max_\pi v_\pi(s)$$

- All optimal policies share the same optimal action-state value function

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

Value functions: Bellman equations

$$v_\pi(s) = \sum_a \pi(a, s) \sum_{s', r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma \sum_{a'} p(a'|s') q_\pi(s', a')]$$

$$v_*(s) = \max_a \sum_{s', r} p(s', r|s, a)[r + \gamma v_*(s')]$$

$$q_*(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')]$$