Introduction to Reinforcement Learning

Jean Martinet, based on the course of Diane Lingrand

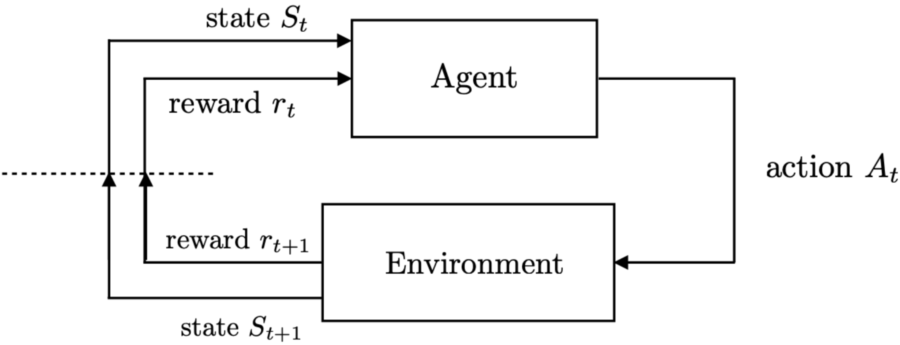
Polytech SI4 / EIT Digital DSC

2020 - 2021

Three main learning paradigms

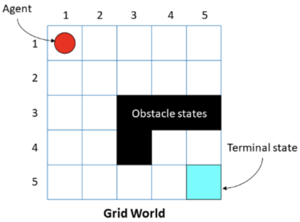
- **Supervised learning**
  - Learn a mapping between inputs and outputs
  - An oracle provides labelled examples of this mapping
- **Unsupervised learning**
  - Learn a structure in a data set (capture the distribution)
  - No oracle
- **Reinforcement Learning**
  - Learn to behave
  - Online learning
  - Sequential decision making under uncertainty, control

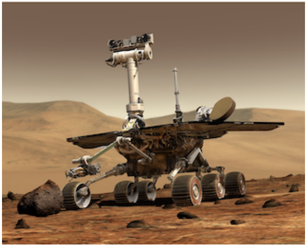
General problem



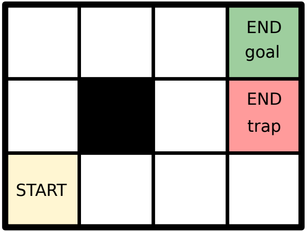
Examples

- **Artificial problems**
  - Mazes, grid worlds
  - Mountain car
  - Inverted Pendulum
  - Games : Backgammon, Chess, Atari, Go
- **Real world problems**
  - Man-Machine Interfaces
  - Data center cooling
  - Autonomous robotics





A classic toy example



- State
  - position
- Reward
  - +1 if end goal
  - -1 if end trap
  - -0.04 move cost in any case

- Transition rules
  - 4 directions : UP, DOWN, LEFT, RIGHT
  - probability of move success : 0.8
  - probability of failure, end up in lateral position : 0.1 for each
  - example : chosen action = UP
    - probability 0.8 to go UP
    - probability 0.1 to go LEFT
    - probability 0.1 to go RIGHT
  - external bouncing walls
- shortest path and reliability

About this class

- Second half of EIIN825 - ECUE IA
- Six sessions (lectures, labs)
- Objectives
  - Understand the key concepts of RL, distinguish from other AI / ML
  - Know if a problem can be formulated as a RL problem and how
  - Implement standard RL algorithms
- Link to other courses
  - SI4 CVML, ...
- Prerequisites : Python proficiency, basics in probability and statistics
- Organisation
  - Material on LMS https://lms.univ-cotedazur.fr/course/view.php?id=1300
  - Slack channel #si4-ia
- Evaluation – will give 50% of EIIN825 grade
  - Two assignments (individual / in groups) – 30% each
    - **First graded assignment TODAY !**
  - One final written exam on May 20 10.30am-12pm – 60%

Sequential decision making

- At each time step t, agent in state $s_t \in S$ executes action $a_t \in A$
- As a consequence, the agent reaches a new state s_{t+1} and receives from the environment a reward r_{t+1}
  - feedback that measures the success or failure of an agent's action
- The total reward (return, also called utility) at time step t is

$$G_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

- Time horizon can be finite / infinite / indefinite
- Rather use a *discounted return*

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

- Discount factor γ says how much you care of immediate/future
  - if $\gamma = 1$, get a reward on step 1000 is as good as on step 3
  - if $0 < \gamma \leq 1$, more important to get rewards sooner
  - if $\gamma = 0$, only care for immediate reward, ignore future (myopic)

Sequential decision making

- Credit assignment problem
  - Rewards can be extremely delayed ; how to select actions that lead to a certain outcome ?
- A policy $\pi(a|s)$ is a mapping from states to probabilities of selecting each possible action – optimal policy π^*
- The **state**-value function of s_t under π is the expected return

$$V^\pi(s_t) = \mathbb{E}_\pi[G_t|s_t] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t]$$

- The **action**-value function of taking a_t in s_t under π is

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|s_t, a_t] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t, a_t]$$

- Goal : select actions to maximise total expected future rewards
  - Requires to balance immediate and long term rewards
  - Requires a strategy – balance exploration vs exploitation

Markov assumption

Andrey Andreyevich Markov (1856 – 1922) was a Russian mathematician
"The future is independent of the past given the present"

$$p(s_{t+1}|s_t) = p(s_{t+1}|s_1, s_2, ..., s_t)$$

- only the present matters
- the state captures all relevant information from the past (if needed)
- stationary (rules do not change)

Markov Decision Process (MDP)

A Process is a Markov Process if it satisfies the Markovian property
A Markov Reward Process is a MP with a reward at each state
A Markov Decision Process is a MRP with decisions

- Formal description of an environment for decision making / RL
- Tuple $\{S, A, P, R, \gamma\}$
  - **States :** s_t
  - **Action :** a_t
  - **Dynamics model (transitions) :** $P(s_t, a_t, s_{t+1}) \sim p(s_{t+1}|s_t, a_t)$
  - **Reward model :** $R(s_t)$ immediate reward
  - **Discount factor :** γ

How to learn a policy ?

- Brute force ?
  - evaluate all policies and return the best one : π^*
  - check your understanding : how many different policies ?
- Dynamic programming – when you know P and R
  - Policy iteration (evaluation+improvement), value iteration
- Monte Carlo methods
- Temporal Difference

"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning."
Sutton and Barto, 2018.

Temporal difference

- Init : agent assumes that the action value is 0 all (s_t, a_t) pairs
- Iterate : update the estimate using the observed difference between expected and actual (current) values
- Remember the state-value function $V^\pi(s_t) = \mathbb{E}_\pi[G_t|s_t]$ with
  - $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots$
  - $G_t = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+3} \dots)$
  - $G_t = r_{t+1} + \gamma G_{t+1}$
- Then

$$V^\pi(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1}|s_t]$$

$$V^\pi(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(s_{t+1})|s_t]$$

- The TD error δ_t is the difference between the *estimated* value of s_t and the *better estimate* $r_{t+1} + \gamma V^\pi(s_{t+1})$

$$\delta_t = r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

Q-learning

- On-policy versus off-policy methods
  - On-policy : evaluated and used to make decisions are the same
  - Off-policy : evaluated and used to make decisions are different
- Q-learning is an off-policy TD algorithm

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

- Iteratively updates action-value using δ_t
- α is the learning rate

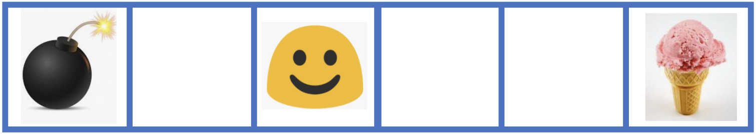
Q-learning algorithm

- Build a Q-table of size $|S| \times |A|$, initial values are 0s
- Iterate episodes (a few hundreds)
  - Iterate steps in each episode
    - Select the best action a in state s, use the reward to update Q
    - Episode terminates when agent reaches a terminal state (or max iteration)

ϵ-greedy

- Used for choosing an action
  - Trade-off between exploration and exploitation using ϵ value
- ϵ starts at 1 (only exploration) then decreases (more exploitation)
- Choose a random number r between 0 and 1 (uniform distribution) :
  - if $r < \epsilon$: choose a random action (exploration)
  - if $r \geq \epsilon$: choose the best action = maximising Q value (exploitation)

Today's lab



- Consider a 1D grid with :
  - one goal location (positive reward, e.g. $+1$)
  - one trap location (negative reward, e.g. -1)
  - a fixed move cost (e.g. -0.01)
  - deterministic actions (probability to go left when trying left is 1)
- Implement Q-learning from the equation
- Run your algorithm to determine the best policy
- Optional : extend to the 2D grid described in the classical toy example
- Notes :
  - Use Python and numpy only (gym next week)
  - Submit your solution at the end of the session, deadline 12.15pm.