Multi Agent Reinforcement Learning (MARL)

Adam Lagssaibi, Dabier Edgard, Nolan Sisouphanthong
Supervisor: Alexandre Reiffers-Masson

Project on recent advance in machine learning
IMT Atlantique

March 28, 2025

Overview

1. Introduction to Reinforcement Learning (RL)

2. Simple Multi-agent Reinforcement Learning (MARL) setup

3. Conclusion and perspectives

Section 1

Introduction to Reinforcement Learning (RL)

What is Reinforcement Learning?

RL is part of Machine Learning, but is neither supervised or unsupervised learning.

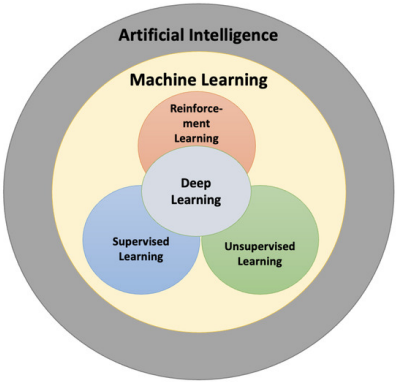
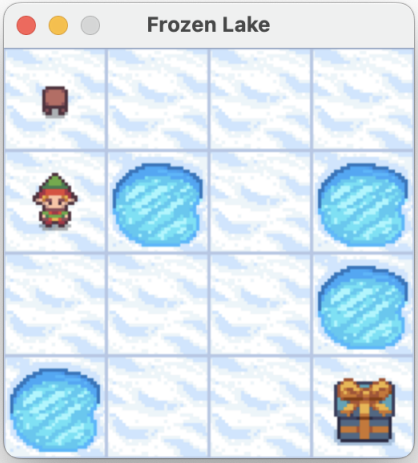


Figure: Reinforcement Learning inside Machine Learning[1]

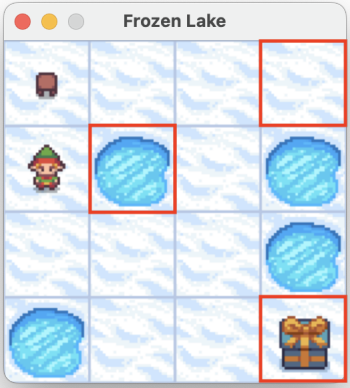
[1] Fahad Mon et al., *Reinforcement Learning in Education: A Literature Review*, 2023. https://www.mdpi.com/2227-9709/10/3/74

The FrozenLake example



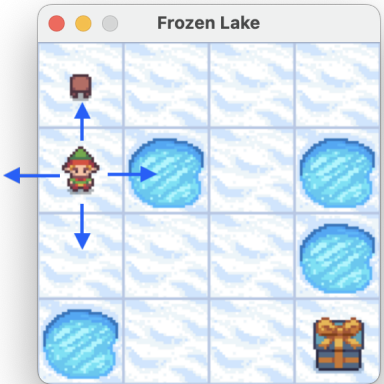
The FrozenLake example

The **environment** of the game is made of ice path, holes, and a goal to be reached by the agent.



The FrozenLake example

The agent can move around over time in the environment, we call this its **actions a**, such that he is in a **state s** at time **t**.



The FrozenLake example

Depending on the agent's actions, he can end up falling in a hole or reaching the goal.

Figure: Fallen Agent

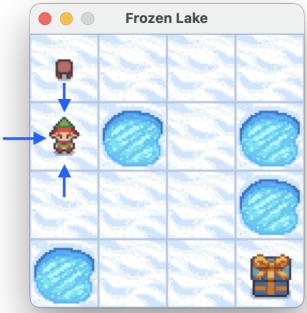
Figure: Goal Agent

Markovianity property

Key notion : **Markovianity** of the action chain

The probability of the agent being in a new state and receiving some reward only depends on its previous state and what action it takes, but not all the previous actions.

$$P(s^{t+1}|s^t, a^t, s^{t-1}, a^{t-1}, ..., s^0, a^0) = P(s^{t+1}|s^t, a^t)$$



The agent has to **learn** what actions to take when at a certain state.

This is called its **policy, π** .

$$\pi : S \rightarrow A$$

What is a good action?

We define a **reward function r** to indicate the agent if his action has a positive or negative impact on the objective.

Figure: Reward = **-1**

Figure: Reward = **+1**

Episodes and total return

Ending a run

We have to set limits so the agent doesn't run forever. We define

- final states: if the agent reaches them, we finish
- maximum time steps: the amount of authorized steps for the agent

Once a final state is reached or the agent exceeds the maximum time step, the game is over. We call this entire process an **episode**.

At the end of an episode, we cumulate all the agent's rewards in the **total return**.
$r = \sum_{i=0}^{T} r^i$

Summary

- **Actions:** The agent can take actions: a
- **Reward:** The agent gets rewards depending on its action: r
- **Objective - episodes:** When reaching the goal, or exceeding the maximum time step, we end the episode
- **policy:** The agent learns a policy π telling him what action a to take on any given state s
- **Total return:** At the end of an episode, we add up all the rewards in the total reward: $r = \sum_{i=0}^{T} r^i$

Discount factor

Reward design

The reward choice shapes what the agent will learn

- Direct cumulative reward for long term objectives
- Discounted reward to favor reaching the objective faster (greedy action taking):

We introduce the **discount factor** $\gamma \in [0, 1]$ such that we favor the first moves

$$r = \sum_{i=0}^{T} \gamma^i * r^i \rightarrow \text{the agent learns to reach the goal faster}$$

Dealing with Uncertainty in Games

What is Uncertainty?

Recall the markovian property: $P(s^{t+1}|s^t, a^t, s^{t-1}, a^{t-1}, ..., s^0, a^0) = P(s^{t+1}|s^t, a^t)$

In some games, things don't always go as planned.

- In a **slippery game**, the agent might slip and not move where it wants to.
- Some plans are riskier than others.

We need to find the best plan that gets the most rewards, even with uncertainty.

To pick the best plan, we test it many times and calculate the average reward:

$$\text{Average Reward} = \mathrm{E}(r) = \mathrm{E}_\pi \left[\sum_{i=0}^{T} \gamma^i \times r^i \right]$$

Value Functions

What Are Value Functions?

Value functions help the agent know how good a situation is in the game $\rightarrow$ learn the optimal policy.

- **State-Value Function** $V^{\pi}(s)$: How much reward the agent expects to get starting from a spot s (a state) if it follows a plan π.
- **Action-Value Function** $Q^{\pi}(s, a)$: How much reward the agent expects to get if it takes a specific move a in spot s, then follows plan π.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} R_{t+1} \mid S_0 = s\right]$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} R_{t+1} \mid S_0 = s, A_0 = a\right]$$

Using these functions to find the best plan $\pi^{\star}$ is called **value-based learning**.

Common RL Algorithms

- **Value Iteration**
- **Q-Learning**

Value Iteration — Understanding the Foundations

What is Value Iteration?

- A method to find the best value function $V^*(s)$.
- Shows how good it is to be in a state s if you act perfectly from there.

$$V_{k+1}(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V_k(s') \right]$$

- $P(s'|s, a)$: Chance of moving to state s' from state s by doing action a.
- $R(s, a, s')$: Reward you get for that move.

Key Properties

- Needs to know the game rules (P and R).
- Keeps updating V_k until it stops changing.
- Gives the best value function, which helps find the **best plan** $\pi^\star$.

Complexity

Grows with $card(S) \times card(A)$

Q-Learning — Learning by Interaction

What is Q-Learning?

- Learns the best action-value function $Q^*(s, a)$ by trying things out.

$$Q(s, a) \leftarrow Q(s, a) + \alpha\Big(r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)\Big)$$

- α: How fast the agent learns.
- r: Reward after doing action a in state s.
- s': The next state after the move.
- $\max_{a'} Q(s', a')$: Best estimated value for the next action.

Policy Learned

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

Common RL Algorithms

Value Iteration

- Updates the value of each state to find the best plan.
- Uses this rule to improve the value over time

Q-Learning

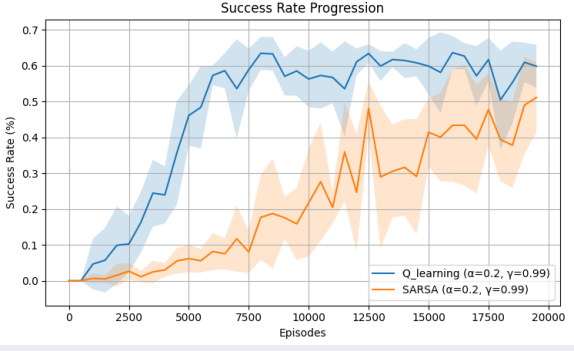
- Learns the value of taking an action in a state, called $Q(s, a)$.
- Updates Q using this rule after each move

Comparing learning algorithms

→ How do we compare learning methods?
→ What makes a learning algorithm good?

Learning curves

A good learning algorithm **converges** toward an **optimal** policy, and if possible, it converges **fast**.



Section 2

Simple Multi-agent Reinforcement Learning (MARL) setup

Introducing Multi-Agent RL (MARL)

Game Types:

Cooperative (shared reward), competitive, or mixed

Max reward:

Agents arrive **together**: *favors cooperation*

How to handle multi agents?

Value iteration

We can simply apply the same value iteration as in single agent setup to learn the **optimal** policy.

Reducing to single agent

Naive and intuitive approach: reduce to a single agent approach, and for that we have two options:

- Centralized Q-Learning (CQL): Train a single **central policy** receiving all agents' observations and selects actions for each agents.
- Independant Q-Learning (IQL): Every agent learns its **own policy**, based on its observations, state, actions and history, ignoring the other agents.

Value iteration

Pseudo code

Initialize $V_i^{\pi}(s)$ for each agent i and each state s (e.g., $V_i^{\pi}(s) = 0 \ \forall \ s, i$)
Repeat until convergence:
for *each agent i and each state s* **do**

$$V_i(s) = \max_{a_i} \sum P(s'|s, a_1, a_2, ..., a_n) * [R(s, a_1, a_2, ..., a_n, s') + \gamma * V_i(s')]$$

end
return $\pi_i^{\star}(s) = arg \max_{a} P(s'|s, a_1, a_2, ..., a_n) * [R_i(s, a_1, a_2, ..., a_n, s') + \gamma * V_i(s')]$

limitations

Double python for loop: (agents + states) $\rightarrow$ can become very heavy. **For example**, a 5×5 FrozenLake grid (25 positions) with 3 players, with 4 actions each $= (25 \times 4)^3 = 10e^6$ iterations, repeated n times until convergence!

Central Q-Learning

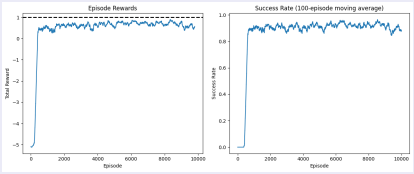
We **aggregate** all agents' states, observations and actions and we want to learn a central policy selecting every action in the **joint-action space** $A = A_0 \times A_1 \times ... \times A_N$ for N agents.

Limitations

Curse of dimensionality: the size of the joint action space grows exponentially with **n** (the number of agents): $A = A_i^n$

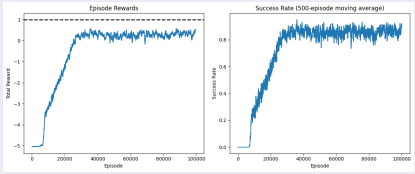
Central Q-Learning - curse of dimensionality

2 agents - 4×4 grid



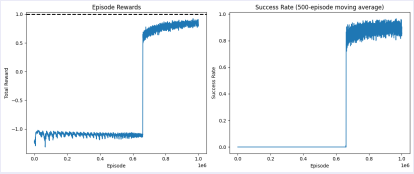
10e4 episodes, converged in **0.9s**

2 agents - 8×8 grid



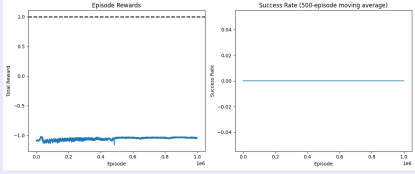
$10e^5$ episodes, converged in **21.9s**

4 agents - 4×4 grid



$10e^6$ episodes, converged in **2mn5s**

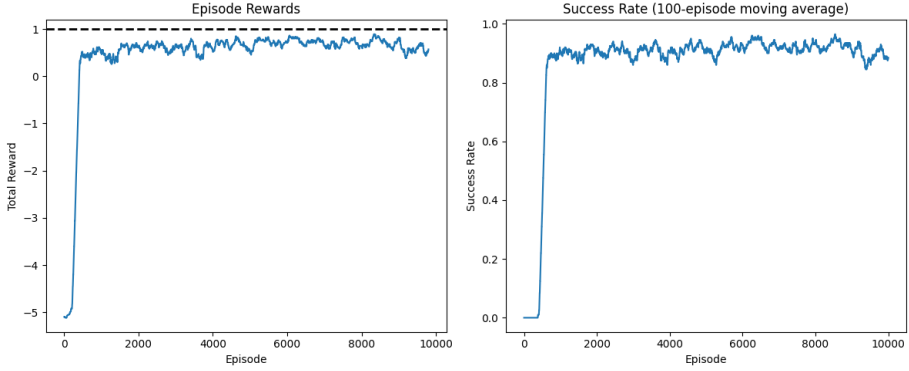
4 agents - 6×6 grid



lightred

$10e^6$ episodes, did not converge after **2mn26s**

Central Q-Learning - curse of dimensionality

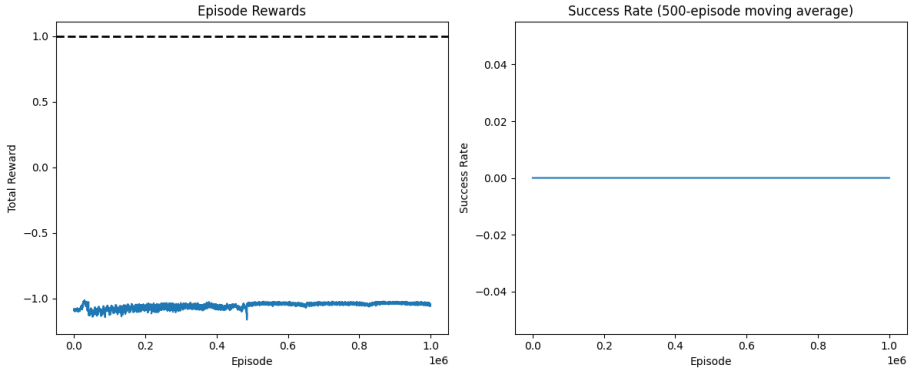


2 agents - 4×4 grid

n_episodes $= 10e^4$

converged in 0.9s

Central Q-Learning - curse of dimensionality



4 agents - 6×6 grid

n_episodes $= 10e^6$

did not converge after 2mn26s

A new setup

Figure: Reward = **0**

Figure: Reward = **+200**

Figure: Reward = **+400**

Four agents in a FrozenLake setup, each trying to reach a different goal and avoid **collisions** between agents to maximize their reward.

Independent Q-Learning

Key Idea

Each agent learns its policy or value function *independently / as if it were alone.*

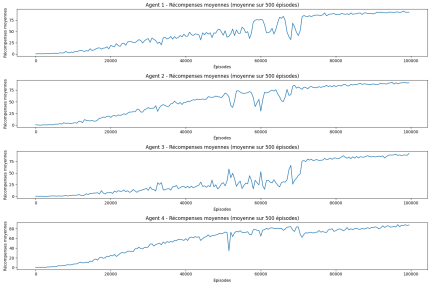
Each agent assumes the other agents are part of the **environment.**

Total Action space $\boldsymbol{A} = \sum_{i=1}^{\boldsymbol{N}} \boldsymbol{A_i} \neq \prod_{i=1}^{N} A_i$ for Central Q-Learning.

Limitations

- Environment appears **non-stationary** to each agent (others are learning/changing).
- Can lead to instability or slower convergence.

[video]



figureFluctuating reward due to non-stationarity.

Independent Q-Learning

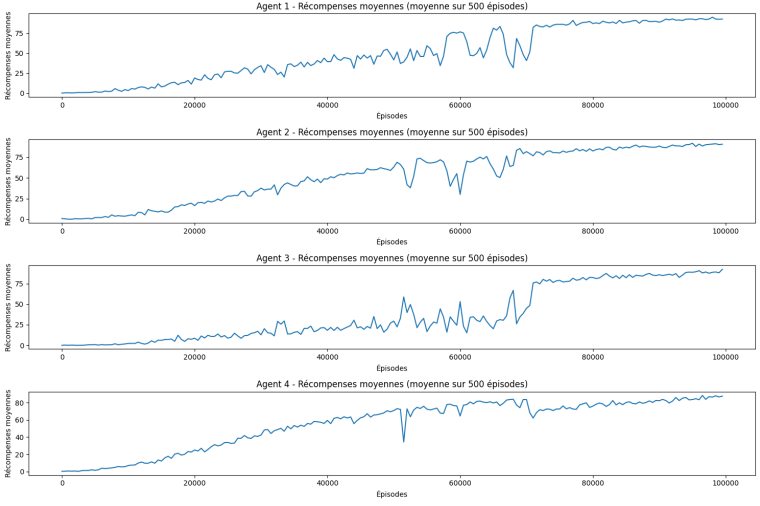
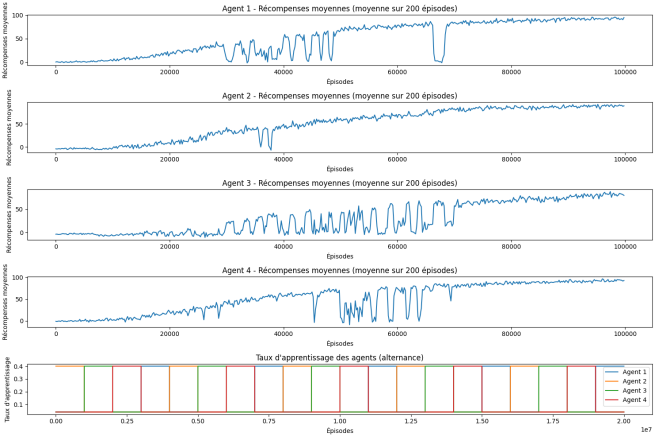


Figure: Fluctuating reward due to non-stationarity.

Alternating Independent Q-Learning

Key Idea

Each agent learns *independently / as if it were alone*, but takes turns learning. This helps reduce non-stationarity by stabilizing each agent's learning process.



Reward of agents with alternating learning rates.

[video]

Alternating Independent Q-Learning

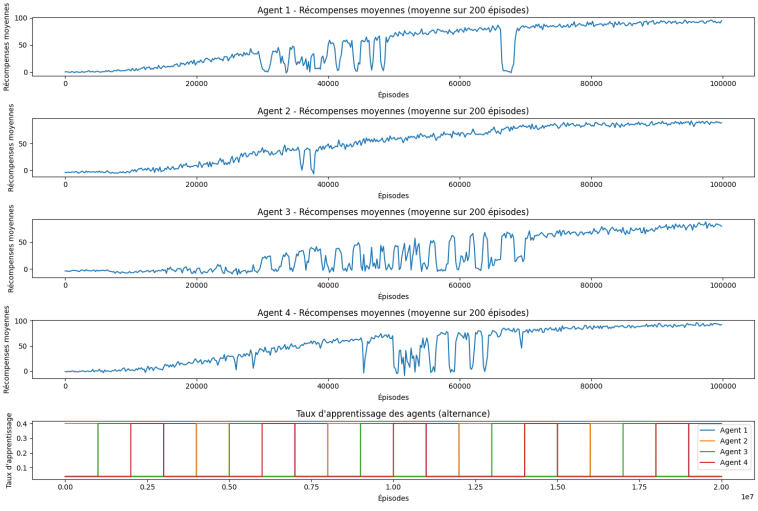
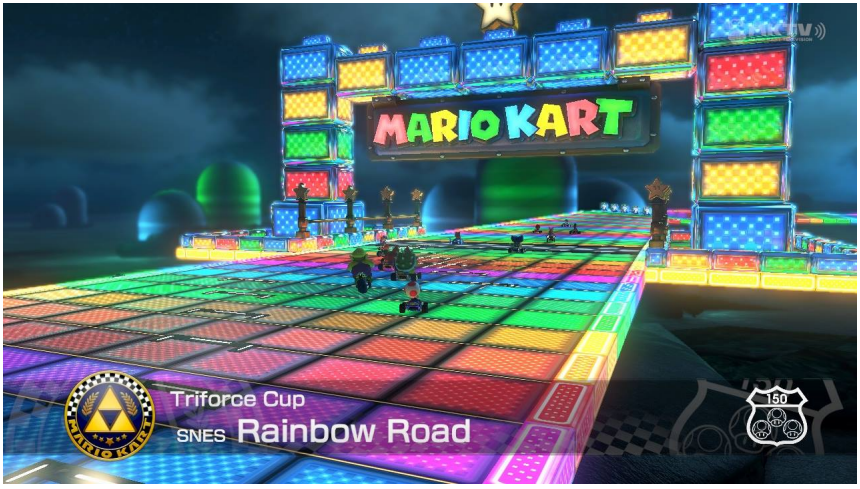


Figure: Alternating Q learning reward and alternating learning rates

Section 3

Conclusion and perspectives

What about more complex games?



Classic Q-Learning methods will not be sufficient $\rightarrow$ use of **Deep Learning** through Deep Q-Learning for example.

Conclusion

Reinforcement Learning (RL)

Agents learn by interacting with the environment.

- The agent learns a policy by optimizing one of the **value function** $V^*(s)$ *or* $Q^*(s, a)$

Multi-Agent RL (MARL)

Agents must also adapt to each other, increasing complexity.

- **Central Q learning** Accurate but not scalable.
- **Independent Q-Learning (IQL)** Scalable, but suffers from non-stationarity.
- **Alternating learning strategies** Helps improve stability and convergence.
- **Deep learning + Q-function** Enables learning in more complex environments.


Come see our work! https://github.com/edabier/MARL-project