MARL (2/2)

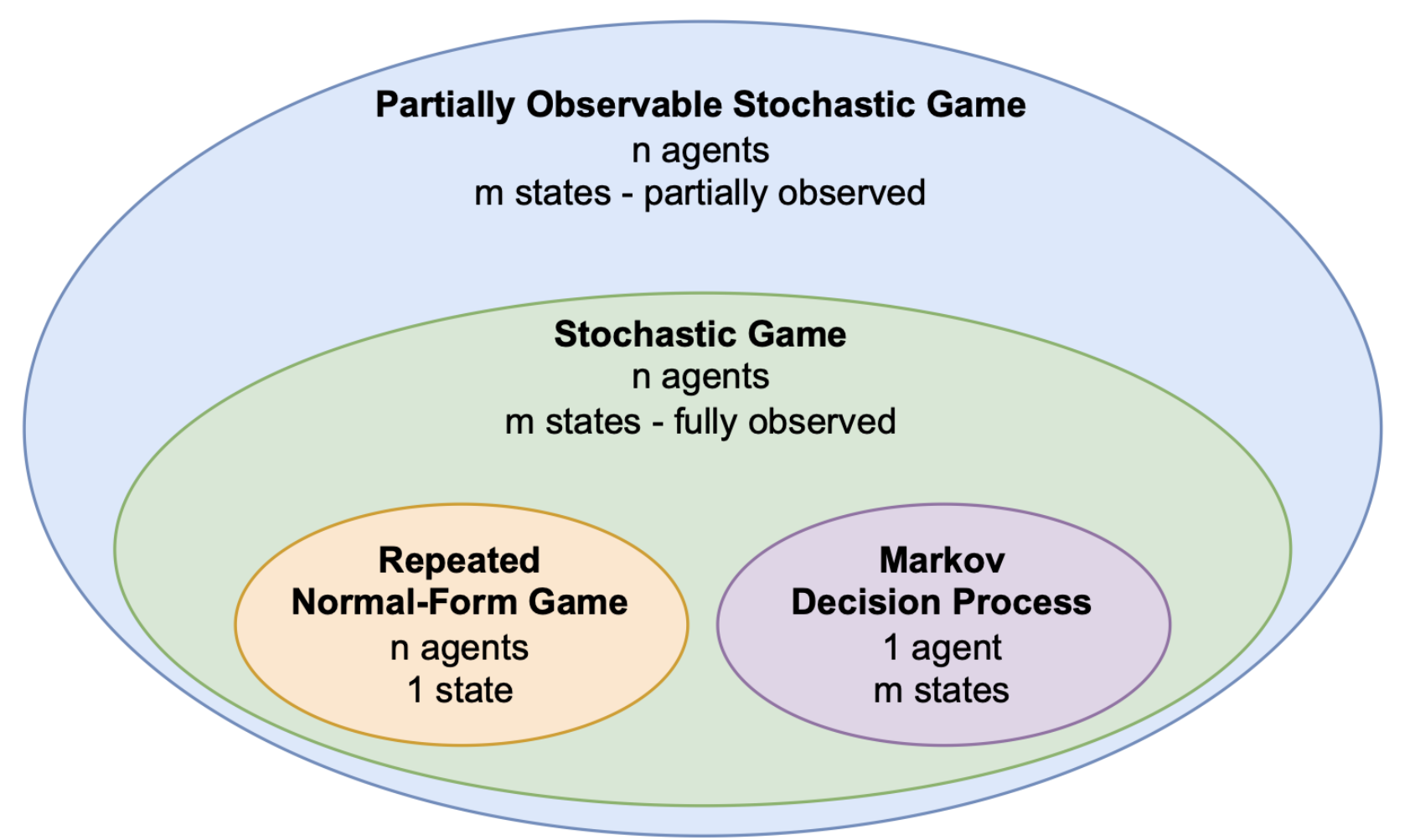
Multi-agents reinforcement learning

MARL

Rappels

MARL Problem = Game Model + Solution Concept

How to model a game? (reminder of last lecture)



Source. Multi-Agent Reinforcement Learning: Foundations and Modern Approaches. Albrecht, Christianos, Schäfer.

What does it mean to solve the game?

What does it mean to solve the game? (MDP)

- **MDP:** Find a policy $\pi : S \to A$ that maximizes the expected sum of discounted rewards

$$\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

What does it mean to solve the game? (MDP)

- **MDP:** Find a policy $\pi : S \rightarrow A$ that maximizes the expected sum of discounted rewards

$$\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

What if more than one agent?

- If cooperative?
- If competitive?
- If mixed?

Minimax

- **Setting:** 2 agents, zero-sum game. One agent tries to maximize the reward, the other tries to minimize it.
  - Examples: Rock-paper-scissors, Chess, Go, ...
- **Solution:** We are looking for a joint policy $\pi = (\pi_1, \pi_2)$. It is a minimax solution if

$$\begin{aligned} U_i(\pi) &= \max_{\pi_1'} \min_{\pi_2'} U_i(\pi_1', \pi_2') \\ &= \min_{\pi_2'} \max_{\pi_1'} U_i(\pi_1', \pi_2') \\ &= -U_j(\pi) \end{aligned}$$

Example

Rock-paper-scissors

Nash Equilibrium

- **Setting:** n agents, general-sum game.
- **Solution:** We are looking for a joint policy $\pi = (\pi_1, \ldots, \pi_n)$. It is a Nash equilibrium if

$$\forall i, \forall \pi_i', U_i(\pi_i', \pi_{-i}) \leq U_i(\pi)$$

Example

- Prisoner's dilemma
- Nuclear weapons during the Cold War

In practice, ϵ-Nash Equilibrium

Practical issues:

- action probabilities may be irrational numbers (hard to represent)
- reaching a Nash equilibrium may be computationally intractable

In practice, ϵ-Nash Equilibrium

Practical issues:

- action probabilities may be irrational numbers (hard to represent)
- reaching a Nash equilibrium may be computationally intractable

Solution

- ϵ-Nash equilibrium:

$$\forall i, \forall \pi_i', U_i(\pi_i', \pi_{-i}) \leq U_i(\pi) + \epsilon$$

Pareto Optimality

- **Setting:** n agents, general-sum game.
- **Solution:** We are looking for a joint policy $\pi = (\pi_1, \ldots, \pi_n)$.

  It is Pareto optimal if
- There is no other joint policy π' such that
  - $\forall i, U_i(\pi') \geq U_i(\pi)$
  - $\exists i, U_i(\pi') > U_i(\pi)$

Example

Prisoner's dilemma solution?

How to find a solution?

First idea, reduce to a single-agent problem and use RL

Central learning

Idea: train a single policy π_c that receives the local observations of all agents and select an action for each agent.

Central learning

Idea: train a single policy π_c that receives the local observations of all agents and select an action for each agent.

Challenges: Transform a joint reward $r_1, \ldots, r_n$ into a single reward r? (for which game is it easy?)

Algorithm 4 Central Q-learning (CQL) for stochastic games

1: Initialize: $Q(s,a) = 0$ for all $s \in S$ and $a \in A = A_1 \times \ldots \times A_n$
2: Repeat for every episode:
3: **for** $t = 0, 1, 2, \ldots$ **do**
4: Observe current state s^t
5: With probability ϵ: choose random joint action $a^t \in A$
6: Otherwise: choose joint action $a^t \in \arg\max_a Q(s^t, a)$
7: Apply joint action a^t, observe rewards $r_1^t, \ldots, r_n^t$ and next state s^{t+1}
8: Transform $r_1^t, \ldots, r_n^t$ into scalar reward r^t
9: $Q(s^t, a^t) \leftarrow Q(s^t, a^t) + \alpha \left[r^t + \gamma \max_{a'} Q(s^{t+1}, a') - Q(s^t, a^t) \right]$

Source. Multi-Agent Reinforcement Learning: Foundations and Modern Approaches. Albrecht, Christianos, Schäfer.

Problem with central learning

- **Curse of dimensionality:** the state space grows exponentially with the number of agents.
  - n agents with m states each: m^n states.
  - $n = 10$ agents with $m = 5x5$ states each: $25^{10} \approx 9.53 \times 10^{13}$ states.
- **Decentralized execution:** agents need to act independently and may not have access to all observations (i.e., the complete state s).

Independent learning (IL)

- train a policy π_i for each agent i, based on its own observations.
- Agents do not observe the actions of other agents (new env)
- Agents are trained independently to maximize their own reward.

Algorithm 5 Independent Q-learning (IQL) for stochastic games

// Algorithm controls agent i

Initialize: $Q_i(s, a_i) = 0$ for all $s \in S, a_i \in A_i$

Repeat for every episode:

for $t = 0, 1, 2, \ldots$ **do**

  Observe current state s^t

  With probability ϵ: choose random action $a_i^t \in A_i$

  Otherwise: choose action $a_i^t \in \arg\max_{a_i} Q_i(s^t, a_i)$

  (meanwhile, other agents $j \neq i$ choose their actions a_j^t)

  Observe own reward r_i^t and next state s^{t+1}

  $Q_i(s^t, a_i^t) \leftarrow Q_i(s^t, a_i^t) + \alpha \left[r_i^t + \gamma \max_{a_i'} Q_i(s^{t+1}, a_i') - Q_i(s^t, a_i^t) \right]$

Source. Multi-Agent Reinforcement Learning: Foundations and Modern Approaches. Albrecht, Christianos, Schäfer.

Pros and cons of IL

- Avoids the curse of dimensionality.
  - n training problems of size m instead of one problem of size m^n.
- Agents can act independently after training. So it is easy to deploy on a real system.
- Downside: non-stationarity of the environment.
  - The environment is changing because the other agents are learning too.
  - May produce unstable learning and may not converge to a solution.

Mixed setting: modelization of a game with two teams

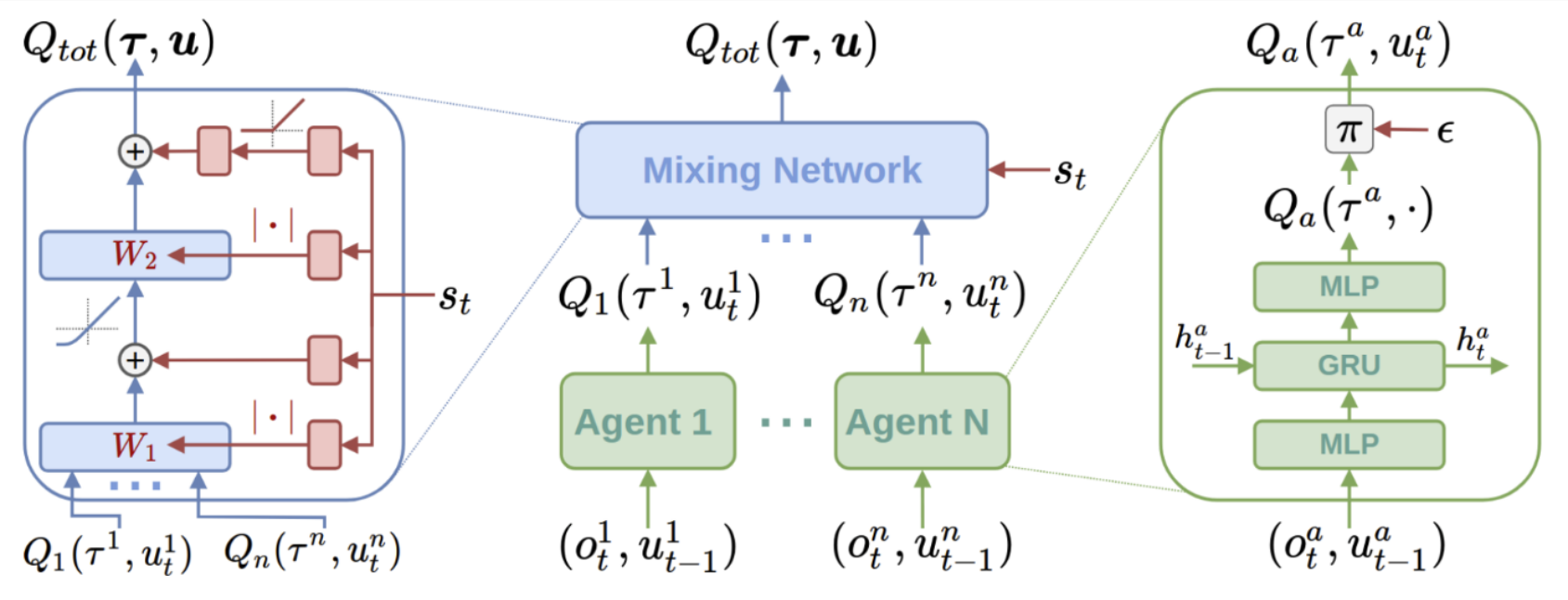
- Competitive + cooperative
- Capture the flag, soccer, ...
- **Idea**. Reduction to two cooperative games.

whiteboard

Use Deep Reinforcement Learning (function approximations)

Another paradigm: CTDE (centralized training, decentralized execution).

QMIX.

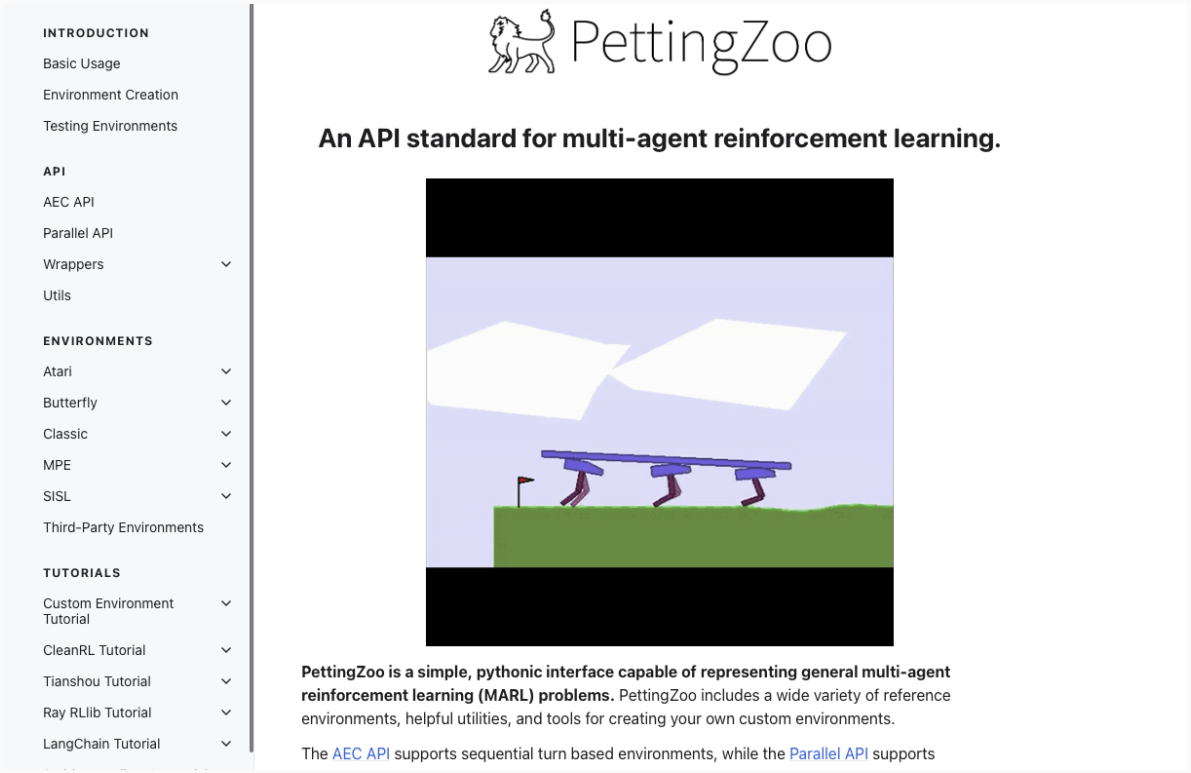


- o_t^a: This refers to the observation of agent a at time t. Each agent in a multi-agent system receives its own local observation o_t^a of the environment, which may not fully capture the global state s_t.
- u_t^a: This represents the action taken by agent a at time t. The joint action of all agents at time t is denoted as $\mathbf{u}_t = (u_t^1, u_t^2, \ldots, u_t^n)$, where n is the number of agents.
- τ^a: This refers to the action-observation history (or trajectory) of agent a. Specifically, $\tau^a = (o_1^a, u_1^a, o_2^a, u_2^a, \ldots, o_t^a)$ represents all observations and actions for agent a up to time t. It's used as the input to the agent's Q-function $Q_a(\tau^a, u^a)$, which estimates the expected cumulative reward for agent a given its local history and the current action u^a.

Practical session

PettingZoo

- A Python library for multi-agent reinforcement learning.
- Provides a collection of environments for multi-agent reinforcement learning.
- Can be combined with `stable-baselines3` for training.



Environment like PettingZoo.

```
from pettingzoo.butterfly import pistonball_v6
parallel_env = pistonball_v6.parallel_env(render_mode="human")
observations, infos = parallel_env.reset(seed=42)

while parallel_env.agents:
    # this is where you would insert your policy
    actions = {agent: parallel_env.action_space(agent).sample() for agent in parallel_env.agents}

    observations, rewards, terminations, truncations, infos = parallel_env.step(actions)
parallel_env.close()
```

Custom environment

- Use `custom_env.py` as the starting point from the environment.
  - `max_steps`: maximum number of steps in an episode.
  - `n_agents` on a grid. There is a target cell. If an agent reaches the target, the team receives a reward. The goal is to reach as many targets as possible during the episode.

```python
def observe(self, agent):
        """Returns the observation of an agent, as a dictionary."""
        position = self.agent_positions[agent]

        return {
            "optimal_action_to_target": self._get_optimal_action(position, self.target),
        }
```

```
Step 300/300
. . . . 0
. . . . .
1 . . . .
. 2 . . .
. . . . T

Cumulative Rewards: {'agent_0': 5.080000000000017, 'agent_1': 2.0500000000000096, 'agent_2': 5.080000000000365}
Cumulative common reward: 12.210000000000063
```

What you need to do.

1. Implement a handcrafted policy for each agent (in `handcraft.py`), `class OptimalIndependentAgent`, so that each agent tries to reach the target as quickly as possible.
   1.1 What is the baseline performance with 3 agents, over 200 episode of length 1000?
2. Implement **Independent Learning** (IL) for each agent (in `IL.py`).
   2.1 What is the performance?
3. Add further data in the observation, to enable agents to cooperate and not act independently.
   3.1 How does it affect the performance?
4. Explain the results by analyzing the policies learned by the agents.
5. Implement **Centralized Training** (CT) with a single policy for all agents.
   5.1 What is the performance?