Lenient Multi-Agent Policy Gradients

Jacopo Castellini

17th June, 2026

dmml group

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
University of Applied Sciences and Arts
Western Switzerland

Multi-Agent Systems

We can formalize a multi-agent problem as a Dec-POMDP $\mathcal{M} = \langle A, S, \{U^a\}_{a \in A}, T, R, \{Z^a\}_{a \in A}, O, \gamma \rangle$:

- A is the set of agents,
- S is the set of states,
- U^a is the set of local actions for agent a, with $\mathbf{u} = \{u^a\}_{a \in A}$ a joint action,
- $T : S \times \mathbf{U} \to \Delta(S)$ are the transition dynamics,
- $R : S \times \mathbf{U} \to \mathbb{R}$ is the reward function,
- Z^a is the set of local observations for agent a,
- $O : S \to \Delta(\mathbf{Z})$ is the observation function,
- γ is a discount factor.

The objective is to find a joint policy $\boldsymbol{\pi} = \{\pi^a\}_{a \in A}$ that maximizes $V^{\boldsymbol{\pi}}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mathbf{u}_t) | s_0 = s\right] \; \forall s \in S$.

Multi-Agent Policy Gradients

Multi-agent policy gradients updates the parameters θ^a of each agent policy π^a individually by following:

$$\nabla_{\theta^a} J(\boldsymbol{\pi}_\theta) = \mathbb{E}\Big[\nabla_{\theta^a} \log \pi_{\theta^a}(u^a|\tau_t^a)\, Q^{\boldsymbol{\pi}_\theta}(s, \mathbf{u})\Big],$$

where $\tau_t^a = \langle z_0^a, u_0^a, \ldots, z_t^a \rangle$ and $Q^{\boldsymbol{\pi}_\theta}(s, \mathbf{u})$ is estimated with a critic.

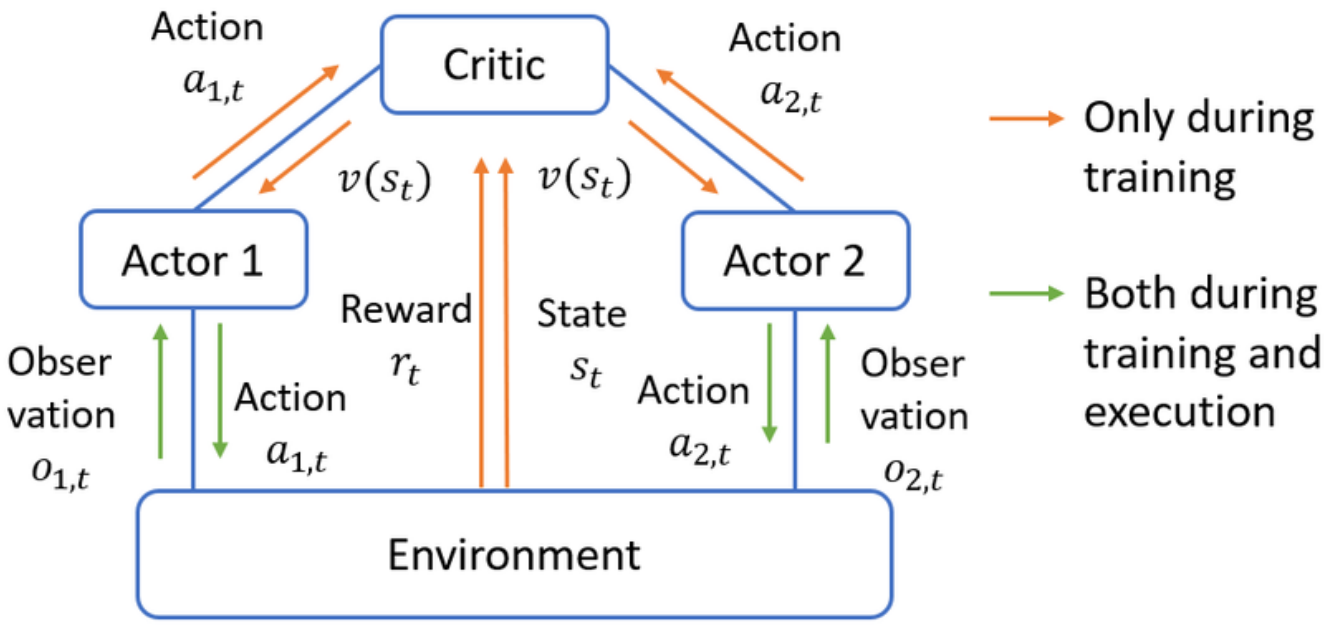
Policy optimization instead finds the optimizer of some constrained problem. PPO is a simple way to enforce this:

$$J(\pi_\theta) = \mathbb{E}\Big[min(r(\theta^a) A^{\boldsymbol{\pi}_\theta}(s, \mathbf{u}), \text{clip}(r(\theta^a), 1 \pm \epsilon) A^{\boldsymbol{\pi}_\theta}(s, \mathbf{u}))\Big],$$

where $r(\theta^a) = \frac{\pi_{\theta^a}(u^a|\tau_t^a)}{\pi_{\theta^a_{\text{old}}}(u^a|\tau_t^a)}$ and $A^{\boldsymbol{\pi}_\theta}(s, \mathbf{u}) = Q^{\boldsymbol{\pi}_\theta}(s, \mathbf{u}) - V^{\boldsymbol{\pi}_\theta}(s)$.

Centralized Training-Decentralized Execution

In CTDE, agents can access all information (e.g., states, joint actions) during training (i.e., to learn the critic), but are restricted to local ones (local observations and actions) at execution time.



Relative Overgeneralization

Climb Game:

Agent 1 \ Agent 2	A^2	B^2	C^2
A^1	8	−12	−12
B^1	−12	0	0
C^1	−12	0	0

Relative Overgeneralization:

- Optimal solution: (A^1, A^2)
- Decentralized solutions: (B^1, B^2), (C^1, B^2), (B^1, C^2), (C^1, C^2)

Overcoming RO: Optimism

Optimism: bad rewards are incurred only because of unlucky actions (e.g., mis-coordination):

- *Hysteresis*: use a second learning rate $\beta < \eta$ for TD-errors that reduce the *Q*-values,
- *Leniency*: forgive others if they do bad at the beginning, but gradually decrease this behaviour when state-action pairs are visited more often.

Note: Leniency seems to be more useful for problems with stochastic rewards, which are not the popular ones in MARL...

Overcoming RO: Optimism (cont'd)

In leniency, a factor $l(s, u^a) = 1 - e^{-K \cdot T(s,u^a)}$ is used, with the temperature $T(s, u^a)$ decreasing as $T_{t+1}(s, u^a) = \beta T_t(s, u^a)$, $0 \leq \beta \leq 1$ whenever (s, u^a) is visited.

We then update $Q^a(s, u^a)$ by TD-error δ as:

$$Q^a(s, u^a) = \begin{cases} Q^a(s, u^a) + \eta\delta & \text{if } \delta > 0 \text{ or } x \sim U(0, 1) > l(s, u^a), \\ Q^a(s, u^a) & \text{otherwise.} \end{cases}$$

Lenient DQN

Independent DQN with leniency, plus some adjustments for sequential problems:

- *Retroactive Temperature Delay Schedule*: to avoid becoming too conservative on initial states too early,
- *Autoencoder + Hashing*: for keeping the temperature table size manageable.

Problem: Lenient DQN are independent learners at the core and works in fully observable settings, not really suited for modern MARL...

Lenient Multi-Agent Policy Gradients

Lenient Multi-Agent Policy Gradients combine CTDE policy gradients with leniency to address partial observability in pathological problems.

LMAPG uses the centralized advantage $A^{\boldsymbol{\pi}_\theta}(s, \mathbf{u})$ in place of the TD-error to decide when to update the policy parameters θ^a:

$$\theta^a = \begin{cases} \theta^a + \eta \nabla_{\theta^a} J(\pi_\theta) & \text{if } A^{\boldsymbol{\pi}_\theta}(s, \mathbf{u}) > 0 \text{ or } x \sim U(0,1) > l(s, \mathbf{u}), \\ \theta^a & \text{otherwise,} \end{cases}$$

In practice, PPO-style clipping can be used as well.

Lenient Multi-Agent Policy Gradients (cont'd)

The leniency factor $l(s, \mathbf{u})$ and temperature $T(s, \mathbf{u})$ are computed and updated similarly to Lenient DQN:

- Using Retroactive TDS (i.e. higher $\beta_t = e^{\rho \cdot d^{(T-t)}}$ for early state-action pairs),
- Autoencoding + SimHash:

$$hash(s) = sign(A \cdot enc_{AE}(s)),$$

where $A \sim \mathcal{N}(0, 1)$ is chosen once at the beginning.

Note: The temperature $T(s, \mathbf{u})$ and leniency factor $l(s, \mathbf{u})$ are maintained for state-*joint* action pairs, following the CTDE paradigm.

Stochastic Rewards from PO

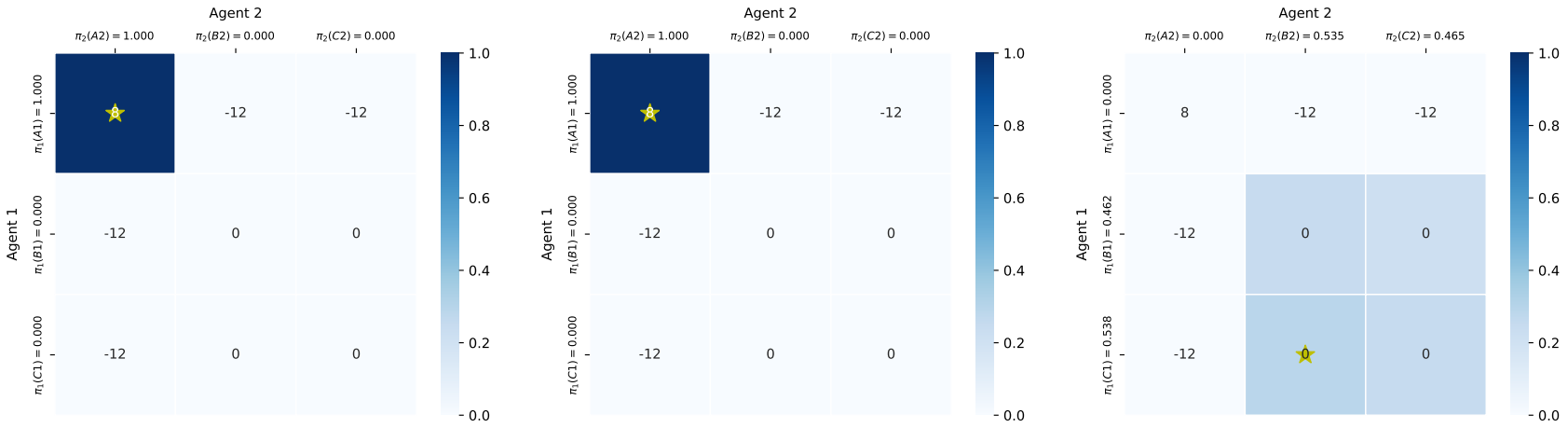
As discussed, leniency helps particularly with stochastic rewards.

In a Dec-POMDP, the reward process perceived by each agent policy π_{θ^a} is different from the true $R(s, \mathbf{u})$ and thus stochastic when only samples from the true reward are available:

$$R(\tau^a, u^a) = \mathbb{E}_{p(\tau_t^a|s), \boldsymbol{\pi}_{\theta-a}}\left[R(s, \mathbf{u})\right],$$

where $p(\tau_t^a|s)$ is the probability that agent a is in history τ_t^a when the state is s, and $\pi_{\theta-a}$ is the joint policy of all agents except a.

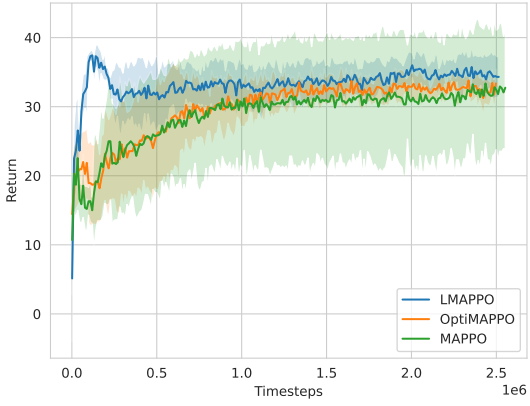
Experiments: Climb Game



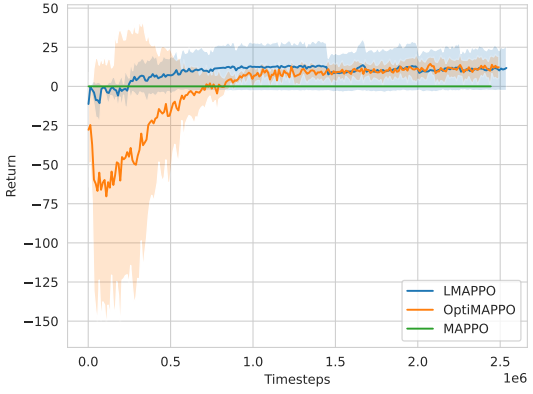
(a) LMAPPO **(b)** OptiMAPPO **(c)** MAPPO

As expected, both LMAPPO and OptiMAPPO (MAPPO with hysteresis) can learn the optimal joint policy, while vanilla MAPPO fails.

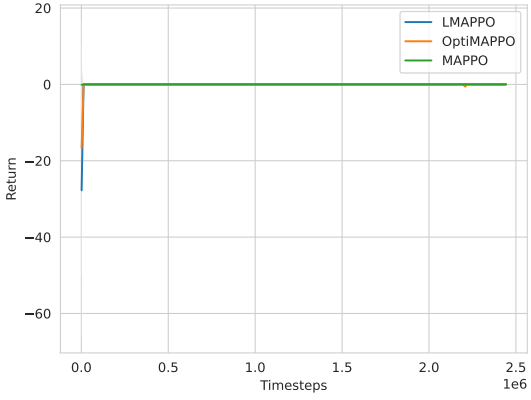
Experiments: Stag Hunt



(a) $p = 0$



(b) $p = -1$



(c) $p = -2$

LMAPPO performs on-par with OptiMAPPO (although faster), and the performances of both drastically reduces with an increased penalty p.

Experiments: Stag Hunt (cont'd)

Possible causes for this (which I have to investigate):

- Loosing leniency towards the others too quickly, and thus being influenced by bad transitions,
- Too small batches of samples (leniency discards bad samples, and thus we may end up not having enough/any to learn from),
- The information of the centralized critic and the decentralized policy are not well-aligned,
- Code bugs (always possible...),
- This is simply what it performs like, and I have to accept it :_)

Next Steps

1. The autoencoder may be scrambling the learning process, as too many things are learned together and with decoupled objectives,
2. The autoencoder may simply not be needed: it does not enforce any particular similarity constraint on the produced latent representations per-se (maybe something like VAE could...), and for the dimensionality reduction aspect the SimHash algorithm itself could handle it,
3. Maybe use some representation that can actually cluster states based on some meaningful similarity concept, such as their return. Could the hidden layer of the critic be a valid representation then?

Bonus: History-Based LMAPG

I also implemented a version using a recurrent critic $V_\omega(\tau_t)$, with the hashing computed using $enc_{AE}(h_{out})$ (i.e., the history encoding from the recurrent layer of the critic).

Note: Differently from the states s, the history encodings h_{out} changes throughout learning, thus making the autoencoder target to shift!

Thank you!

Questions?