Northeastern University

Masters Project Course

Multi-Agent Reinforcement Learning with Continuous Macro-Actions

Author:
Atharva Wandile
(002110295)

Advisor:
Prof. Christopher Amato
Enrico Marchesini

27th April 2023

Contents

1 Abstract

Hierarchical methods provide a useful structured framework to solve asynchronous reinforcement learning problems efficiently, they also have the added benefit of transferring knowledge between similar tasks. However most of these approaches have been developed for single-agent environments with discrete action spaces, this limits their potential in real-world settings where most problems require the use of multiple agents working together asynchronously and the action spaces are usually continuous. Hence this project aims to explore the performance of the continuous macro-action algorithms on standard multi-agent reinforcement learning environments that will be extended to the macro-actions framework. The results of this research will provide valuable insights into the capabilities of macro-action methods in multi-agent, continuous action space environments and inform future developments in the field.

Keywords: Reinforcement Learning, Multi-Agent Systems, Macro-Actions

2 Introduction

Reinforcement learning has recently shown a lot of promise in spaces like robotics, simulations, games, etc. using actor-critic methods. However, most of these methods assume synchronous primitive actions and single-agent settings. This does not scale well into the real world requiring asynchronous execution and temporally extended actions.

Due to the above issues, hierarchical methods were developed for better transfer between tasks and a more abstract representation of state and action spaces. Besides this, they also naturally represent complex tasks in an abstract and human-understandable manner. For example, waypoint navigation, and robotic tasks like grasping, placing etc. can be explained to a human rather than primitive actions like hundreds of fine motor controls, muscle contractions, sensors, etc.

The options framework is a well-known approach to adding temporal abstraction to an environment's actions. This is useful because it does not require prior knowledge of the environment and can learn both the action and the options from scratch without human input. However, it suffers from very high training variance and does not work well for multi-agent and partially observable environments. An alternative is to use macro-action-based frameworks which can be extended with Macro-Action Decentralized Partially Observable Markov Decision Process (MacDec-POMDP). The MacDec-POMDP is a general model for multi-agent problems with partial observability and different action durations. As a result, agents can start and end macro-actions at different time steps so decision-making can be asynchronous.

While there has been some work done in the macro-action space for MARL (Xiao et al., 2022). These focus on discrete action spaces. However, most systems in the real world require continuous actions so that agents can be controlled at a finer granularity (robotic controllers, automobile steering, waypoint navigation, etc.). This makes it necessary to develop and test methods using continuous macro-actions.

There are no in-built libraries or environments that use continuous macro-actions as input. Hence it makes it necessary to not only build these but also test the primitive algorithms to benchmark these environments. Then they can be used for testing macro-action algorithms.

3 Preliminaries

This project develops centralized training with decentralized execution (CTDE) methods for Macro-Action based MARL using the MacDec-POMDP framework. The next sections describe Dec-POMDPs and MacDec-POMDPs.

3.1 Multi Agent Reinforcement Learning

Multi-agent RL (MARL) involves a set of agents in a shared environment, which must learn to maximize their individual returns. This work focuses on cooperative settings, where agents share a joint return. Unique challenges arise in MARL due to agent interactions during learning. Multi-agent domains are non-stationary from agents' local perspectives, due to teammates' interactions with the environment. Agents, in particular, are susceptible to shadowed equilibria, where local observability and non-stationarity cause locally optimal actions to become a globally sub-optimal joint action.

This section introduces MARL under partial observability. We formalize single-task MARL using the Decentralized Partially Observable Markov Decision Process (Dec-POMDP) (Oliehoek and Amato, 2016), defined as $\langle I, S, A, T, R, \Omega, O, \gamma \rangle$, where I is a set of n agents, S is the state space, A = is the joint action space, and Ω is the joint observation space. Each agent i executes action $a^{(i)} \in A$ where joint action $a = \langle a^{(1)}, ..a^{(n)} \rangle$ causes environment state $s \in S$ to transition with probability $P(s'|s, a) = T(s', a, s)$. At each timestep agent receives observation $o^{(i)} \in \Omega$ with joint observation probability $P(o|s', a) = O(o, s', a)$ where $o = \langle o^{(1)}, ..o^{(n)} \rangle$

The agents receive a joint reward $r_t = R(s_t, a_t) \in R$ at each timestep t, the objective being to maximize the value (or expected return), $V = \mathbb{E}\left[\sum_{t=0}^{H} \gamma_t r_t\right]$.

3.2 Macro-Action based MARL

Dec-POMDPs with temporally extended actions that are based on the options framework (Bacon et al., 2016) are referred to MacDec-POMDPs (Amato et al., 2014) . Formally a MacDec-POMDP is represented as a tuple $\langle I, S, A, M, \zeta, \Omega, T, O, Z, R \rangle$, where I, S, A, Ω, O, R are the same as defined in Dec-POMDP; $M = \times_i M_i$ is set of joint macro-actions with M_i being the finite set of macro-actions for each agent i; $\zeta = \times_i \zeta_i$ is set of joint macro-observations with ζ_i being the finite macro-observation for each agent i. Each macro action is defined as a tuple $m = \langle \beta_m, I_m, \pi_m \rangle$ where stochastic termination condition $\beta_m : H_i^A \rightarrow [0, 1]$ and the initiation set $I_m \subset H_i^M$ of the corresponding macro-action m, depending on the agents primitive-action-observation history H_i^A and macro-action-observation history H_i^M ; $\pi_m : H_i^A \rightarrow A_i$ denotes low level policy to achieve macro-action m. Taking into account the stochastic termination of a macro-action the transition probability is written as $T(s', \overrightarrow{\tau}, s, \overrightarrow{m}) = P(s', \overrightarrow{\tau} | s, \overrightarrow{m})$ where $\overrightarrow{\tau}$ timestep at which any agent completes its current macro-action m. The objective is to find a joint high level policy that only picks at macro-action level $\Psi = \times_i \Psi_i$ such that value of Ψ from initial state s_0, $V_\Psi(s_{(0)}) = \mathbb{E}\left[\sum_{t=0}^{h-1} \gamma_t r(s_t, a_t) | s_{(0)}, \pi, \Psi\right]$

3.3 Previous Approaches

The use of macro-actions for learning asynchronously will require a different approach to maintain and update replay buffers. (Xiao et al., 2020) shows 2 approaches in centralized and decentralized policies. The decentralized one is called Macro-Action Concurrent Experience Reply Trajectories (Mac-CERTs). In this approach, each agent maintains its own macro-action and reward and updates are done only based on the individual agents' macro-action terminations. The centralized approach is called Macro-Action Joint Experience Replay Trajectories (Mac-JERTs). In this approach, a joint reward is collected at each time step and updates take place when any of the agents' macro-actions terminate. These approaches were tested on (a) Capture Target, a variant of an existing multi-agent-single-target (MAST) domain (b) Box Pushing, a benchmark Dec-POMDP domain and (c) Warehouse Tool Delivery Domain inspired by human-robot interaction. The results show superior performance to primitive action alternatives and the centralized version outperforms the decentralized one.

Actor-critic methods and variants (Mnih et al., 2016) are very well known and have shown state-of-the-art performance in most single agent tasks, however, they struggle to perform well in multi-agent asynchronous tasks. This paper Xiao et al. (2022) shows an approach to apply actor-critic methods to macro-action multi-agent domains in three training paradigms : decentralized, centralized and centralized training for decentralized execution (CTDE). The three approaches shown by the paper are:

- Macro-Action-Based Independent Actor-Critic (Mac-IAC)
- Macro-Action-Based Centralized Actor-Critic (Mac-CAC)
- Macro-Action-Based Independent Actor with Centralized Critic (Mac-IACC)

These approaches were tested on the Box Pushing and Overcooked domains. The results show that macro-action learning outperforms primitive actor-critic approaches. Further out of all the approaches Mac-CAC and Mac-IAICC seem to perform consistently better than the others in all the domains.

While these approaches use the original policy gradient update, we will be extending it to the Trust Region Policy Optimization(TRPO) Schulman et al. (2015) algorithms (specifically Multi-agent PPO Yu et al. (2021)) and hope to get similar performance improvement for this project.

This project assumes that each domain has been predefined with a set of continuous macro-actions defined by a human. The continuous macro-actions will be tested on multiple custom domains for benchmarking; like macro-action versions of Multi Particle Environments MPE (Lowe et al., 2017) environments. These will then be compared to the primitive action multi-agent PPO(Yu et al., 2021). Initially, the goal is to design these environments for macro-action algorithms, so that they can take macro-actions as input. Then the next task is to extend Multi-Agent PPO to Continuous Macro-Actions.

4 Methods

This section describes the approach for the 2 parts of the project. The first goal was to create macro-action wrappers for the MPE environments (Lowe et al., 2017) that can take in a macro-action from the agent and convert it into the primitive action steps to run in the environment and collect the correct joint rewards and observations for the agent. The second goal was to develop and test macro-action multi-agent PPO, which is an extension of multi-agent PPO (Yu et al., 2021) in the macro-action MPE domain using the approaches mentioned in the background section. My focus was mainly on creating the environment wrappers and running and testing the macro-action MAPPO algorithm.

4.1 Creating Macro-Action MPE Wrappers

Using macro-actions for the agents' policy gives some abstraction to the agent and should make it easier to learn sub-goals and actions in a sample efficient way. There is a lot of flexibility in the type of macro-action to choose and also the low-level controller policy that the macro-action will follow.

The original MPE environments use 5 primitive actions for movement. UP, DOWN,

LEFT, RIGHT, NO-ACTION, and these are fairly consistent with all the environments. Some environments have an additional communication element where agents can send a communication signal to other agents for cooperation. This communication signal can be a discrete value from 1-10 [say_0, say_1, say_2, say_3, say_4, say_5, say_6, say_7, say_8, say_9].

The approach in this project was to use 2-dimensional x and y coordinates for the macro-actions and a low-level policy that will alternate between discrete x and y movements. This was chosen as it is fairly intuitive to understand and can be easily interpreted by a human. The agent then has to choose sub-goals that progressively get closer to the target goal while trying to avoid any actions that have negative rewards (like bumping into other agents or adversaries). An example action output would be (3,5), now the low-level policy will look at the agent's current position and create primitive actions that take it towards the given x,y coordinate. So if the agent is at (2,2) the primitive action output would be RIGHT->UP->RIGHT->RIGHT. For environments with communication, the agent would output a single integer between -1 to 1 that would then be transformed into the vector to send to other agents. The number of macro-action steps would be counted in the environment and once all agents are done then the macro-action is considered done and the agent will have to give a new macro-action.

Another change was to remove negative rewards (collisions between agents) from the environment and to have a separate cost estimate that would be used for multi-objective optimization. This would avoid the problem of reward shaping and makes sure we can maximize rewards and minimize costs in a multi-objective or constrained RL algorithm.

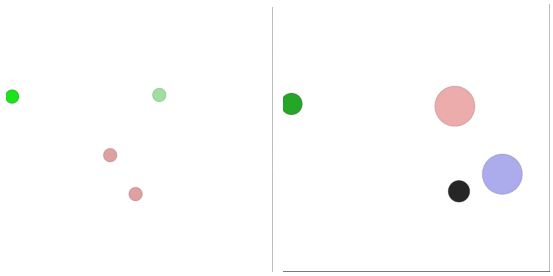

Figure 1: Simple Push and Simple Adversary Rendered Run

4.2 Continuous Macro-Action MAPPO

The Proximal Policy Optimization(PPO) algorithm is a model-free, on-policy RL algorithm that uses an actor-critic architecture. It works by optimizing a surrogate objective function that is designed to ensure that the new policy does not deviate too far from the old policy Schulman et al. (2017). The algorithm uses a clipped surrogate objective to prevent the policy from changing too much at once, which can lead to instability.

$$L_t^{CLIP+VF+S}(\theta) = \mathbb{E}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta(s_t)] \right]$$

where c1 and c2 are coefficients, S denotes an entropy bonus and $L_t^{VF}(\theta)$ is squared-error loss and $L_t^{CLIP}(\theta) = \mathbb{E}_t\left[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right]$

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 2: Original PPO algorithm (Schulman et al., 2017)

Multi-Agent Proximal Policy Optimization(MAPPO) (Yu et al., 2022) resembles the structure of single agent PPO by learning a policy π_θ and value function $V_\phi(s)$; these are represented as 2 separate neural networks. $V_\phi(s)$ is used for variance reduction and is only utilized during the training phase hence it can take extra global input not present in the agents local observations, thus following CTDE structure in multi-agent domains. For our experiments, The global input in each environment is the joint observations of all the agents' local information.

Macro-action-based MAPPO is more challenging than primitive MAPPO as it is hard to determine the agent's trajectories for updates and when to consider a macro-action terminated. This is discussed in detail in previous works (Xiao et al., 2022, 2020) section 3.3. To deal with the issues we use separate centralized critics for each agent as proposed in Xiao et al. (2022). However, we don't use the trajectory squeezing process described there and instead modify the buffer as described below in section 4.2.2 to get the correct replay trajectories from the buffer.

Upon testing this algorithm, we discovered certain issues with this approach that prevented the agents from learning a good policy. There were 2 main issues in our algorithm and they are explained in the following section:

4.2.1 Discounting

The returns for each agent were calculated using the sum of discounted rewards for the entire joint trajectory of the agents. However, this did not take into account the fact that macro-actions of different agents have different lengths. This was causing an incorrect sample to be used for the critic update and return calculation.

This was fixed by keeping track of the macro-action lengths for each agent and storing the gamma values per agent based on the macro-actions lengths during training. Hence

when calculating the discounted returns if we take this new discount then each agent gets the correct discounted return based on the macro-action length.

Old Discounting:

```
for t in range(returns.shape[0] - 2, -1, -1):
    returns[t] = rewards[t] + gamma * (1 - dones[t]) * returns[t+1]
```

As it can be seen the old discounting will have the same gamma for all agents regardless of the macro-action lengths, which leads to incorrect discounting.

New Discounting:

```
for t in range(returns.shape[0] - 2, -1, -1):
    returns[t] = rewards[:idx][t] + gamma * gammas[:idx][t]
                        * (1 - dones[:idx][t]) * returns[t+1]
```

In the new discount calculation, as we have stored the gamma values beforehand during training for each agent, we will be able to get the correct discount factor based on macro-action lengths.

4.2.2 Replay Buffer

Initially, the replay buffer would use Macro-Action Joint Experience Replay Trajectories that used a trajectory squeezing process described in (Xiao et al., 2020). However, this idea was then rejected to use normal experience replay trajectories with the above discounting change, and empirically the performance of the algorithm is still better than primitive MAPPO and also will have better memory and compute time. However an episode may end before a macro-action is complete, hence we need to add each agent's last macro-action to the buffer when the episode terminates before the end of a macro-action. This gives us the correct values for the updates from the buffer.

5 Experiments

5.1 Primitive Baselines on Wrapper Environments

The results on the macro-action wrappers were consistent with the baselines in previous works(Lowe et al., 2017). Thus proving that the wrapper environments are equivalent and can be used for creating the macro-action PPO benchmarks.

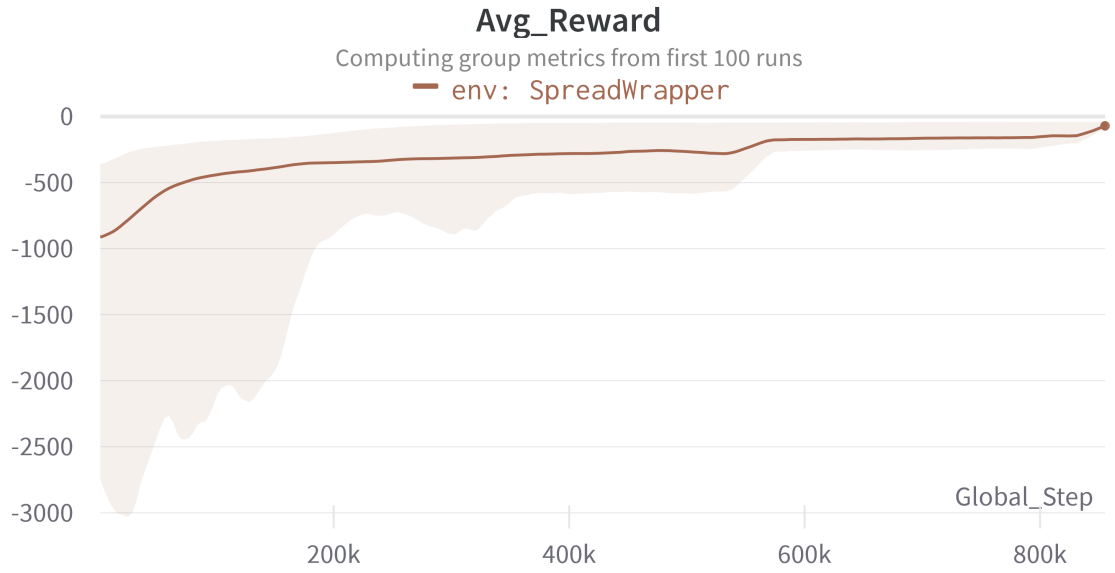

Figure 3: Primitive Simple Spread Average Rewards Over Multiple Runs

5.2 Macro-action PPO

The final rewards for macro-action PPO seem very similar to the primitive versions and macro-action ppo also converges faster than primitive ppo.

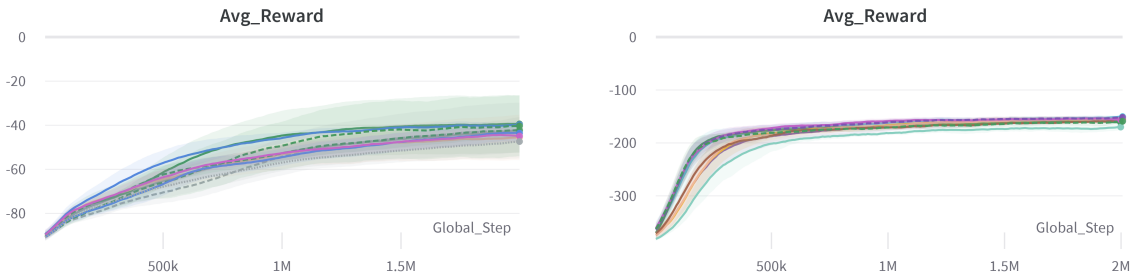

Figure 4: Macro-Action PPO Average Reward on Spread(Left) and Reference(Right)

Since we are using a Gaussian Actor for training, we decided to also have a separate evaluation reward with std deviation 0 to measure the agent's performance without noise. We measured the rewards for 10 evaluation episodes for every 5000 training episodes. Removing the Gaussian noise does not change the performance of the agent much compared to the training reward but it should reduce variance.

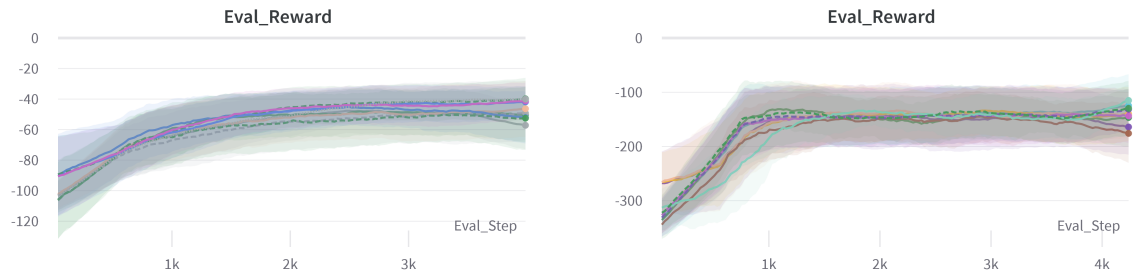

Figure 5: Evaluation Average Reward Simple Spread(Left) and Simple Reference(Right)

The number of macro-actions per episode was also evaluated as a metric. The macro-actions start at a small number and then usually increase as the agent gets better, as the goals are reached quicker towards the end and the agent oscillates near the target position.

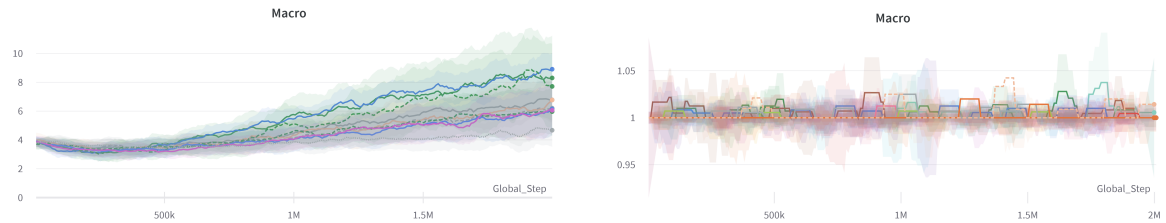

Figure 6: Macro-actions per episode Simple Spread(Left) and Reference(Right)

As we can see from the plot below for a comparison of simple spread between primitive action and continuous macro-action versions. The macro-action version outperforms the primitive-action version and is more sample efficient. Hence proving that Macro-action MAPPO is superior to MAPPO in these MPE domains.

Figure 7: Primitive vs Macro-action PPO Average Rewards

6 Discussion

This project introduces Macro-Action MAPPO for asynchronous multi-agent RL. This gives another method for training multiple agents asynchronously using macro-actions. Empirically, our methods are able to learn high-quality macro-action-based policies allowing agents to perform asynchronous collaborations in large and long-horizon problems.

The next obvious step would be to evaluate this in all of the MPE cooperative environments and benchmark all of them. This project can be further extended to the adversarial MPE environments to make sure that agents can learn simultaneously in an adversarial setting using the same approach. Another step would be to try different macro-actions and low-level controllers to check if the algorithm is robust to different types of macro-actions and low-level actions.

References

Amato, C., G. D. Konidaris, and L. P. Kaelbling (2014). Planning with macro-actions in decentralized pomdps. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '14, Richland, SC, pp. 1273–1280. International Foundation for Autonomous Agents and Multiagent Systems.

Bacon, P.-L., J. Harb, and D. Precup (2016). The option-critic architecture.

Lowe, R., Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*.

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). Asynchronous methods for deep reinforcement learning.

Oliehoek, F. and C. Amato (2016, 01). A concise introduction to decentralized pomdps.

Schulman, J., S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel (2015). Trust region policy optimization.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). Proximal policy optimization algorithms.

Xiao, Y., J. Hoffman, and C. Amato (2020, 30 Oct–01 Nov). Macro-action-based deep multi-agent reinforcement learning. In L. P. Kaelbling, D. Kragic, and K. Sugiura (Eds.), *Proceedings of the Conference on Robot Learning*, Volume 100 of *Proceedings of Machine Learning Research*, pp. 1146–1161. PMLR.

Xiao, Y., W. Tan, and C. Amato (2022). Asynchronous actor-critic for multi-agent reinforcement learning.

Yu, C., A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu (2022). The surprising effectiveness of PPO in cooperative multi-agent games. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Yu, C., A. Velu, E. Vinitsky, Y. Wang, A. M. Bayen, and Y. Wu (2021). The surprising effectiveness of MAPPO in cooperative, multi-agent games. *CoRR abs/2103.01955*.