Multi-agent Reinforcement Learning with Proximal Policy Optimization using an Actor-Critic Neural Network Model

Oswaldo Ilhuicatzi Mendizábal
School of Engineering and Sciences
Tecnológico de Monterrey
Campus Santa Fe
Email: A01781988@tec.mx

Mario Ignacio Frias Piña
School of Engineering and Sciences
Tecnológico de Monterrey
Campus Santa Fe
Email: A01782559@tec.mx

***Abstract*—This paper presents the implementation of a Multi-Agent Reinforcement Learning (MARL) framework using Proximal Policy Optimization (PPO), an actor-critic neural network-based algorithm. The study focuses on the "navigation" environment in the VMAS simulator, where multiple agents are tasked with reaching their respective goals while avoiding collisions and conflicts. The agents learn through centralized training and decentralized execution, leveraging Generalized Advantage Estimation (GAE) for stable and efficient learning. The results demonstrate that the PPO algorithm enables agents to achieve their objectives efficiently, with improved cooperation and reward maximization over training iterations. While the model effectively handles dense reward environments, its limitations include challenges with complex cooperative tasks and sparse rewards. This work highlights the potential of PPO in MARL applications and suggests avenues for further enhancements, including the use of centralized critics for better coordination among agents.**

***Index Terms*—Proximal Policy Optimization, Multi-Agent Reinforcement Learning, Generalized Advantage Estimation.**

1. Introduction

Since 2017, the field of artificial intelligence has experienced a significant revolution, more specifically in the domain of Reinforcement Learning (Schulman et al. 2017). This paradigm has been applied to optimize travel routes, train humanoid robots to walk, conduct strategic business simulations, and even develop autonomous vehicles.

These applications highlight decision-making challenges, many of which involve the interaction of multiple agents, situating them within the area of multi-agent reinforcement learning (MARL). In such environments, multiple agents operate simultaneously, each trying to achieve their individual goals based on their unique rewards. However, in certain scenarios, the agents' interests may conflict.

Thus, the objective of the model is to train and adapt multiple agents to reach their goals while avoiding interference with others, meaning cooperation is required among them. The simulation will only conclude when all agents have successfully reached their respective targets. The algorithms implemented to achieve this are detailed in the next section of this document.

2. Theoretical Framework

2.1. Proximal Policy Optimization

Proximal Policy Optimization (PPO), a variant of the Actor-Critic framework, is a widely-used reinforcement learning algorithm designed to optimize the behavior of agents by improving their policies iteratively. It achieves this by balancing exploration and exploitation while ensuring stable and efficient training. PPO operates within a policy gradient framework, which directly optimizes the policy - the agent's decision making process - by adjusting it to maximize the cumulative rewards gained by experiencing an environment with the policy dictating the actions to be taken in each state.

Proximal Policy Optimization (PPO) is a Model-Free algorithm of Reinforcement Learning which aims to learn the action policy best suited for the environment, without estimating the underlying dynamics or reward function of the environment. It undergoes a trial-and-error phase in which the policy is trying many different actions in order to find the best one for each state.

In the context of multi-agent reinforcement learning (MARL), PPO is particularly effective due to its flexibility and simplicity. It introduces a clipped objective function to constrain policy updates, preventing drastic changes that could destabilize training. This ensures that agents improve their strategies incrementally while adapting to dynamic, multi-agent environments where interactions and potential conflicts must be addressed.

2.1.1. PPO Loss Function. The Proximal Policy Optimization objective function is the collective loss to be optimized with the gradient descent process, it can be written as follows:

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

This function is further divided into 3 different steps that update different parts of the optimization step:

1) **Clipped Surrogate Objective:** Controls the updating of the policy that directly controls the actions taken by each actor.
2) **Value Loss Function:** Controls the value function given by the critic that estimates the value of the current state.
3) **Entropy Regularization:** Controls the randomness given to the optimization steps, this helps primarily as the way to model the balance between exploration and application of the learning process.

2.1.2. Clipped Surrogate Objective. The Clipped Surrogate Objective function is the loss obtained from the action-taking policy directly, it can be written as follows:

$$\mathbb{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right]$$

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio of the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$.
- $\hat{A}_t$ is the estimated advantage function at time step t.
- ϵ is a hyperparameter that determines the clipping range.
- The clip function ensures the ratio $r_t(\theta)$ stays within $[1 - \epsilon, 1 + \epsilon]$.

2.1.3. Value Loss Function. The Value Loss Function is the loss obtained from the critic compared to the real value of the observations, it can be written as follows:

$$\text{MSE} \left(r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} - V(s_t), V(s_t) \right)$$

- $r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1}$ is the advantage computed from the rewards given by the environment.
- $V(s_t)$ is the value estimated by the critic of the states in the exploration.

2.1.4. Entropy Regularization. Entropy Regulatization Loss is an extra value added during the PPO update process to add randomness to the policy to help it explore the environment thoroughly.

$$S[\pi_\theta](s_t) = -\int \pi_\theta(a_t \mid s_t) \log(\pi_\theta(a_t \mid s_t))$$

- $\pi_\theta(a_t \mid s_t)$ is the current policy on a given state and action pair.

This function is an integral when the environment is continuous, in our case where the environment is discrete the integral is replaced with a summation.

2.2. Generalized Advantage Estimation

The purpose of Generalized Advantage Estimation (GAE) is to balance the trade-off between bias and variance in policy gradient methods for reinforcement learning. Specifically, it helps to compute more stable and efficient advantage estimates, which are critical for training policies using algorithms like Proximal Policy Optimization (PPO). The Generalized Advantage Estimation (GAE) can be written as:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

Where:

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

Here:

- $\hat{A}_t$ is the advantage estimate at time t.
- γ is the discount factor.
- λ is the GAE hyperparameter for controlling bias vs. variance trade-off.
- δ_t is the temporal difference (TD) error at time t.
- r_t is the reward at time t.
- $V(s_t)$ is the value function estimate for state s_t.

The GAE can also be computed recursively as:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\hat{A}_{t+1}$$

This recursive form was used in our practical implementation for computational efficiency.

2.3. EMA Normalizer

The Exponential Moving Average (EMA) normalizer is defined as:

$$\text{EMA}_t = \alpha \cdot x_t + (1 - \alpha) \cdot \text{EMA}_{t-1}$$

where:

- EMA_t: The current EMA value at time t.
- x_t: The current data point at time t.
- EMA_{t-1}: The previous EMA value at time $t - 1$.
- α: The smoothing factor, $0 < \alpha \leq 1$, often calculated as:

$$\alpha = \frac{2}{N + 1}$$

where N is the number of periods considered for the moving average.

The EMA gives more weight to recent observations, making it sensitive to recent changes while still considering historical data.

2.4. About Actor-Critic

The actor-critic neural network is a powerful framework in reinforcement learning that combines two components: the actor and the critic, each represented by separate neural networks or shared parts of a single network. Its purpose is to improve learning stability and efficiency by simultaneously optimizing the policy (actor) and the value function (critic).

The actor-critic framework consists of two components:

1. Actor Loss:

$$L_{actor}(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi(a|s;\theta)\hat{A}(s,a)\right]$$

2. Critic Loss:

$$L_{critic}(\phi) = \mathbb{E}\left[\left(V(s;\phi) - R_t\right)^2\right]$$

3. Combined Loss:

$$L(\theta,\phi) = L_{actor}(\theta) + c \cdot L_{critic}(\phi)$$

Where:

- $\pi(a|s;\theta)$: Policy parameterized by θ.
- $V(s;\phi)$: Value function parameterized by ϕ.
- R_t: Target return.
- $\hat{A}(s,a)$: Advantage estimate (e.g., from GAE).
- c: Coefficient balancing actor and critic losses.

Actor-Critic (AC) and Proximal Policy Optimization (PPO) share similarities because PPO is a variant of the actor-critic framework. Specifically, both involve training a policy (actor) and a value function (critic). However, PPO builds upon the basic actor-critic approach by introducing mechanisms that improve stability and performance.

2.5. Centralized Learning

The learning process used during the training of the actors was centralized, the different agents all shared the same policy that was updated from the observations of each agent. This made the actions of the agents homogeneous in a specific state, but it also prevented the complete cooperation between them, as they each need their own specific reward to calculate the advantages correctly.

In our implementation of PPO there is a singular network which contains both the agent and the critic, they both share the initial layers of the Neural Network and only differ for their final layer. This was both to save on computational resources and for the critic to share the same initial processing with the actors.

3. Methodology

Proximal Policy Optimization (PPO) was chosen for this study due to its stability in policy updates and efficient learning of optimal behaviors in complex environments. The clipped surrogate objective ensures controlled policy updates, preventing divergence and improving sample efficiency. Consequently, the Pytorch library is imminent to work with the neural networks Actor and Critic.

3.1. Navigation Environment

3.1.1. Observation space description. In line with these principles, the model is working on a vectorized multiagent simulator (VMAS) environment called "Navigation". Here, the observation vector for each agent follows:

- **Agent's position:** $[x, y]$ (2-dimensional vector).
- **Agent's velocity:** $[v_x, v_y]$ (2-dimensional vector).
- **Relative goal positions:**
  - If `observe_all_goals=True`, relative positions to all goals: $2 \cdot n_{\text{agents}}$.
  - Otherwise, only the relative position to the agent's own goal: 2.
- **Lidar sensor readings:** $n_{\text{lidar\_rays}}$ (only if `collisions=True`).

The total observation dimension is:

$$\dim(\text{Observation}) = 4 + g + l,$$

where:

- $g = 2 \cdot n_{\text{agents}}$, if `observe_all_goals=True`, otherwise $g = 2$.
- $l = n_{\text{lidar\_rays}}$, if `collisions=True`, otherwise $l = 0$.

3.1.2. Action space description. In addition, the environment reports 9 possible discrete actions:

1) **Move Up:** The agent moves upward.
2) **Move Down:** The agent moves downward.
3) **Move Left:** The agent moves to the left.
4) **Move Right:** The agent moves to the right.
5) **Move Diagonally Up-Left:** The agent moves diagonally up and left.
6) **Move Diagonally Up-Right:** The agent moves diagonally up and right.
7) **Move Diagonally Down-Left:** The agent moves diagonally down and left.
8) **Move Diagonally Down-Right:** The agent moves diagonally down and right.
9) **No Action:** The agent remains stationary.

In the following diagram are represented the eight possible actions for the agents to move, the ninth being to remain stationary:

3.1.3. Reward system. In the environment, each agent receives its own reward depending on its actions. When it gets closer to its target, the agent receives a positive reward. However, the reward will be negative when it moves away from the goal or collides with another entity.

3.2. Model

The following diagram contains a visual representation of the layers of the neural network: it contains an input layer with the observations of size 18 given by the environment

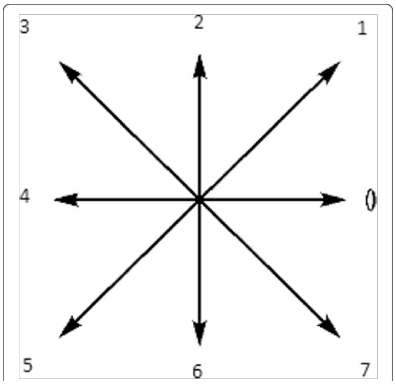

Figure 1. Diagram representing eight possible discrete actions.

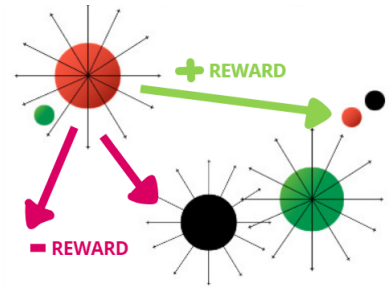

Figure 2. Diagram representing the reward system for the ”navigation” environment.

to each agent, 2 middle layers with 32 nodes each, and an output layer with 9 nodes representing the actions. The critic agent has the same initial layers, and only changes the last one to output a single scalar value.

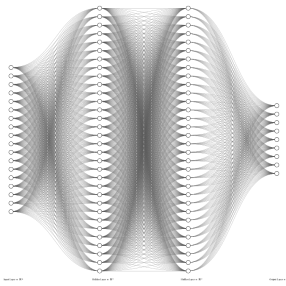

Figure 3. Neural Network Actor.

4. Results

The algorithm can help the agents achieve their goal in a decreasing number of steps while in the navigation environment, and by the end of their training, they were capable of consistently reaching their goals before the maximum number of steps were taken. At the same time, the mean loss was close to zero and the average reward was increasing, in other words, every agent’s performance to achieve their goal was delightful.

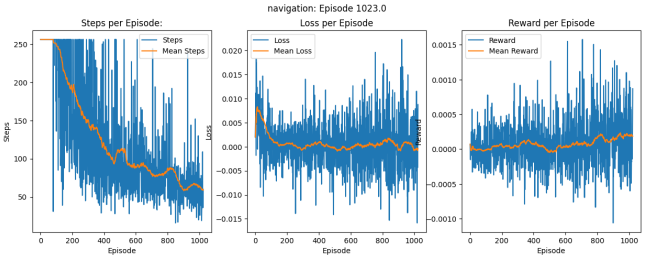

Figure 4. Navigation Improvements

The following figure is an example image of the environment, where each agent is moving towards their respective goals effectively.

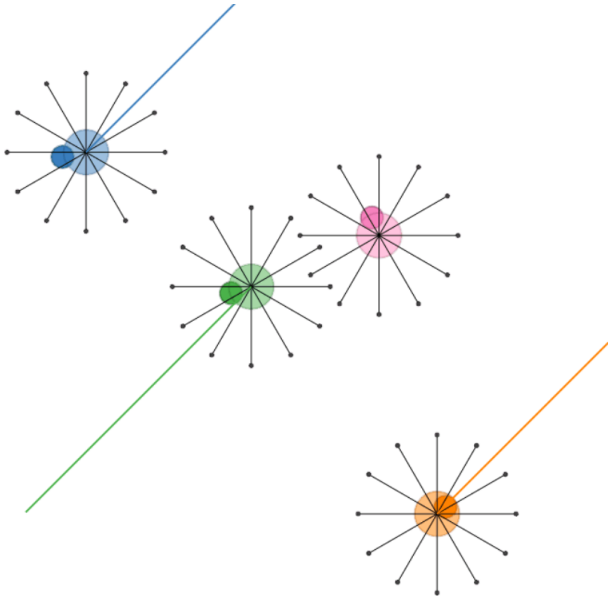

Figure 5. Navigation Environment Example

Through different simulations in the same environment, the training results are different, but they have similar behavior during testing; where they approach their goals while trying to evade the other agents when they get near.

5. Conclusions

At the end of this journey, the model was able to represent the expected behavior for each agent: they were able to learn how to ”cooperate”, or at least avoid being an obstacle for the other entities in the environment as they all seek to achieve their own goals.

The current implementation of the PPO algorithm was successful; nevertheless, it has several limitations on the environments that can be used. It requires the environment to

have dense rewards, where there are intermediary rewards towards reaching the goal. It also cannot model complex cooperation behavior due to each agent being their own agent and critic, they can only take into account their own observations so there is no effective inter-agent communication.

In the future, the improvements that can be made to the model would be to create a centralized critic that takes into account the observations of all the agents to provide better coordination.

References

OpenAI (2017). *proximal policy optimization — spinning up documentation*$_2$017. URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html#id3.

Schulman, John et al. (2017). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

Cai, Zhaowei et al. (2021). *Exponential Moving Average Normalization for Self-supervised and Semi-supervised Learning*. arXiv: 2101.08482 [cs.LG]. URL: https://arxiv.org/abs/2101.08482.

Bettini, Matteo et al. (2022). "VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning". In: *The 16th International Symposium on Distributed Autonomous Robotic Systems*.

Yu, Chao et al. (2022). *The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games*. arXiv: 2103.01955 [cs.LG]. URL: https://arxiv.org/abs/2103.01955.