Implementing OpenAI's PPO algorithm for RL and MARL environments

Jeremy Jung
jj523

David Forcinito
dvf26

Implementation: https://github.com/force-x/CS4756_final_project

1 Introduction

Policy gradient methods, while popular in deep reinforcement learning, are well-known for suffering from low stability and high variance. These issues stem from the fact that the basis of updating the gradient in these methods depend on random trajectories—inherently, this leads to high variability among trajectories which leads to high variability in the gradient.

Actor-Critic models are increasingly being used for their higher stability and smaller variance in comparison to policy gradients. These models subtract a baseline from the reward—implicitly, this leads to smaller variance since the gradients themselves are smaller, allowing for more stability. In this paper, we focus on OpenAI's Proximal Policy Optimization (PPO) algorithm, which is a variant of the Actor-Critic model that uses advantages as the baseline. The PPO algorithm is known for even improving more on the stability that Actor-Critic models already excel in by clipping policy updates at an upper and lower threshold, ensuring they are never too large.

Our paper creates an open and easy-to-use implementation of PPO and MAPPO, the latter of which is the multi-agent version of PPO that can be applied to multi-agent reinforcement learning (MARL) environments. We found that a lot of existing implementations of these algorithms, specifically MAPPO, are outdated and hard to generalize to new RL environments. Our implementation of the algorithms are built with PyTorch and very flexible, allowing them to work with both single-agent and multi-agent environments in various different libraries.

Our experiments test the application of our algorithms, PPO and MAPPO, to single-agent and multi-agent environments respectively. We measure the average reward over training iterations in the famous single-agent Cart Pole environment provided by OpenAI Gym and the multi-agent Simple Spread environment provided by PettingZoo. Our results demonstrate their gradual and stable improvement over time, illustrating their correctness as well as their stability.

2 Problem

2.1 Background

Actor-Critic models, as stated in the introduction, are based on subtracting from the reward a baseline, where the objective model can be modeled as follows [4]:

$L(\theta) = \mathbb{E}_t[\log \pi_\theta(a_t|s_t)(G_t - b(s))]$

A specific category of Actor-Critic models, called Advantage Actor-Critic models, takes in as the baseline function the value, or V, function. As is known with the policy gradient objective model, we can replace G_t with $Q(s_t, a_t)$ by doing some math with expectation values. Thus, our term $G_t - b(s)$ becomes $Q(s_t, a_t) - V(s_t)$, which we call the advantage, or $A(s_t, a_t)$. We call it the advantage because it measures how much our current action does better than the average action, which is modeled by $V(s_t)$. Thus, in the advantage case, we have that the objective model is:

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

$L(\theta) = \mathbb{E}_t[\log \pi_\theta(a_t|s_t)A_t]$

However great this model is, the Advantage Actor-Critic model still suffers from instability—if the gradient step size is too large, then the model is not stable. Thus, OpenAI came up with PPO to solve this exact problem. PPO clips the step size to make sure it is never too large, modeled by [1]:

$L(\theta) = \mathbb{E}_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$

The other part of PPO that makes it unique, as seen in the formula, is r_t, or the ratio. r_t is defined as [3]:

$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$

r_t is the ratio of taking a given action for a given state for the new and old policies.

Clearly, PPO improves stability—the algorithm ensures that no step size is ever too large, creating more stability in gradients.

PPO can also be applied to environments with multiple agents. In this case, it is called Multi-Agent PPO, or MAPPO. MAPPO differs from PPO in that it has a central value function for all agents [5]. In the case of MAPPO, we learn value function for the entire global state of the environment, as opposed to observations for each agent, with the rest of the PPO implementation staying the same.

2.2 Issues with current implementations

Current implementations struggle with adaptability to novel environments. We noticed that a lot of these algorithms are specifically built for certain types of environments and do not work for any novel environments. OpenAI's library, stable-baselines, for instance, has its version of PPO built specifically for Gymnasium environments, which contain only single-agent environments. If we were to run it on a multi-agent environment from PettingZoo, it would return the following result:

```
from pettingzoo.sisl import multiwalker_v9
from stable_baselines3 import PPO
from stable_baselines3.ppo.policies import MlpPolicy

env = multiwalker_v9.env()
model = PPO(MlpPolicy, env, verbose=0)
```

```
ValueError: The environment is of type <class 'pettingzoo.utils.
    wrappers.order_enforcing.OrderEnforcingWrapper'>, not a Gymnasium
    environment. In this case, we expect OpenAI Gym to be installed
    and the environment to be an OpenAI Gym environment.
```

Clearly, this is an issue—we would expect more adaptability to allow PPO algorithms to extend to the multi-agent case easily.

Furthermore, the official MAPPO algorithm also suffers from similar issues. As seen in the official implementation, it is only supported on four environments: SMAC, Hanabi, MPEs, and Google Research Football. We would also expect the MAPPO algorithm to be extendable to any multi-agent environment.

2.3 Environments

We applied our algorithms to two environments.

The first environment is the famous single-agent environment Cart Pole, which has a pendulum set standing up on a cart and rewards are given based on how well the agent, or the cart, balances the pendulum by moving left and right with a fixed amount of force.

The second environment is the Simple Spread environment, which is a multi-agent environment part of the Multi Particle Environments (MPE). MPEs are environments of particles, which are the agents, that can move and interact with each other. The Simple Spread environment in particular has the particles collaborating with one another to get as close as they can to all the landmarks—they are rewarded based on how close the closest particle is to each landmark while not colliding.

(a) Cart Pole (b) Simple Spread

Figure 1: Environments tested

3 Approach

For the PPO algorithm, we coded three neural networks—two policy networks and one value network. The two policy networks represent the actor, with one representing the current policy and the other representing the previous policy. The value network represents the critic.

The policy networks take in the state and output the probabilities of what action to take next. The two policy networks had the same architecture, defined with:

- A linear layer with input size of the state's dimension and output size of the hidden dimension
- A ReLU layer
- A linear layer with input size of the hidden dimension and output size of the action dimension
- A softmax layer that outputs probabilities for each action

The value network takes in the state and outputs the value of the state. The value network had an architecture as follows:

- A linear layer with input size of the state's dimension and output size of the hidden dimension
- A ReLU layer
- A linear layer with input size of the hidden dimension and output size of one

We defined the loss of our policy network as

$L^P(\theta) = min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)$

and the loss of our value network as

$L^V(\theta) = \frac{1}{2}(V_\theta(s_t) - V^{target})^2$

Here V^{target} is the discounted rewards-to-go. We adapted our algorithm for PPO from the pseudocode found in this article[2].

For the MAPPO algorithm, we similarly used two policy networks (one old and one new) and one value network. The value network, in this case, takes in as input to the first linear layer the global state of the environment, which is equal to the concatenation of the local observations of all of the agents. Thus, the value network now evaluates how good the global state of the environment is for all agents. The policy networks are similarly shared among all agents, with the policy network producing the optimal action distribution given an agent's current local observation. Beyond this, much of the algorithm is the same as in PPO.

We ran three experiments with our environments to test our algorithms. Our first two experiments were self-explanatory. The first evaluates the PPO algorithm's performance on the Cart Pole environment. The second evaluates the MAPPO algorithm's performance on the Simple Spread environment, where the model is trained from scratch.

Figure 2: Simple environment

Our final experiment attempted to use transfer learning in the multi-agent setting, which is something that we had not seen before while reading the literature on the topic. Essentially, we trained a single-agent PPO model on the Simple environment from PettingZoo. The Simple environment consists of a single agent that is rewarded based on how close it gets to a designated landmark. From that training on Simple, we then transferred the learned weights from the actor and critic networks to the actor and critic networks of a MAPPO model to be trained on Simple Spread. We hypothesized that this would lead to a quicker learning rate and higher overall rewards since our MAPPO model would be pre-populated with weights trained on a similar task.

For all of our PPO experiments, we ran our training loop for 100 iterations and collected 10 trajectories on each iteration. For all of our MAPPO experiments, we trained for 150 iterations and collected 40 trajectories per iteration. For both PPO and MAPPO, our current policy network was updated by the optimizer 10 times for each iteration of the training loop (i.e. 10 epochs).

4 Results

4.1 Cart Pole Environment Results

The following diagram depicts our PPO algorithm's performance on the Cart Pole environment. Note that the rewards at a given iteration were calculated by evaluating the current model over ten episodes and taking the average.

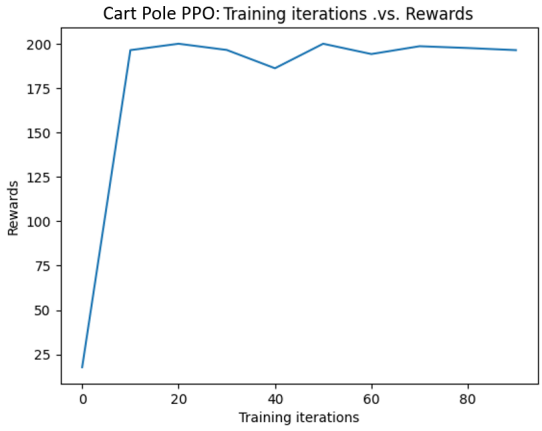


Figure 3: PPO Agent Performance in Cart Pole Environment

As evident in the graph, the PPO agent learns at an extremely quick rate. After just ten iterations of the training loop, the agent is receiving approximately 200 rewards per episode. For the remainder of training, the agent stagnates at around 200 rewards per episode except for a slight dip in performance forty iterations in. At the end of training, the agent was evaluated on 100 episodes and produced an average episode reward of 195.63. According to the Gym documentation, there are a maximum of 200 steps in the v0 version of Cart Pole with a reward of 1 for each step taken. Therefore, the

maximum attainable score in an episode is 200. Clearly, our agent learned the task of keeping the pole upright as it averages close to the maximal rewards. In addition, the graph is relatively smooth, which means the learning is very stable. Below is a graph of the rewards vs. training iterations for the vanilla Actor-Critic Model from assignment A3.

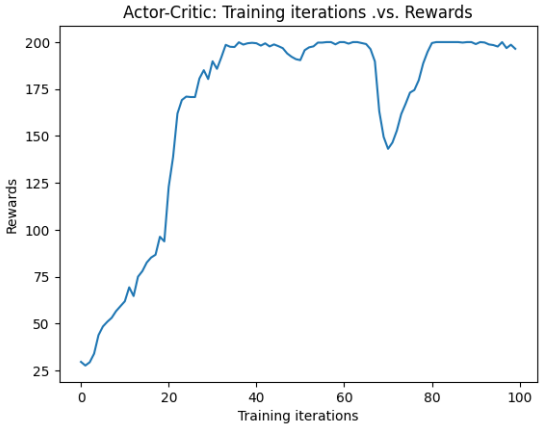


Figure 4: Vanilla Actor-Critic Model on Cart Pole Environment

Notice how this graph is quite jagged, which shows that the learning is more volatile. For, example, there is a large dip in rewards from iterations sixty to eighty. The clipped objective function in PPO helps eliminate dips such as this one by making the learning process more gradual and controlled.

4.2 Simple Spread Environment Results

The following diagram depicts our MAPPO algorithm's performance on the Cart Pole environment. Note that the rewards for a given episode were calculated by summing up the rewards received by each agent for each step of the environment. This was then done for ten episodes, and the average reward per episode was reported to make the plot.

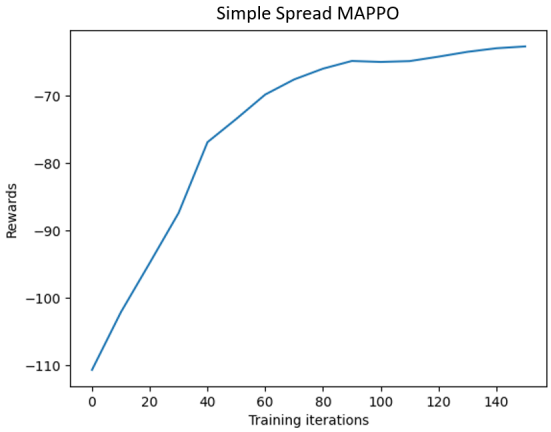


Figure 5: MAPPO Agent Performance in Simple Spread Environment with 3 Agents

The average rewards starts at around -110 and gradually climbs up to -65 at iteration ninety. After this point, learning levels off causing the average rewards to flat line around -65 for the remaining sixty iterations. One can immediately see that the learning rate in the multi-agent setting is drastically slower than in the single-agent setting. The plot in Figure X converges much quicker than Figure X. We suspect this is due to the fact that multi-agent learning is significantly harder than the single-agent case and takes more data and time to learn. The MAPPO model was also evaluated on 100 episodes

at the end of training and produced an average reward of -62.4. As a baseline comparison, the random policy that simply randomly samples an action at each time step produces average rewards of -117.1 over 100 episodes. So, our MAPPO agents are a significant improvement over the random policy and exhibit signs of learned reward-maximizing behavior.

4.3 Transfer Learning for the Simple Spread Environment Results

4.3.1 Simple Environment Results

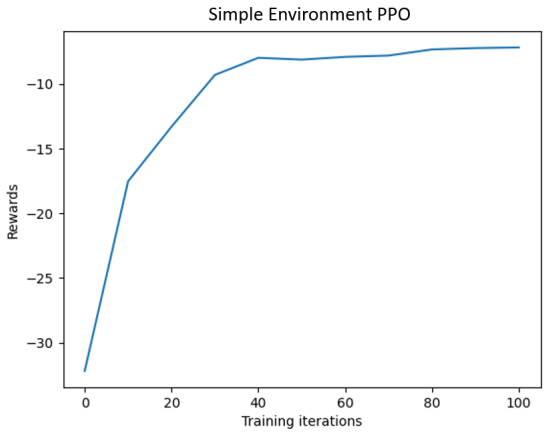


Figure 6: PPO Agent Performance in Simple Environment

As seen above, the average rewards over ten episodes starts at around -35 at the beginning of training. It then quickly climbs, reaching -10 average rewards by iteration thirty. After that point, the average rewards level off and remain around -8 for the rest of the training iterations. At the end of training, the average rewards over 100 episodes was -7.2, which is significantly better than the average rewards of the random policy at -40.4.

4.3.2 Simple Spread Environment Results (Transfer Learning)

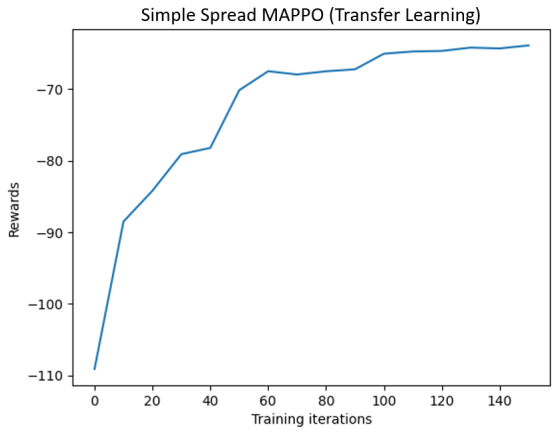
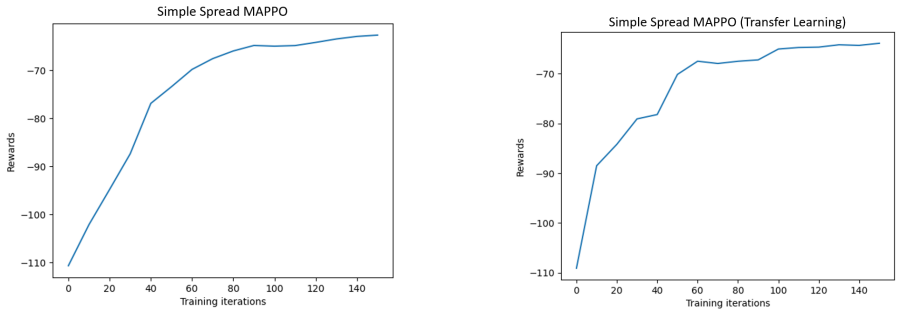


Figure 7: MAPPO Agent Performance in Simple Spread Environment with 3 Agents

In the plot above, one can see that the average rewards starts at -110 at the beginning of training. It then steeply climbs to -80 after just thirty iterations. At that point, the model stagnates for the next ten iterations before climbing again to -70 at iteration fifty. For the remaining iterations the average rewards asymptotically approaches about -65 average rewards. After fully training the model, it was then evaluated on 100 episodes and produced an average reward of -64.1. Comparatively, the random policy baseline over 100 episodes was -115.9.

4.3.3 Comparing "Cold Start" vs. Transfer Learning



(a) Cold Start (b) Transfer Learning

Figure 8: Cold-Started vs. Transfer Learning MAPPO

From the figures above, one can see that the MAPPO model that exploits transfer learning does initially exhibit a faster learning rate. For the first thirty iterations, the transfer learning model outpaces the model trained from scratch. For example, at iteration ten, the cold-started model obtains -105 average rewards while the transfer learning model has a reward value around -80. However, this is short-live as by iteration forty both models obtain the same rewards. The transfer learning model also has another short burst of learning from iterations forty to sixty, which outpaces cold-started model. After that point though, both models level off and hover around -65 average rewards.

4.4 Limitations

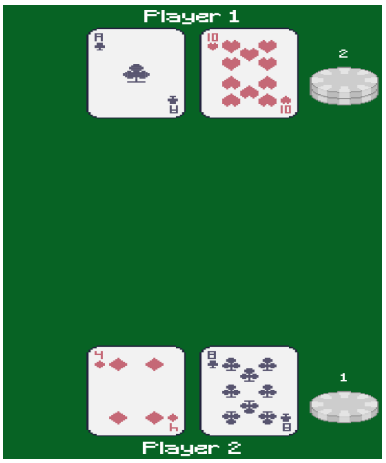
Notably, there are some limitations to our work. For one, our current PPO and MAPPO implementations only support discrete action spaces. Therefore, if one wants to use our implementation in an environment where the action space is continuous, they would need to make modifications to our code base. We hope this should be as simple as changing the output dimension of the actor network and altering how an action is selected.

In addition, our MAPPO implementation uses a shared actor network and critic network for all agents (i.e. parameter sharing). This is ideal for homogeneous environments, like Simple Spread, where all agents have the same observation and action space and the same reward function. However, if this were not the case, like in adversarial multi-agent games, then our implementation would no longer work.

Lastly, our implementation does not allow users to output a video of their trained model executing an episode. We believe this is not due to a fault of our own. Our implementation does support the recording of episodes. Instead, we believe that a bug exists in the PettingZoo code base that affects the rendering of MPE environments. Essentially, the rendered image of an MPE environment is always a purely black image when using render mode “rgb_array". For example, the following code was run on both a Classic environment and an MPE environment.

```
env.reset()
frame = env.render()
img = Image.fromarray(frame, 'RGB')
img.show()
```

However, the difference in results are drastic:



(a) Classic Environment (Poker)

(b) MPE Environment (Simple Spread)

Figure 9: Rendering Two Types of Environments

5 Conclusion

Overall, we have successful accomplished our goal of developing an open source and easy-to-use implementation of PPO and MAPPO. Our results show that our implementations are able learn on common RL and MARL environments and significantly outperform baselines set by random policies. We have also shown that PPO methods are more stable than traditional Actor-Critic models due to their clipped objective functions. Lastly, we have shown that by using transfer learning one can potentially increase the learning rate of a MAPPO model.

It is important to remember that our work focuses on virtual environments. However, in the future, MARL techniques can control drone swarms to coordinate search and rescue efforts and help self-driving cars communicate with each other safely. While we have not realized these goals ourselves, we hope that our work will enable advancement in the field of multi-agent reinforcement learning.

6 Role of Group Members

The workload of this project was split evenly between the two of us. As for the code, rather than taking on assigned roles, we took turns working in sprints on the project. For example, using Jeremy's Actor-Critic work from assignment A3, David was able to modify it to implement PPO on Cart Pole. Later, Jeremy typed up the framework for MAPPO, which David then ran on the Simple and Simple Spread environments. For this report, the sections were divided evenly between us. Similarly, the slides for our video presentation were also a 50/50 split.

References

[1] "Proximal policy optimization (ppo)." [Online]. Available: https://huggingface.co/blog/deep-rl-ppo

[2] "Proximal policy optimization¶." [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html#documentation-pytorch-version

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[4] C. Yoon, "Understanding actor critic methods," Feb 2019. [Online]. Available: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

[5] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," 2022.