CENTRALIZED PPO IN MULTI-AGENT REINFORCEMENT LEARNING [*]

Anish Sahoo
CS 4180
Northeastern University
Boston, MA
sahoo.an@northeastern.edu

ABSTRACT

This paper explores the use of centralized Proximal Policy Optimization (PPO) in multi-agent reinforcement learning. We use the KnightsArchersZombies environment from PettingZoo to train a model for multiple agents with a centralized, shared actor-critic model. We implement a version of PPO inspired by the original PPO paper and the MAPPO paper. We observe that the centralized PPO model without any temporal data converges to a suboptimal policy. We then update our model to include temporal data and observe that the model converges to a much better policy. We also observe some emergent behavior that suggests that the agents are learning to cooperate. We conclude that centralized PPO with temporal data is a promising approach to multi-agent reinforcement learning that encourages cooperation and generalization.

1 Introduction

Multi-agent reinforcement learning (MARL) is a branch of reinforcement learning that focuses on training multiple agents to interact with each other and their environment. MARL is a challenging problem because the agents must learn to cooperate and compete with each other in a complex, dynamic environment. The KnightsArchersZombies environment presents a unique challenge, simulating a scenario where different agent types (knights, archers, and zombies) interact within a 720 by 1280 pixel map. Each agent type possesses unique characteristics and capabilities, creating complex interaction dynamics that make it hard to learn optimal policies.

This research focuses on Proximal Policy Optimization (PPO), a state-of-the-art policy gradient method. Introduced by OpenAI [1] in 2017, PPO has become very popular in the reinforcement learning community due to its ability to balance exploration and exploitation while being very sample efficient. Additionally, due to its clipped objective, PPO is known for being more stable than other policy gradient methods like REINFORCE.

We implement a specific configuration of PPO that aims to be more efficient and effective in multi-agent environments. Instead of using separate actor-critic models for each agent like in the MAPPO paper [2], we implement a centralized, shared actor-critic model that allows agents to share information and perhaps learn from each other. By leveraging this version of PPO, we aim to investigate if agents can learn from each other and develop effective strategies. We hypothesize that this centralized model will enable agents to develop more effective strategies by leveraging the collective knowledge of the group.

Our study seeks to understand how this version of PPO performs in this environment and whether the centralized shared model can help agents learn effective strategies.

[*] *Citation*: **Authors. Title. Pages.... DOI:000000/11111.**

2 Methodology

2.1 Environment

Knights Archers Zombies is a multi-agent environment from PettingZoo butterfly family of environments. The environment consists of three types of agents: knights, archers, and zombies. The knights and archers are controlled by agent(s), while the zombies are controlled by the environment. The goal of the knights and archers is to kill the zombies, while the goal of the zombies is to kill the knights and archers. Zombies are spawned randomly on the top of the 720 by 1280 pixel map, while knights and archers are spawned on the bottom of the map. The agents die when they collide with a zombie.

The environment can be formally defined as a Markov Decision Process (MDP) for each agent:

- $\mathcal{S} = \{s \in [0, 255]^{512\times512\times3}\}$, where each state is a 512 by 512 pixel RGB image of the area around the agent
- $\mathcal{A} = \{$ UP, DOWN, LEFT, RIGHT, ATTACK $\}$
- $\mathcal{R}(s, a)$: $+1$ for killing a zombie, otherwise 0
- $\mathcal{T}(s'|s, a)$: The environment is deterministic in agent movement but stochastic in zombie movement/spawning.

An episode ends in one of the following scenarios:

- All knights and archers are dead
- All zombies are dead
- A zombie reaches the bottom of the map

We will evaluate whether the agents can learn to play by observing the total reward accumulated over an episode. If the policy is very good, the agents should be able to kill most the zombies without any of the knights or archers dying. A good baseline reward would be around 25-30 per episode.

2.2 The PPO Algorithm

Proximal Policy Optimization (PPO) is a policy gradient method that aims to maximize the expected return of an agent by updating its policy in the direction that increases the return. PPO is an on-policy method, meaning that it learns from fresh data it has collected using the current policy. This makes PPO more sample efficient than off-policy methods like DDPG and DQN.

PPO is supported by two key ideas: the clipped objective and the importance ratio. The equation for the clipped objective is:

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

In this equation, $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the importance ratio.

The clipped objective is a modification to the policy gradient objective that prevents the policy from changing too much in one direction. This helps stabilize the learning process and prevent the policy from diverging.

The importance ratio is the ratio of the probability of taking an action under the new policy to the probability of taking the same action under the old policy. This ratio is used to adjust the policy update in the direction that increases the return.

The PPO algorithm can be summarized as follows:

1. Collect data by running the current policy in the environment
2. Compute the advantage function using the collected data
3. Compute the policy gradient using the advantage function
4. Update the policy using the policy gradient
5. Repeat until convergence

PPO usually involves training a neural network to approximate the policy and value function. The policy network takes the state as input and outputs the probability distribution over actions, while the value network takes the state as input and outputs the value of the state. The policy network is updated using the policy gradient, while the value network is updated using the value loss.

We implement a 'centralized' version which has the following key differences:

1. Shared actor-critic model - The Actor and Critic networks share the same weights
2. Centralized training and decisions - All agents use the same network to learn and make decisions
3. Centralized data collection - All agents share the same data buffer

As this is a multi-agent problem, we also incorporate some features from the MAPPO[2] paper to improve the performance of the model:

1. GAE - We use the Generalized Advantage Estimation (GAE) to compute the advantage function
2. Entropy regularization - We use entropy regularization to encourage exploration
3. Temporal Data - In our second implementation, we incorporate temporal data by stacking frames

3 Initial Implementation: No Temporal Data

3.1 Network architecture

Our initial network architecture consists of a shared actor-critic model with two separate heads for the actor and critic. The actor head outputs the probability distribution over actions, while the critic head outputs the value of the state. The network consists of three convolutional layers followed by a fully connected layer.

3.2 Environment setup

We use the KnightsArchersZombies environment from PettingZoo to train our model. The environment is initialized with the following configuration:

- Spawn Rate:15
- Number of Archers: 2
- Number of Knights: 2
- Maximum number of Zombies in an episode: 30

Before passing the state to the network, we preprocess the state using SuperSuit [3] wrappers by resizing it to 84 by 84 pixels and converting it to grayscale.

Table 1: Hyperparameters

hyperparameter	value
total timesteps	10000
rollout size	5000
data buffer capacity	8000
reward scale	1
epochs	6
minibatch size	256
clip epsilon	0.2
value coefficient	0.5
entropy coefficient	0.03
discount factor	0.99
GAE lambda	0.95
value loss	0.5
huber delta	10
optimizer	Adam
optimizer learning rate	0.00005
network initialization	Orthogonal

We train the model for 10,000 timesteps with a rollout size of 5000 and a data buffer capacity of 8000.

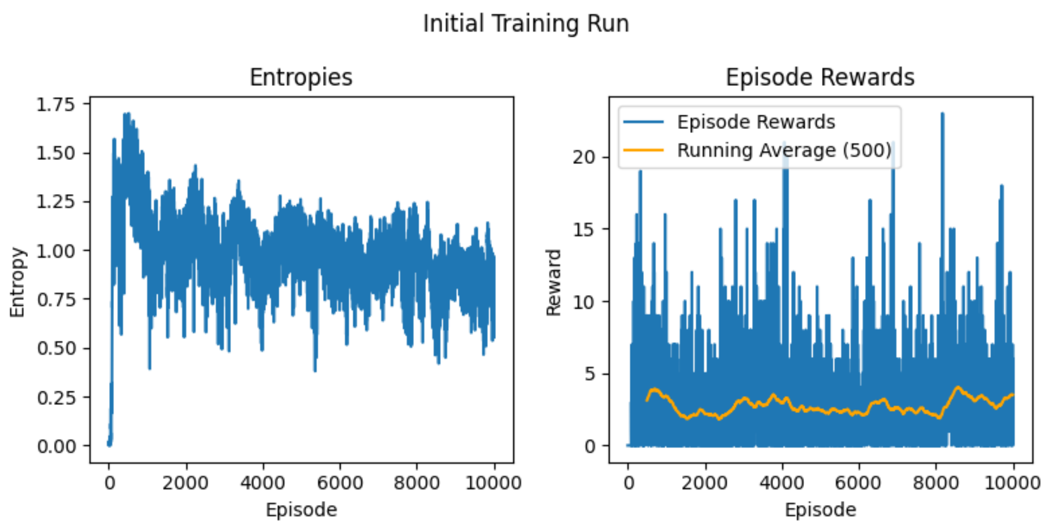

Figure 1: Training results

3.3 Results

The training results show that the model converges to a suboptimal policy with a total reward of around 6 per episode. As the entropy of the policy decreases, it shows that exploration is decreasing and the policy is converging suboptimally. This suggests that the model is not learning an effective strategy to kill the zombies. This could be due to the fact that the model is not able to capture zombie movement patterns properly. This makes it harder for the agent to predict zombies and targeting them effectively.

4 Improved Implementation: Incorporating Temporal Data

4.1 Motivation for change

The initial implementation did not perform well, suggesting that the model was not learning an effective policy. We hypothesized that the model was not able to learn an effective policy because it was not able to capture the temporal dynamics of the environment properly. The model was only able to see a single frame at a time, which made it hard capture patterns of movement and learn those details. To address this issue, we decided to incorporate temporal data into the model in the form of stacked frames.

4.2 Changes to architecture

We updated the network architecture to include a stack of 4 frames as input to the network. This allows the network to capture the temporal dynamics of the environment and learn patterns of movement. The network architecture now consists of three convolutional layers followed by a fully connected layer. The input to the network is a stack of 4 frames of size 84 by 84 pixels.

4.3 Updates to environment setup

We updated the environment setup to include a stack of 4 frames as input to the network. We used the SuperSuit wrappers to stack the frames before passing them to the network. We also modify the entropy coefficient to a nonstandard value 0.04 to encourage exploration and learning of a better policy.

4.4 Results

We see a drastic improvement in the performance of the model after incorporating temporal data. While not a lot, the total reward per episode has increased to around 10. The entropy of the policy is also higher, suggesting that the model is exploring more and learning a better policy.

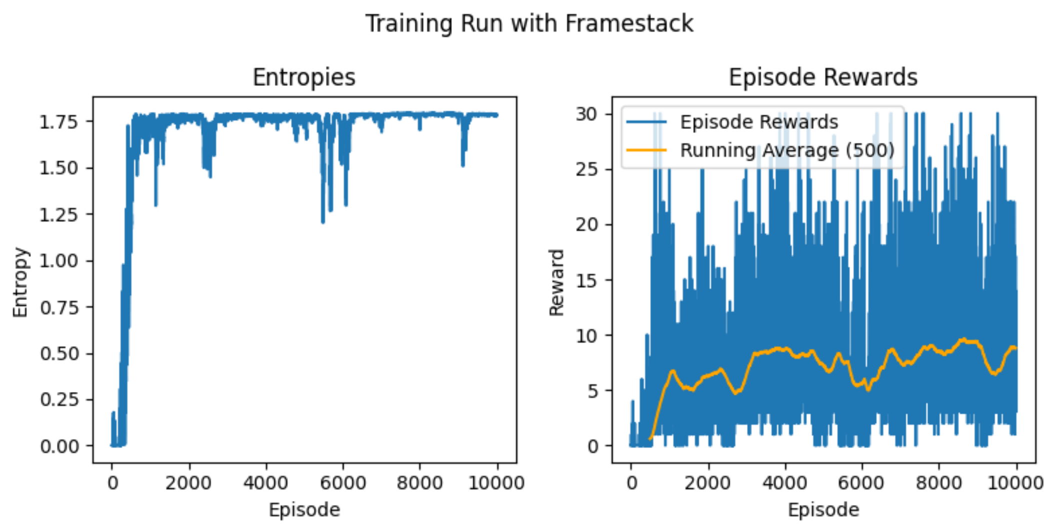

Figure 2: Training results with temporal data

The entropy of the policy is high till the very end, which means that it has kept exploring and learning throughout the training process. The average episodic reward during training has increased to around 8, which is a significant improvement over the initial implementation. However, this also indicates that the model has not yet converged to an optimal policy and there is still room for improvement. We estimate that training for at least 50,000 timesteps would be required to reach a good policy.

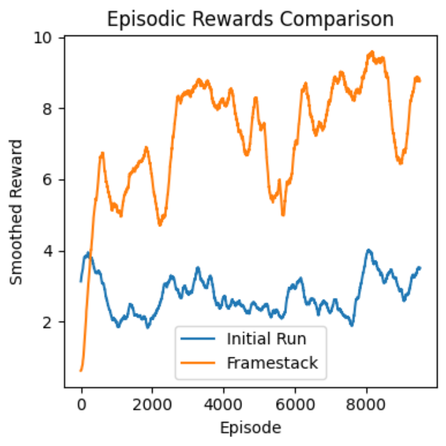

Figure 3: Difference in average episodic rewards between the two implementations

5 Emergent Behavior

We find that the agents have learned to cooperate and target different directions, covering each other's blind spots and maximizing the number of zombies killed. The agents have also learned to avoid the zombies and not collide with them, which is a good sign that the model is learning an effective policy. Surprisingly, the agents have learned to predict zombies outside their observation limits (512 by 512 pixels) and move towards them, suggesting that the agents are learning to plan ahead and anticipate the movement of the zombies.

- Interesting observations
- Qualitative analysis of agent behavior
- Insights gained from the temporal data approach

6 Discussion

6.1 Implications

- Theoretical implications of the findings
- hybrid centralized policy for all agents
- Why temporal data made a significant difference
- Broader insights into multi-agent reinforcement learning

6.2 Limitations

- Limitations of the current approach

6.3 Future Work

- Potential future work

7 Conclusion

Your conclusion here

Acknowledgments

This was was supported in part by Professor Chris Amato and the CS 4180 course staff at Northeastern University.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[2] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre M. Bayen, and Yi Wu. The surprising effectiveness of MAPPO in cooperative, multi-agent games. *CoRR*, abs/2103.01955, 2021.

[3] J. K Terry, Benjamin Black, and Ananth Hari. Supersuit: Simple microwrappers for reinforcement learning environments. *arXiv preprint arXiv:2008.08932*, 2020.