Cooperative Ping Pong Game Multi-Agent Reinforcement Learning

Guided by ,

Professor Jahan Ghofraniha

Submitted By,

Teepika Ramasamy Marimuthu

Jagruti Mohanty

Duncan Inganji

Table Of Contents

Executive Summary

We developed a cooperative ping pong multiplayer game using Multi-agent Reinforcement Learning. In PettingZoo, we created a gym vector environment with multiple copies of the environment and concatenated the parallel environments during training. Using Super Suit functions, we preprocessed the steps such as adjusting the frame size, stacking the parallel environment and frames, etc. is done in the reinforcement learning environment. We have developed cooperative ping pong games with four models using PPO and A2C reinforcement learning algorithms. The models are built using stable baseline cnnpolicy as well as building cnnpolicy from scratch. It is observed with the results that PPO with custom-built cnnpolicy and hyper-parameter tuning performed better than the A2C model.

Background/Introduction

Multiagent reinforcement learning is also based on a similar principle as single-agent reinforcement learning. Each agent is trying to learn its policy to optimize its rewards.
But using a central policy for all agents and using a central node to compute the actions gets complex as the workers increase. Hence a decentralized multiagent reinforcement is used. The model replaces the main orchestrator with a connected graph in which all the agents train independently and then share some intermediate values or parameters with their neighbors. By communicating with each other, nearby agents reach a common objective. As information is passed across the network, every agent gets the data from every other agent's learning process. As the agents can only interact with their neighboring agents, the computational complexity and communication overhead per agent increases linearly with the number of neighbors instead of the total number of agents.

Problem Statement

There are two players/agents in a cooperative ping pong game, there are two players/agents, and the objective is to play the ball between these agents for the longest time. One agent keeps moving top to bottom on the left edge and the other on the right edge. Once the ball gets out of bounds, the game is terminated. We developed this game using the Butterfly environment of Pettingzoo.

The problem that we aim to solve is a multi-agent reinforcement learning (MARL) model using a pettingzoo multi-agent environment that can learn to coordinate and interact with each other. In this problem, two agents interact in the background.

Purpose/Motivation

We are using reinforcement learning to build a system that learns to play multiplayer games. The goal in multiplayer games is to have coordination, plan the next action based on other players' moves, and control multiple agents at once. All other environments require a high degree of coordination and require learning of emergent behaviors to achieve an optimal policy. As such, these environments are currently very challenging to learn.

Differentiator/Contributor

This currently proposed implementation plans to use PettingZoo along with the SuperSuit multi-agent version of it for development. There is currently no implementation available in online resources that are using pettingzoo butterfly environment, supersuit library for preprocessing for the cooperative ping pong game. For the implementation of this project, we have gone through several online resources and API documentation of pettingzoo, baseline algorithms (A2C and PPO).

We have developed four models:

1. Using PPO with cnnpolicy from the stable-baselines and hyper tuning the parameters.
2. Using PPO with custom cnnpolicy that we build and passed to the PPO model
3. Using A2C with cnnpolicy from the stable-baselines and hyper tuning the parameters.
4. Using A2C with a custom cnnpolicy that we built and passed to the A2C model.

As the first step, we have done the preprocessing using supersuit library functions for preprocessing the environment parameters and then passed it into the model. We have implemented, tuned the hyperparameters for having the best output model. We have run all the models for 500,000 steps.

For the PPO model below are the parameters that we have tuned :

- Policy type - We have taken the Cnnpolicy as this is an image-based model. For one model, we used the baseline cnnpolicy, and for the other model, we have created our own ccnpolicy and passed it as an argument. We have built a custom cnn model using conv2d layers and relu activation function. This is passed using policy_kwargs into the baseline PPO reinforcement learning algorithm.
- Env - We have stacked eight environments for cooperative ping pong game from the pettingZoo environment and concatenate them into one, and passed them into the PPO model
- Gamma value is the discount factor, and for the PPO model, we have set it to 0.95
- N_steps - It is the number of steps to run for each environment per update, and the environments run in parallel. In this case, we have set it to - 256
- Vf_coef is the value function for the loss calculation - 0.042202
- Ent_coef is the entropy coefficient for the loss calculation - 0.09051
- The learning rate value is set to - 0.0006221
- Clipping range is the value for the gradient clipping - 0.3
- Verbose is set to 3 and will print out the details of the steps during the training process.
- Enabled tensorboard logging

For the A2C model below are the parameters that were tuned :

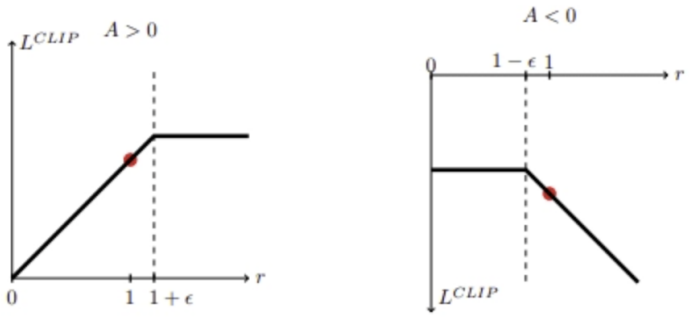
- Policy type - We have taken the Cnnpolicy as this is an image-based model. For one model, we used the baseline cnnpolicy, and for the other model, we have created our own ccnpolicy and passed it as the argument. We have built a custom cnn model using conv2d layers and relu activation function. This is passed using policy_kwargs into the baseline A2C reinforcement learning algorithm.
- Env - We have stacked eight environments for cooperative ping pong game from pettingZoo environment and concatenate them into one and passed them into the A2C model
- Gamma value is the discount factor, and for the A2C model, we have set it to 0.98
- N_steps - It is the number of steps to run for each environment per update, and the environments run in parallel. In this case, we have set it to 5
- Vf_coef is the value function for the loss calculation 0.49
- Ent_coef is the entropy coefficient for the loss calculation 0.0
- The learning rate value is set to
- Max_grad_norm is the maximum value for the gradient clipping 0.5
- RMS Epsilon value stabilizes the computation 1e-05
- Verbose is set to 3 and will print out the details of the steps during the training process.
- Enabled tensorboard logging

Methodology

The methods Proximal Policy Optimization(PPO) and Advantage Actor-Critic(A2C) are policy gradient approaches. They have an advantage function along with the objective function or loss function of the vanilla policy gradient. This approach helps to reduce the variance in gradient between the old and new policies. The advantage function estimates how good an action is compared to the average action for a specific state. The reduction in variance increases the stability of the RL algorithm. If the advantage function is positive, that implies the action taken by the agent is good and can get a good reward by taking action. If the advantage function is negative, then it implies the action taken by the agent is not going to fetch a good reward; hence the probability of the action is reduced.

Proximal Policy Optimization(PPO)

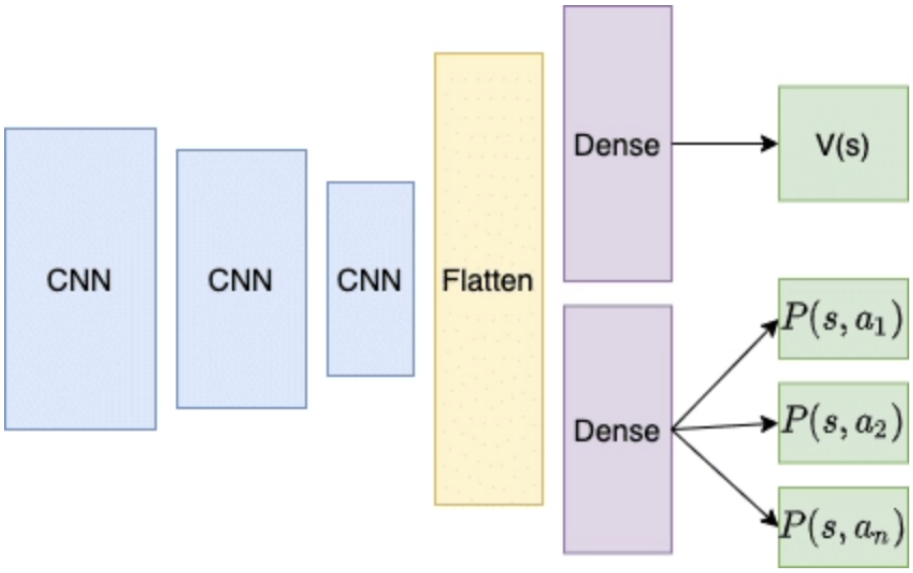
PPO is a first-order optimization algorithm, and it defines the probability ratio between the new policy and the old policy. In this case, the policy is updated explicitly. PPO function objective function takes the value between the original and the clipped value, and hence it does not deviate largely and lies within a range. In this case, as well the positive advantage and negative advantage are indicators of good and bad actions. This is shown in figure below



PPO clipped loss function (from https://arxiv.org/pdf/1707.06347.pdf)

Advantage Actor Critic(A2C)

In the Actor-Critic Model, the training of actor and critic models happens in separate networks. The Actor model takes in the input state and, based on policy, outputs the best possible actions. It controls how the agent behaves by learning the optimal approach. The Critic Model evaluates these actions by computing the value function and updating weights at each step and not at the end of each episode of the training process. The computation of Q values can be decomposed into two pieces: the state Value function V(s) and the advantage value A(s, a).A2C evaluation of action is based on how good the step is on how much better it can be. It does it using the advantage function to show how better the action is compared to others .A network value function captures how good it is to be in that state. The A2C model overcomes the vanilla policy gradient approach as it is less prone to variance and noisy data.A2C with multiple agents will wait for all the agents to finish and update the global network weights and results from all agents.



A2C architecture

Source Citation : Samsami, Mohammad Reza & Alimadad, Hossein. (2020). Distributed Deep Reinforcement Learning: An Overview.

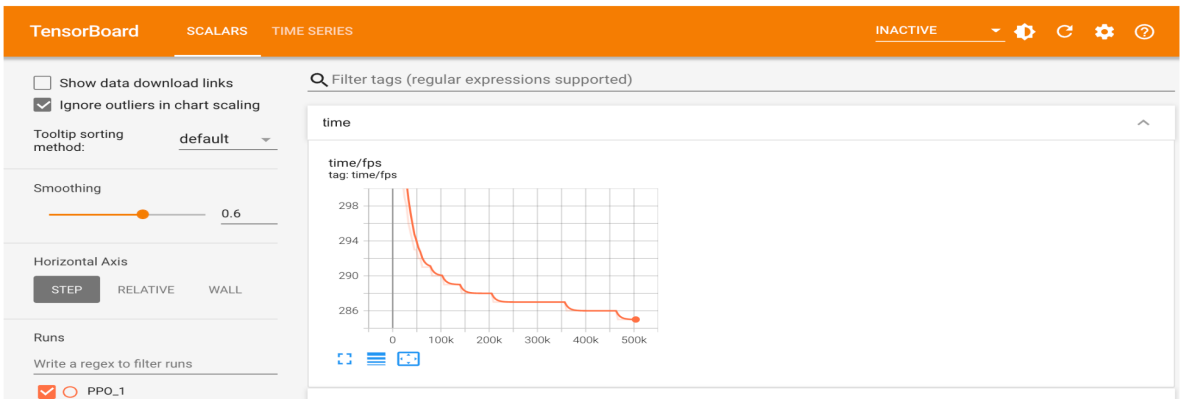
Implementation/Results

Based on the observations of the results, A2C algorithms vary a lot with minor changes in hyperparameters. PPO algorithm is much stable, and the training time is lesser compared to A2C. Proximal Policy Optimization performs better than A2C as per our training model, and it's simpler to tune the hyperparameters for it.

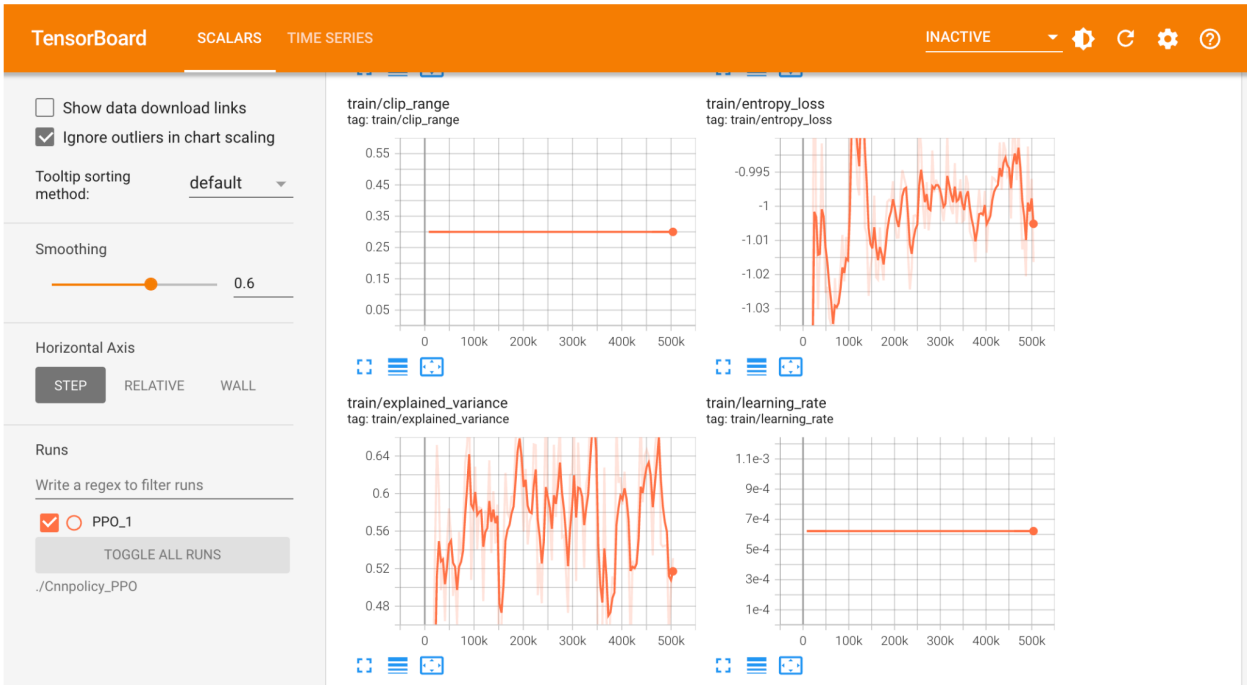
Results from Tensorboard :

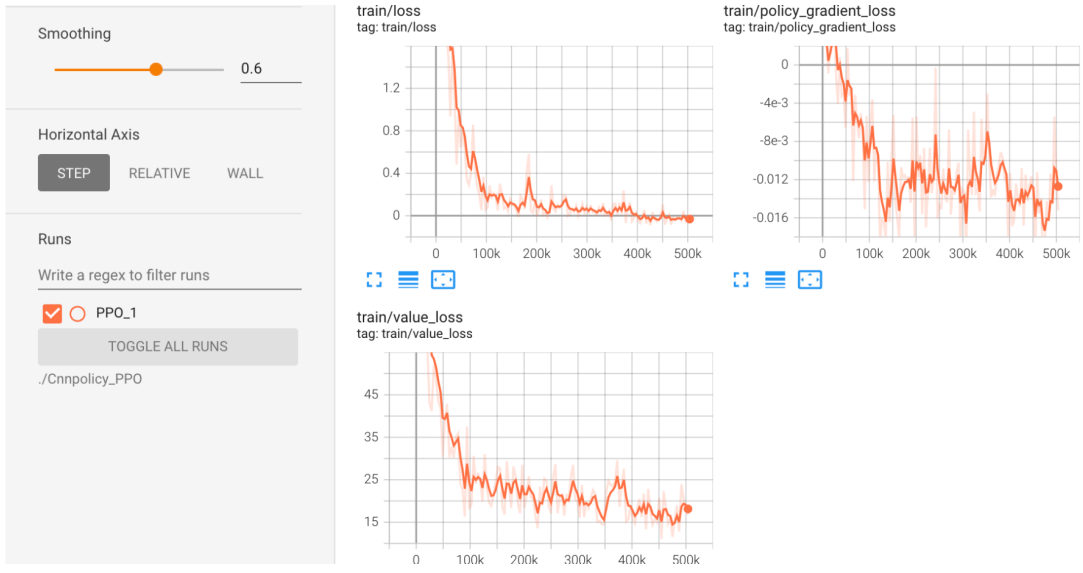
CNNpolicy with PPO:

```
# Load the Tensorboard notebook extension
%load_ext tensorboard

%tensorboard --logdir='./Cnnpolicy_PPO'
```



```
%tensorboard --logdir='./Cnnpolicy_PPO'
```

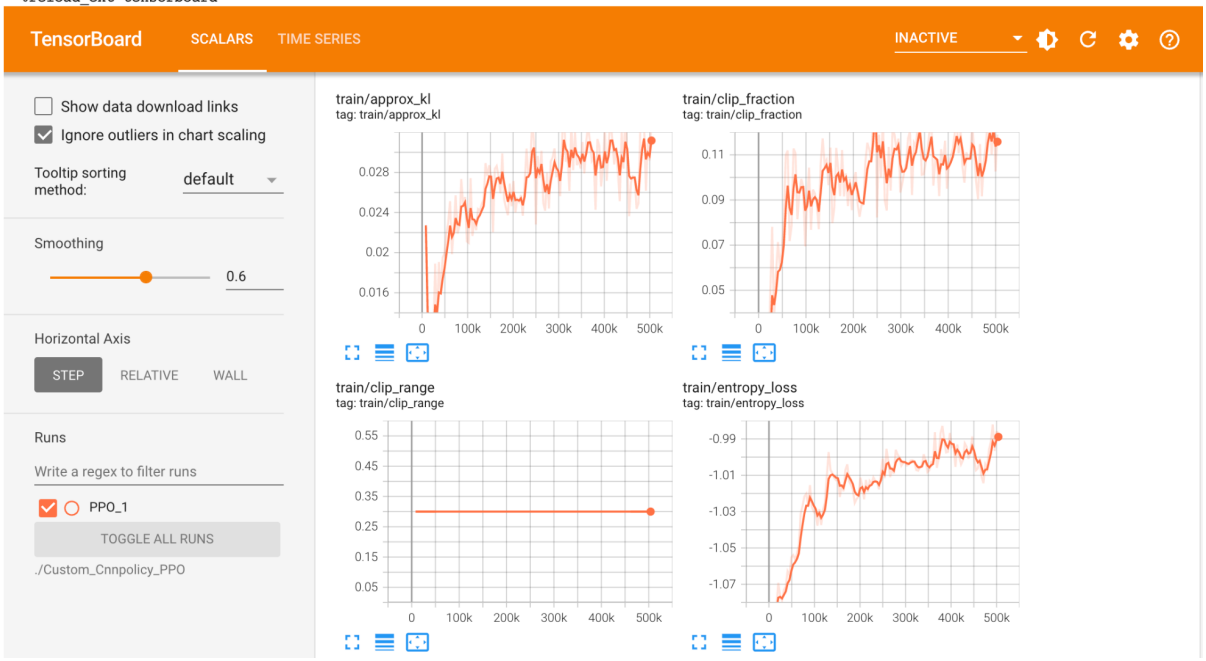




Custom CNNpolicy with PPO:

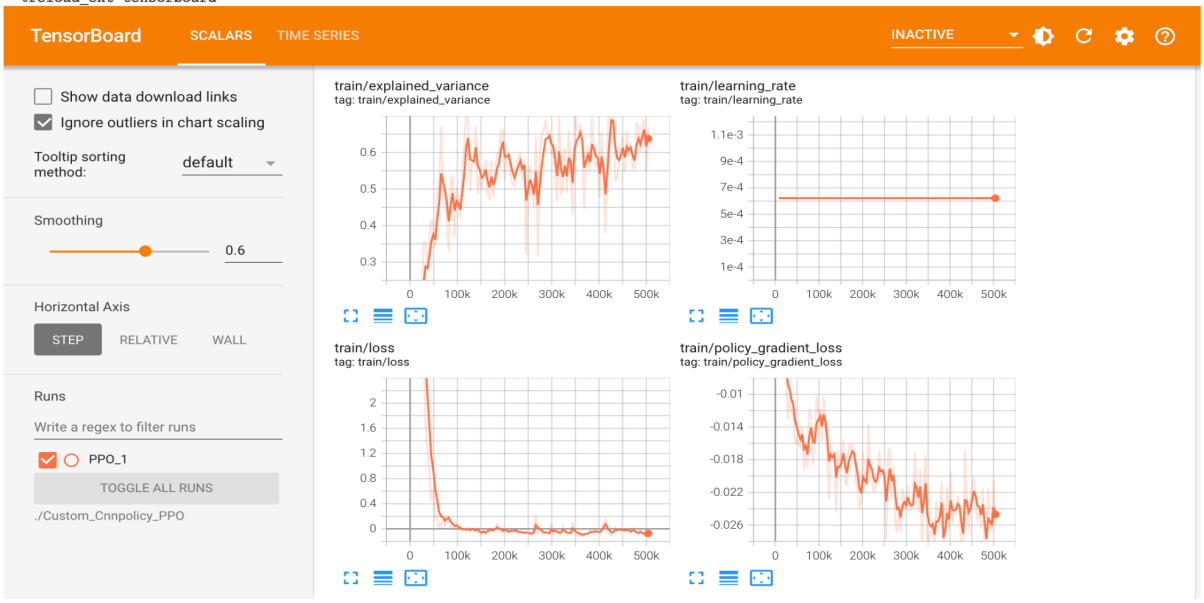
```
%tensorboard --logdir='./Custom_Cnnpolicy_PPO'
```

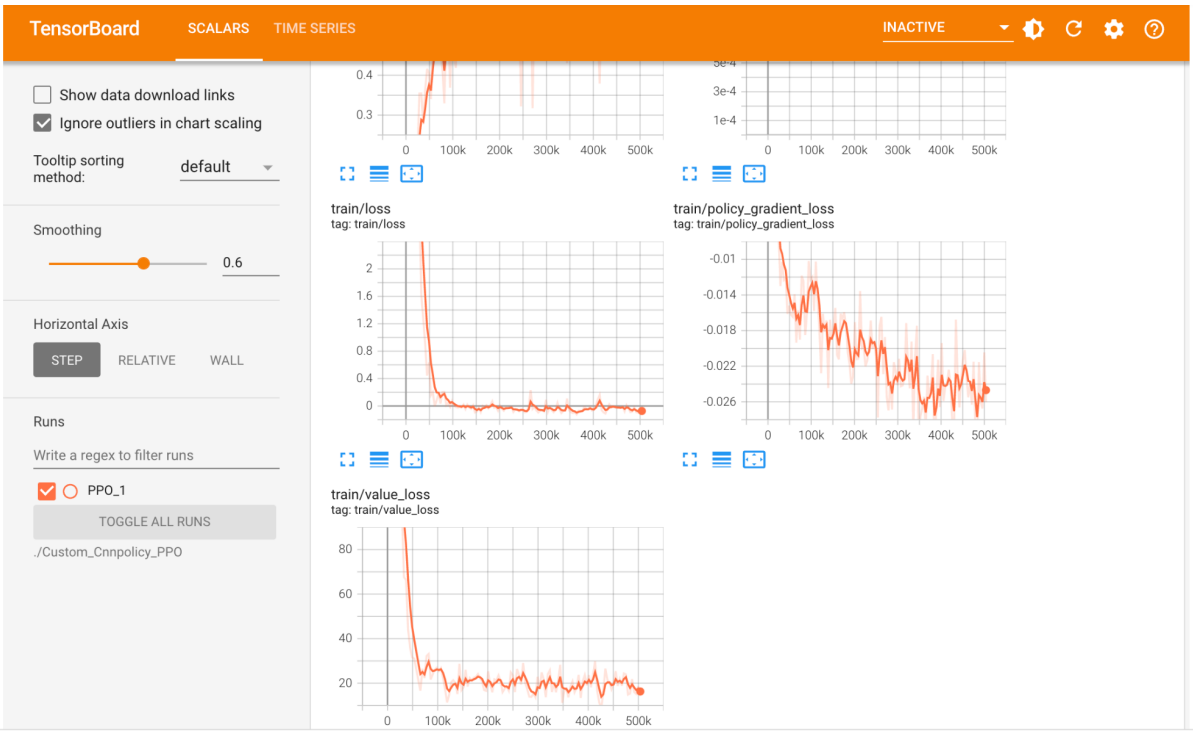
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```



```
[ ] %tensorboard --logdir='./Custom_Cnnpolicy_PPO'
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

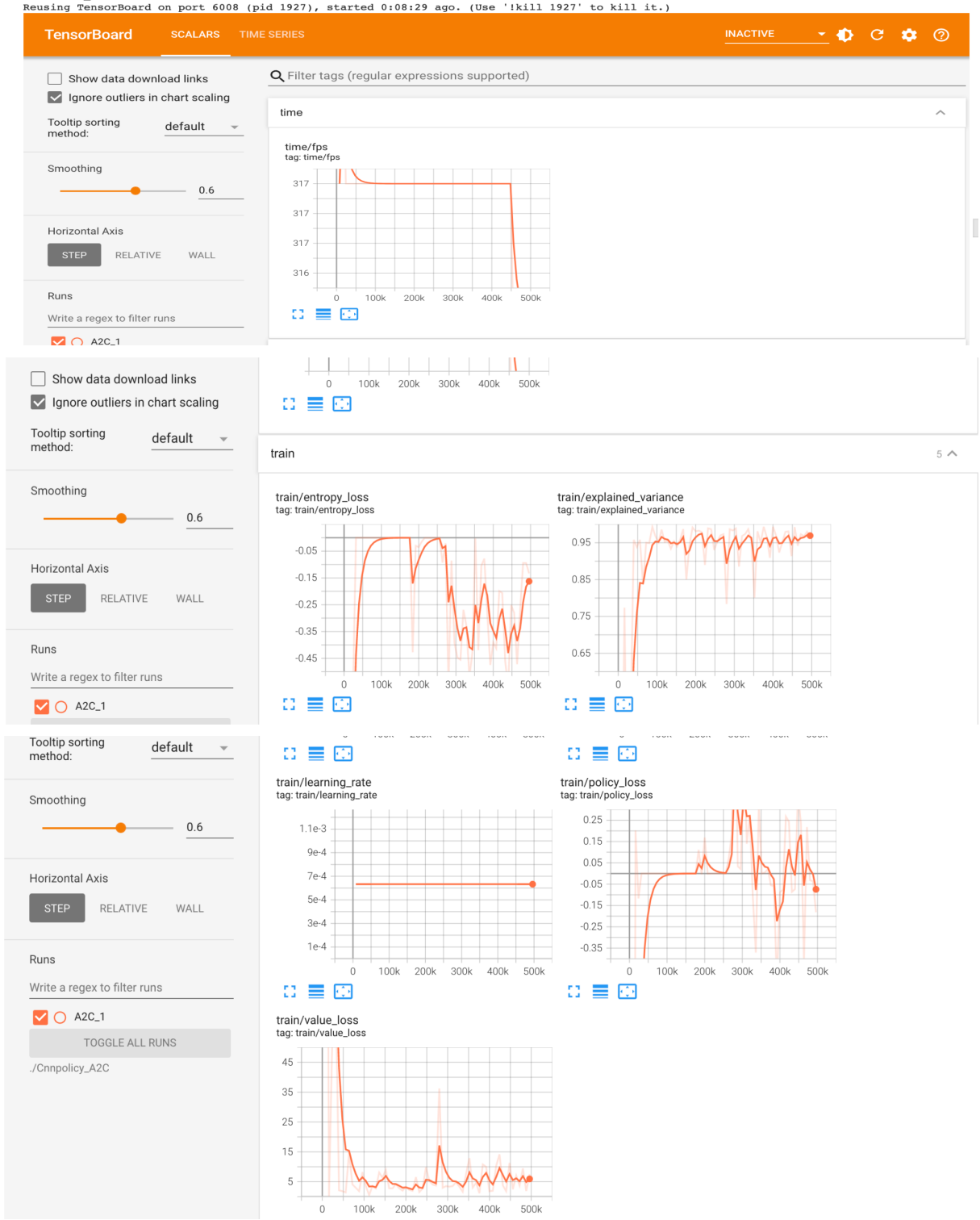




CNNpolicy with A2C :

```
# Load the TensorBoard notebook extension
%load_ext tensorboard

%tensorboard --logdir='./Cnnpolicy_A2C'
```

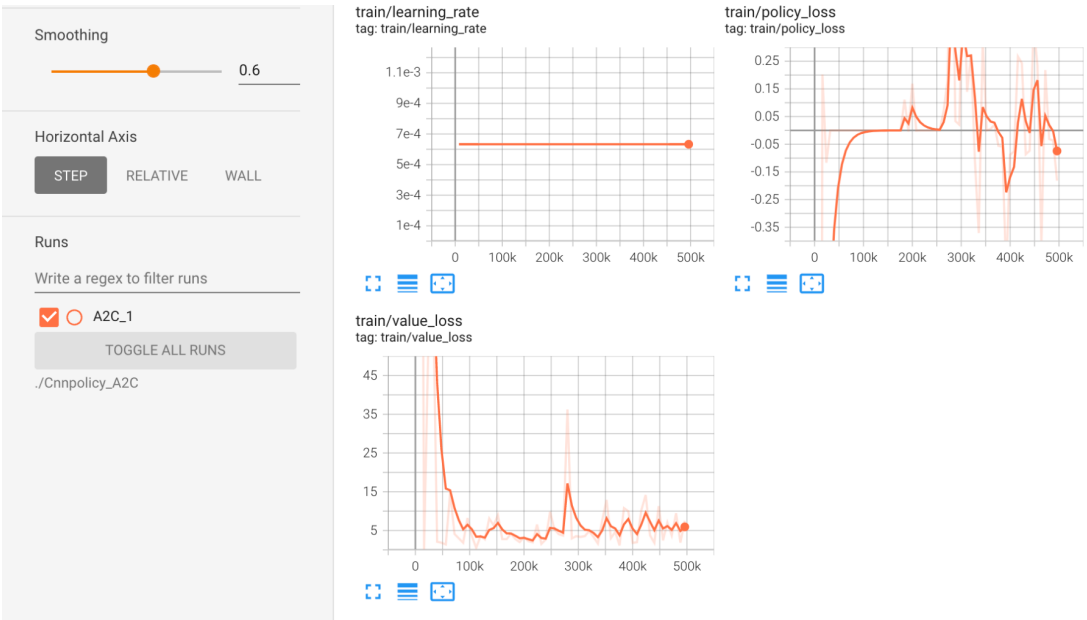
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6008 (pid 1927), started 0:08:29 ago. (Use '!kill 1927' to kill it.)
```



Custom CNNpolicy with A2C :

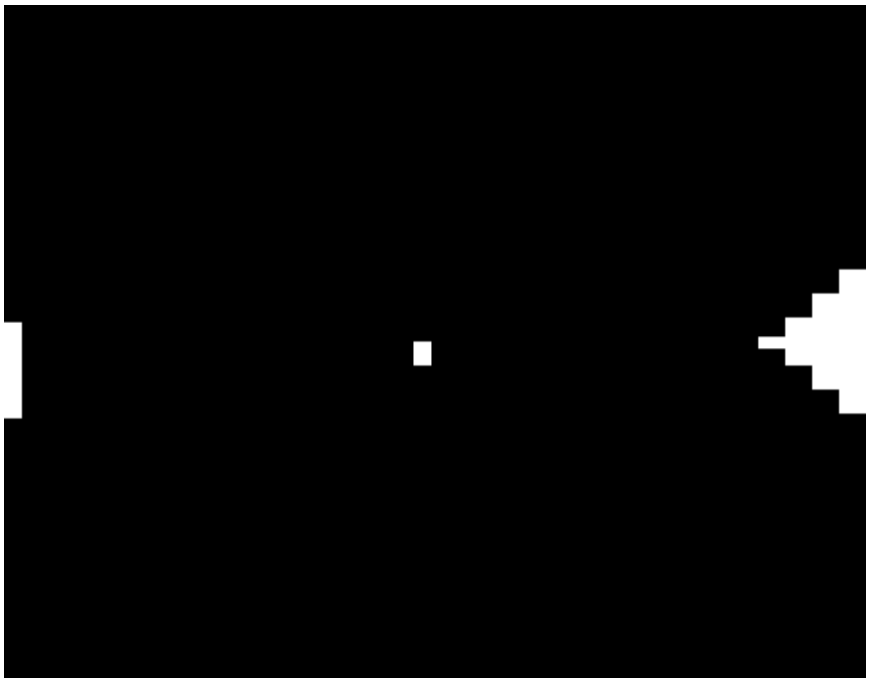
```
# Load the TensorBoard notebook extension
%load_ext tensorboard

%tensorboard --logdir='./Custom_Cnnpolicy_A2C'
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

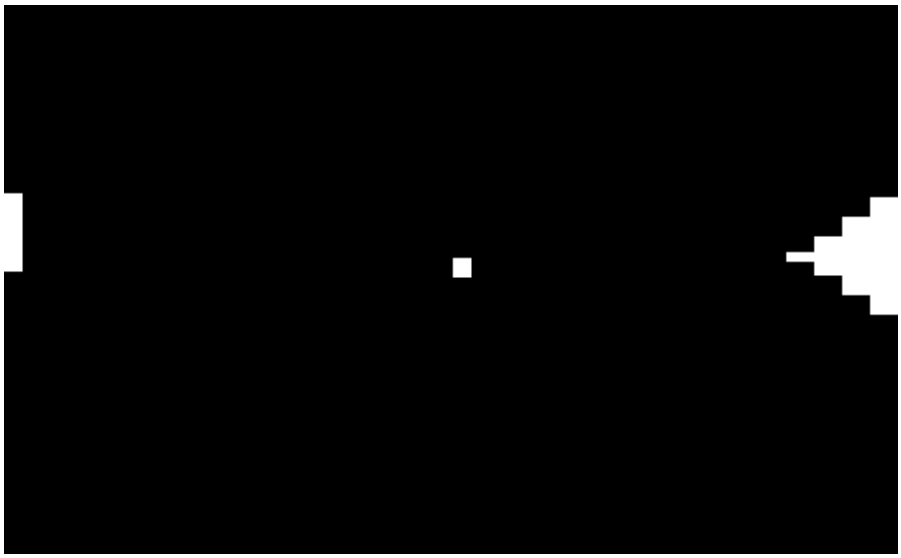


Demo

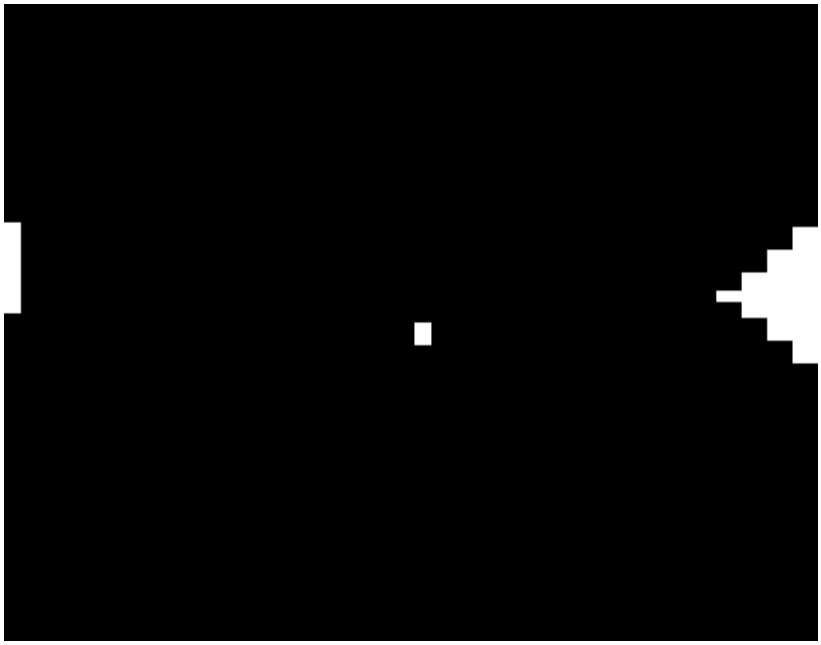
1) **PPO with default CNN - Average Total Reward = 55.11**



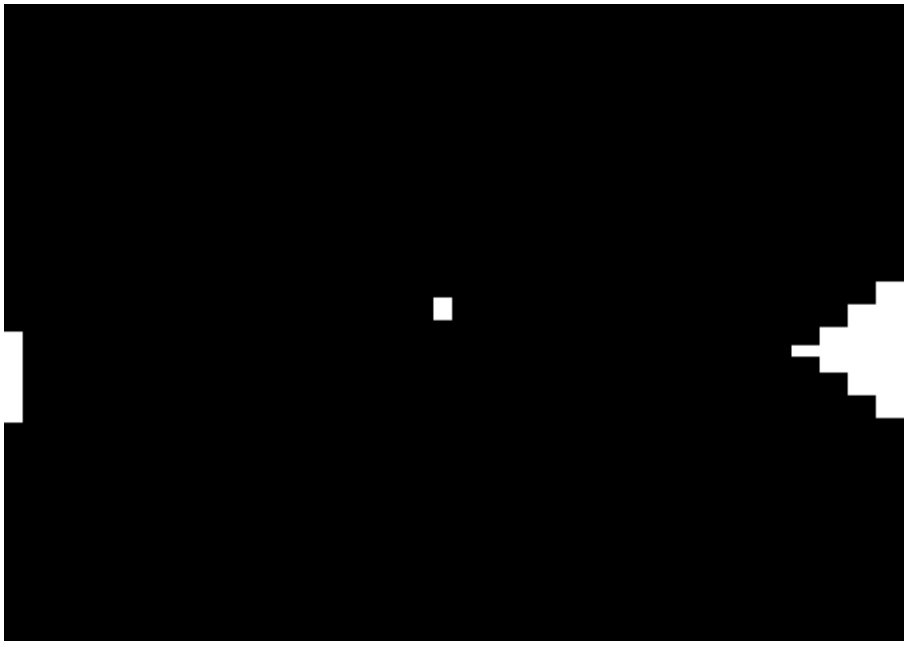
2) **PPO with Custom CNN - Average Total Reward = 33.66**



3) **A2C with default CNN - Average Total Rewards = 24.33**



4) **A2C with Custom CNN - Average Total Reward = 25.66**



Conclusion

In this project, we have demonstrated the use of a multi-agent environment using PettingZoo to train the A2C and PPO models and also created a cnnpolicy for training the model. We learned to create parallel environments to train these multi-agent models. The observation results show that the PPO model performed better than A2C for the specific cooperative ping pong game.

Appendix

Code Link :

https://colab.research.google.com/drive/1Fx3NsdiwNgFAvEqdYiLz6OmPlmGdi6Y3?authuser=1#scrollTo=gXSigF6gqXQy

https://github.com/Teepika-R-M/CMPE-260-Reinforcmnt-Learning

Reference

1.https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

2.https://towardsdatascience.com/multi-agent-deep-reinforcement-learning-in-15-lines-of-code-using-pettingzoo-e0b963c0820b

3.https://www.pettingzoo.ml/butterfly/cooperative_pong

4.https://theaisummer.com/Actor_critics/

5.https://stable-baselines.readthedocs.io/en/master/modules/a2c.html

6 Samsami, Mohammad Reza & Alimadad, Hossein. (2020). Distributed Deep Reinforcement

Learning: An Overview