NYU | COURANT

Cooperative Multi-Agents with Reinforcement Learning

Tanran ZHENG[+] Xinru XU[+]

CSCI 3033-090 Fall 2021 Final Project

Supervised by Prof. Lerrel Pinto

[+] New York University. Equal contribution.

Introduction

Reinforcement Learning (RL) has made significant progress in playing video games. Although a large number of video game agents with RL have achieved super-human performance, most of them excel in single player games only. Multi-Agent Reinforcement Learning (MARL) problem has become a more and more important topic as it has a wide range of applications, including but not limited to multi-robot control, communication in natural language processing context, multiplayer games, etc. As a result, MARL is an on-going, rich field of research.

Among many challenges when traditional reinforcement learning algorithms are facing under the MARL settings, non-stationary environment and exponentially increased action space are the most difficult ones. Because of the involvement of other agents in the MARL problem, the environment is less explainable by the changes made by the agent's own policy. The nonstationarity typically makes learning far more difficult as the policy's knowledge of the environment becomes wrong over time, and as a result most of the multi-agent reinforcement learning algorithms suffer from slow convergence. Moreover, as the number of agents increases, the action space of the problem becomes so big that it makes most of the traditional algorithms less effective.

Therefore, in this final project , we investigate and implement major algorithms for Multi-Agent Reinforcement Learning (MARL) problems, and evaluate their performance in a cooperative and adversarial gaming environment. Moreover, we implement and examine different modifications for these algorithms in order to improve the effectiveness and robustness of their performances.

Related Work

There has been a number of algorithms in the MARL area, and most of them can be categorized into 5 categories: (1)Independent Learners, (2) Parameter Sharing, (3) Fully Observable Critic, (4) Value Function Factorization, and (5)Consensus. In this final project, we will focus on the first three categories.

Independent Learners

One of the simplest approaches is to solve the MARL problem as an extension of the single-agent RL problem, which is called independent learners. First proposed by Tan (Tan, 1993), each agent is trained independently, and the other agents and their actions are considered as part of the environment. More specifically, every agent has its own network that is trained by their local observations (Watkins & Dayan, 1992), and all of them try to maximize the team reward.

Despite it is a straightforward approach, it has several advantages compared to some more complicated algorithms: it can be scaled easily to a more complicated environment with huge observation and action space, and it only needs the local history of observations during the training and the inference time (Oroojlooy & Hajinezhad, 2019).

However, training each agent independently will cause the environment to become non-stationary from the perspective of each individual. This is especially vital for DQN as it prevents the straightforward use of past experience replay, which is crucial for stabilizing deep Q-learning (Lowe & Wu, 2017).

Parameter Sharing

Parameter sharing is a paradigm that uses a single shared policy simultaneously for multiple agents, and this policy is used for every agent to select action. In this way, it effectively bootstraps single-agent RL methods to learn in cooperative environments. Most of the traditional reinforcement learning algorithms, such as Q-learning and policy gradient, can be used as the shared policy network, which gives this paradigm a high degree of freedom in model selection. Some recent studies have also shown that parameter sharing is one of the most effective algorithm paradigms for MARL problems by outperforming more recent algorithms such as MADDPG in many environments (Terry & Grammel, 2020).

Centralized Critic

In order to address the nonstationarity of the MARL problem, Lowe et al. (Lowe & Wu, 2017) proposed a new method in which all critic models observe the same state of the environment, and each agent has its own actor network. This is also referred to as a centralized critic and decentralized actor. The centralized critic is fully observable in a way that it has all the observations from all the agents and the environment. As a result, from the perspective of the centralized critic, the environment is stationary even though the policy of other agents changes. In order to avoid having lazy agents, Sunehag et al. (Sunehag, 2017) later proposed a method that distinguishes the share of each individual agent into the global reward.

Environment

Soccer Twos is one of the multi-agent environments of the ML-Agents Toolkit, developed by Unity. The environment simulates a 2 vs 2 toy soccer game, in which 2 of the agents are considered as teammates, and other 2 agents are considered as adversaries. The goal of the agents in this game is to get the ball into the opponent's goal while preventing the ball from entering its own goal. All agents use the same reward function: +1 point when ball enters opponent's goal, -1 points when ball enters team's goal, +0.02 points when the agent touches the ball, and -0.001 existential penalty as time goes by.

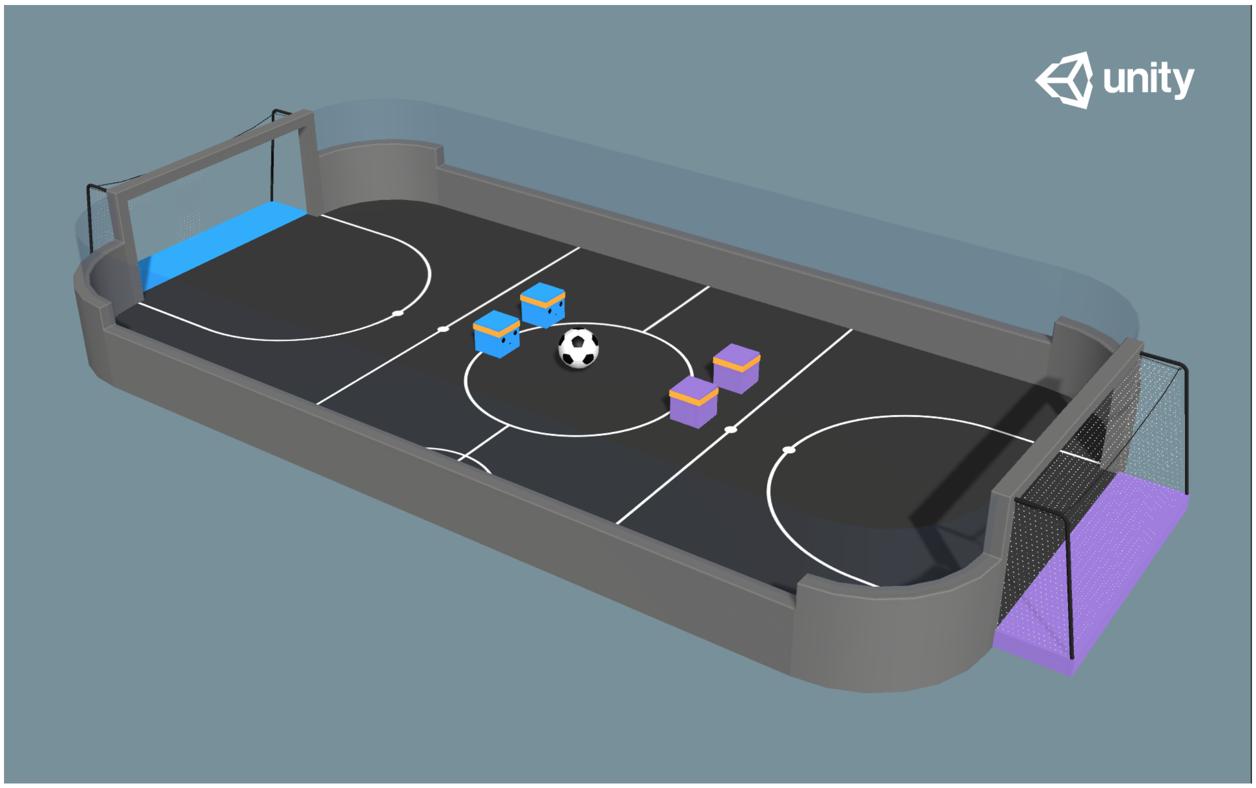


Figure 1. Screenshot of **Soccer Twos** environment

We choose **Soccer Twos** as the environment of our project because some of its features suit our project objectives. Agents of the same team need to cooperate in order to score, so it can test the ability of our agents to perform cooperation. The action of each agent, including that of adversaries, can be fed into the environment independently based on different policies, therefore, during the training and evaluation phase, we can apply different policies to different agents in order to perform different kinds of tests (for example, if we let the adversaries to stay still, this environment becomes a pure cooperative environment, which we will elaborate in later sections). The reward function is reasonable in a way that there is existential penalty for the game time and reward for touching the ball. In this way, the "lazy agent" problem (Sunehag & al., 2017) can be mitigated. Last but not least, we can test not only the ability of our agents to cooperate, but also the ability of our agents under a competitive environment.

Despite these advantages of **Soccer Twos**, it has relatively poor Python API support compared to other popular reinforcement learning environments. The environment simulates 32 agents, 8 games simultaneously, see Figure.2. However, it assigns each agent to each team/court randomly, and there is no way for us to differentiate which 4 agents are in the same court. Thus, one of the biggest challenges we faced during our project is to figure out which four players are on the same court so that we can correctly implement the models with shared information. In other words, we need to figure out which two players should be cooperating with each other, otherwise, the global environment (especially for MADDPG where the centralized critic has the global observation) becomes non-stationary from the perspective of each agent.

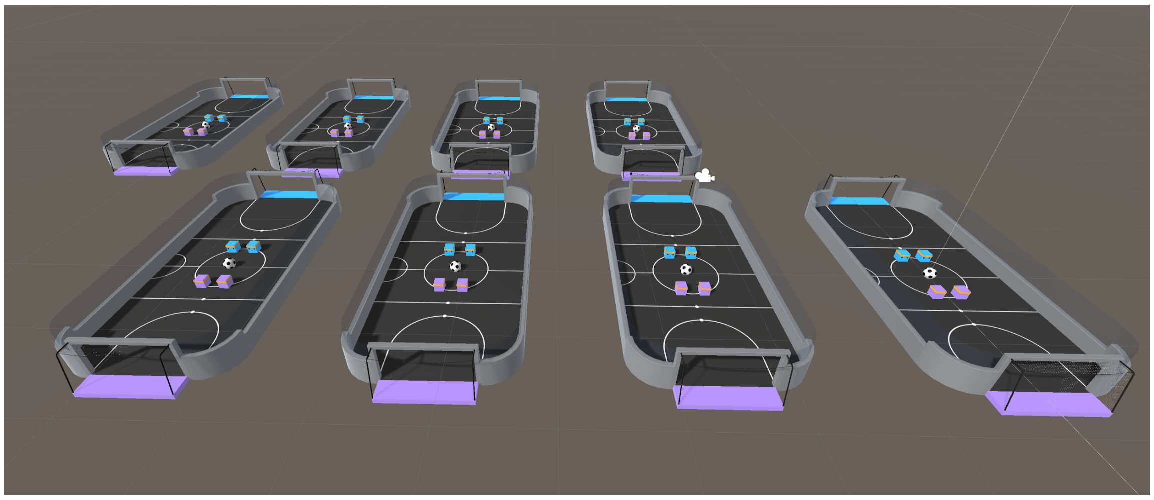


Figure.2 Screenshot of **Soccer Twos** environment in Python API

We tried to solve this problem by building program with both C# and Python that allows us to get information from the underlying objects of this environment. Unfortunately, due to our lackness of experience of C#, we couldn't solve some minor bugs and did not add this part to our final code. Instead, we implemented a rather naive method in Python, that we first run the environment for a few hundreds of steps (we call it the warm-up phase), iterate through the observations of all agents, and determine the assignment of all agents by comparing the changes of their observations.

Methods

In this project, we focus on three common categories of the MARL paradigms: independent learners, parameter sharing, and centralized critic.

Independent Learners - DQN

We choose DQN as the underlying model of the independent learners paradigm, in which each player is trained with a single DQN model, and use it as the baseline of our experiment. During the training time, each model will use local observation and local action, i.e. the observation and action of its corresponding agent, and the goal is to maximize the team reward. Prioritized Replay Buffer, double Q-learning and dueling Q-learning are used in our experiment.

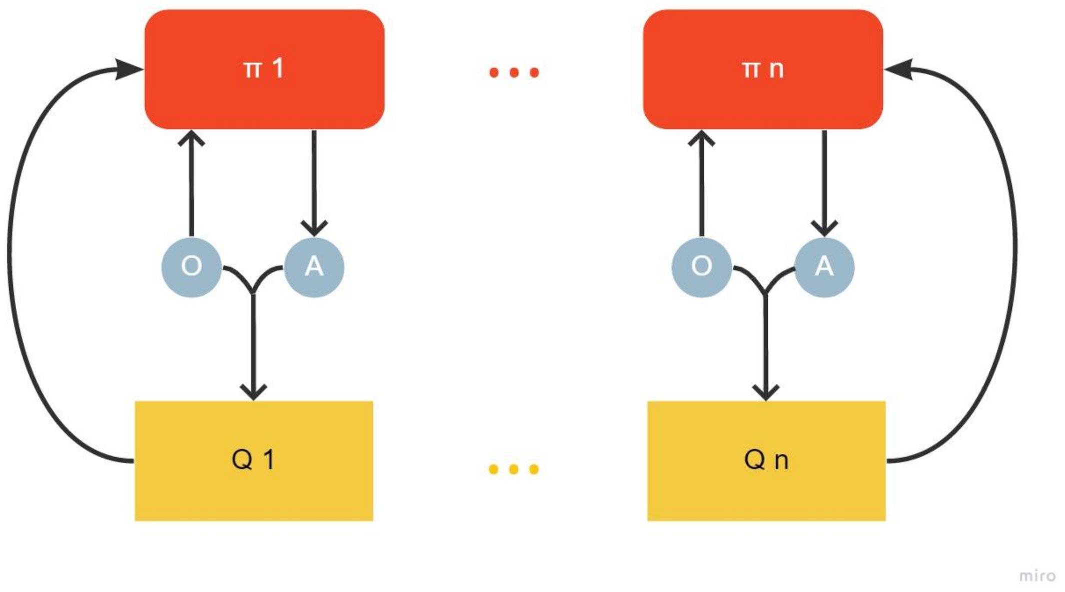


Figure.3 Architecture of Independent Learners paradigm

Parameter Sharing - PPO

For parameter sharing, we choose to use the proximal policy optimization (PPO) model as the only policy network shared between all agents, so that all agents use this policy network to pick an action. Each agent will train this shared network using its local observation and local action, and their common goal is to maximize the team reward.

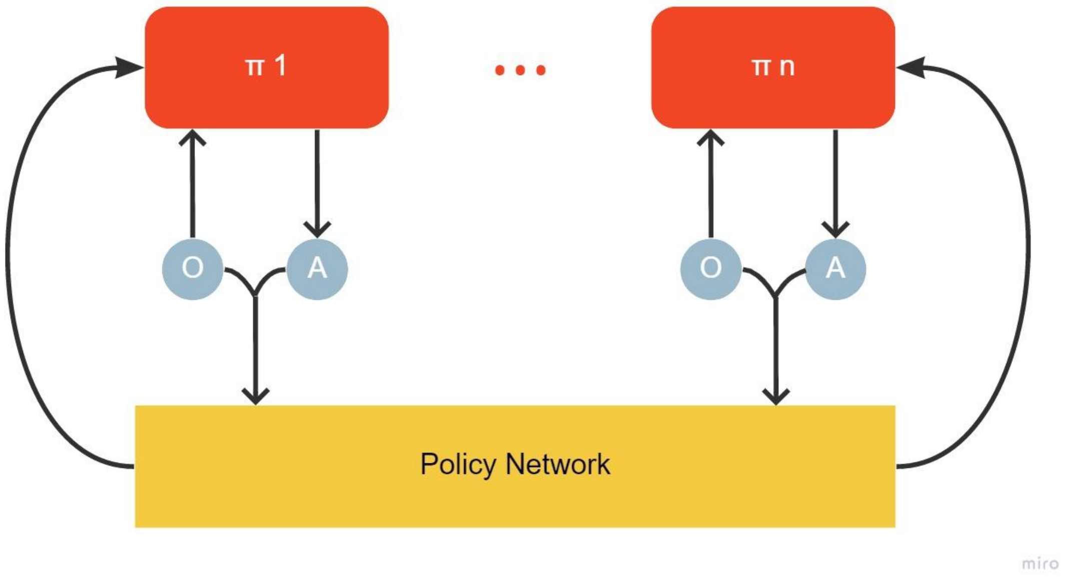


Figure.4 Architecture of Parameter Sharing paradigm

Centralized Critic - MADDPG

We implement the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) model, proposed by Lowe el. of OpenAI, as the representative of the centralized critic paradigm. It is an extension of Deep Deterministic Policy Gradient (DDPG) to the multi-agent environment, and its main architecture is shown in Figure 5.

In MADDPG, each agent has its own actor network which uses local observations for deterministic actions, and a target actor network with the same functionality in order to stabilize the training process. Each agent also trains a critic network that is allowed to access the observations, actions and the target policies of all agents in the training time. Then, the state-actions of all agents are concatenated as the input of each critic network, which will use the individual reward to obtain the corresponding Q-value. (Oroojlooy & Hajinezhad, 2019) During the execution time, however, the centralized critic network is removed. In this way, each agent will learn its own policy with the guidance of the critic network. (Lowe & Wu, 2017) The critic networks are trained by minimizing the loss function:

$$L(\mu_i) = \mathbb{E}_{\boldsymbol{o}^t,a,r,\boldsymbol{o}^{t+1}} \left[\left(Q_i(\boldsymbol{s}^t, a_1^t, \ldots, a_N^t; \mu_i) - y\right)^2 \right],$$
$$y = r_i^t + \gamma Q_i \left(\boldsymbol{o}^t, \bar{a}_1^{t+1}, \ldots, \bar{a}_N^{t+1}; \bar{\mu}_i\right) \Big|_{\bar{a}_j^{t+1}=\bar{\pi}(o_j^{t+1})},$$

in which o^t is observation of all agents, $\overline{\boldsymbol{\pi}}_j$ is the target policy, and $\overline{\mu}$ is the target critic (Lowe & Wu, 2017). Therefore, each critic deals with a stationary environment, and during the execution time, it only needs the local information.

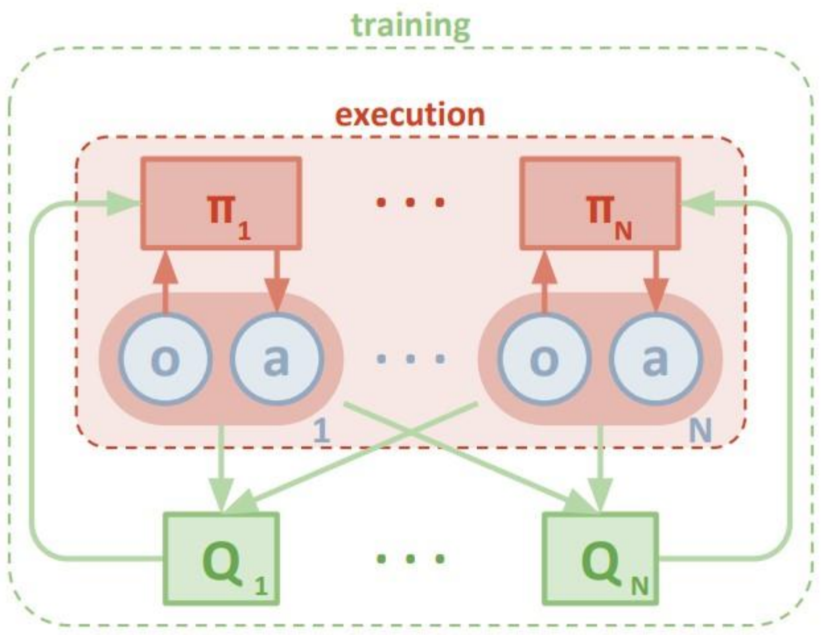


Figure.5 Architecture of MADDPG (Lowe & Wu, 2017)

Besides the typical features of the centralized critic paradigm, MADDPG also has two additional features in order to improve the robustness of agents in the multi-agent environment. Since each critic has global observations, the above-mentioned architecture assumes that each agent has the information of the other agent's policies. To remove this assumption, the original paper proposed that each agent can additionally maintain an approximation to the true policy of each other agent, which can be learned by maximizing the log probability of the other agent's action. This is called Policy Inference. Moreover, the original paper proposed a method called Policy Ensembles, which means each agent will train a collection of K different sub-policies, and at each episode, each agent will randomly select one particular sub-policy to execute. The goal of Policy Ensembles is to prevent overfitting problems, meaning the agents can derive a strong policy by overfitting to the behavior of their competitors in competitive settings. (Lowe & Wu, 2017)

Improvement of MADDPG - Separated critic input

To further improve the MADDPG model, we propose a method that separates the input of the critic network of MADDPG. In the original model, all agents' state and action are concatenated into one vector and are fed to the critic network at the same time. Even though the architecture tries to prevent the non-stationary problem by design, the environment is still non-stationary. In order to further mitigate the non-stationary problem, we instead divide the input into separated vectors: the first vector includes the state of all agents and actions of all other agents except the acting agent; the second vector includes the action of acting agent only. During the training time, the first vector will be the input of the first NN layer of the critic network, and the second vector will only be fed to the critic network after the first NN layer generates its output. The first vector will be concatenated with the output of the first NN layers. This will stabilize the agent with all other parameters kept the same.

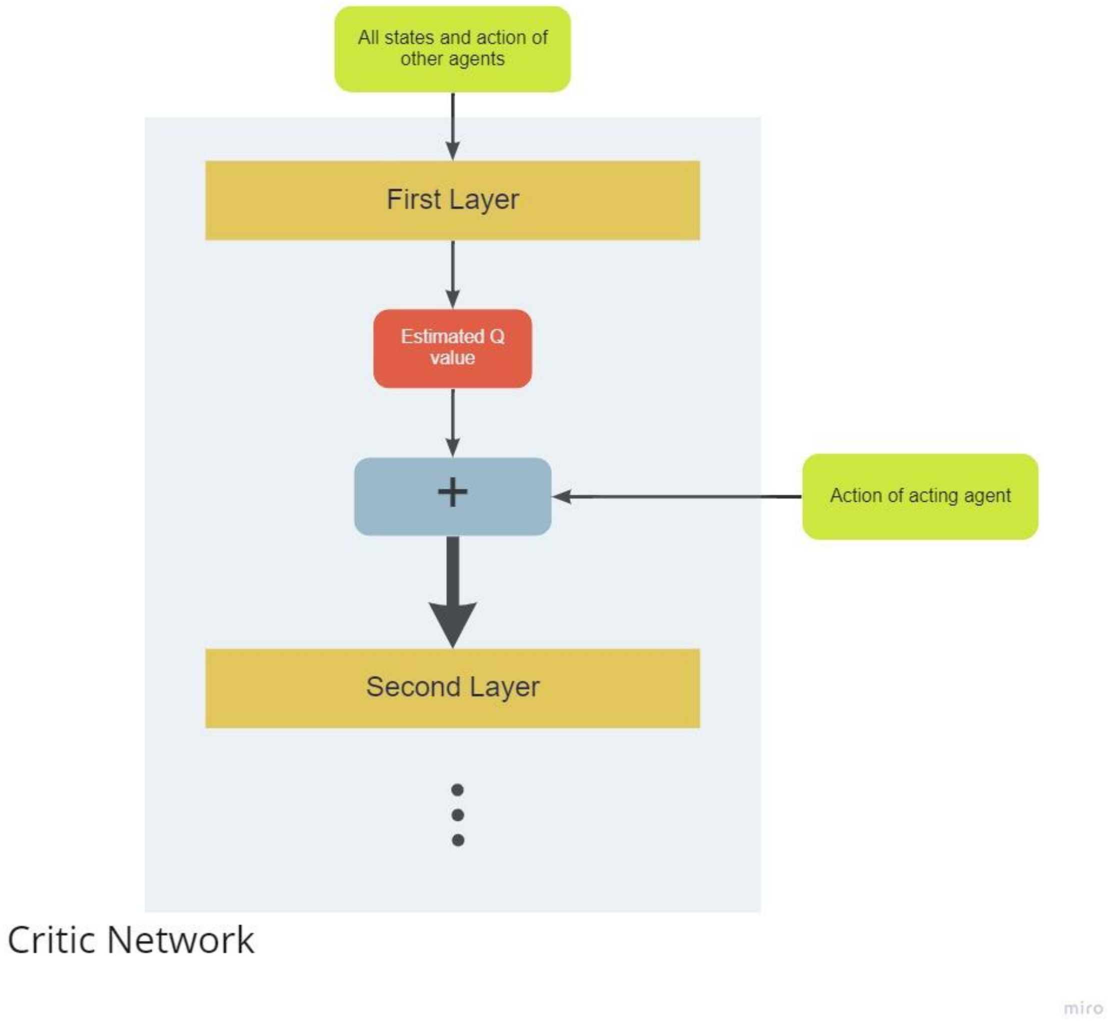


Figure.6 Demonstration of separated critic input

Self-play

In the first round of the implementation, we train these models in a purely cooperative environment by setting the adversary agents to stay still during both the training phase and evaluation phase. In this setting, the environment is purely cooperative so that we can focus on whether these models are able to improve the performance of agents in a cooperative context. Moreover, we allow the agents to focus on basic objectives of the game, such as locating the ball and the goal, kicking the ball, and scoring as soon as possible. The modified cooperative environment, in this case, is more stationary without the action of the adversary agents, especially for MADDPG, as all the changes of the environment are consequences of the actions of the agents.

In the later round of our experiments, we added the adversaries back to the environment and used self-play during the training and evaluation. During the training time, we used a past snapshot of the current agent's policy as the policy of all adversaries embedded in the environment. During the evaluation time, both the adversaries and the agents will use the same policy trained and play against each other. In this way, the agents are effectively training against opponents of increasing strength; as a result, the self-play allows us to bootstrap an environment to train competitive agents in a competitive environment. (Bansa, 2017) (Zhong et al., 2020)

However, there are practical issues that arise from the self-play framework (Cohen, 2020). The trained agents are more prone to overfit because they are trained to defeat a particular playstyle; self-play also makes the environment more non-stationary as the opponents, which are treated as part of the environment in some cases, are evolving during the training process. Therefore, to address the first problem, we store multiple past policies of the agents and choose one of them as the policy of adversaries at each given time. In this way, the agent will see a more diverse set of opponents during the training. Furthermore, to address the second problem, we fix the policy of the opponent for a longer training duration, so that the learning environment will be more consistent during the training process.

Experiment and Results

Naive Models

In this round of our experiment, we train the naive version of the models without using self-play. More specifically, the opponents are set to stay still during both the training process and evaluation process (i.e. in the purely cooperative setting).

DQN

As our baseline model, we train the independent learners paradigm with DQN. As the designated baseline algorithm, we expect that it will struggle to solve this task, and it does, as shown in the figure below.

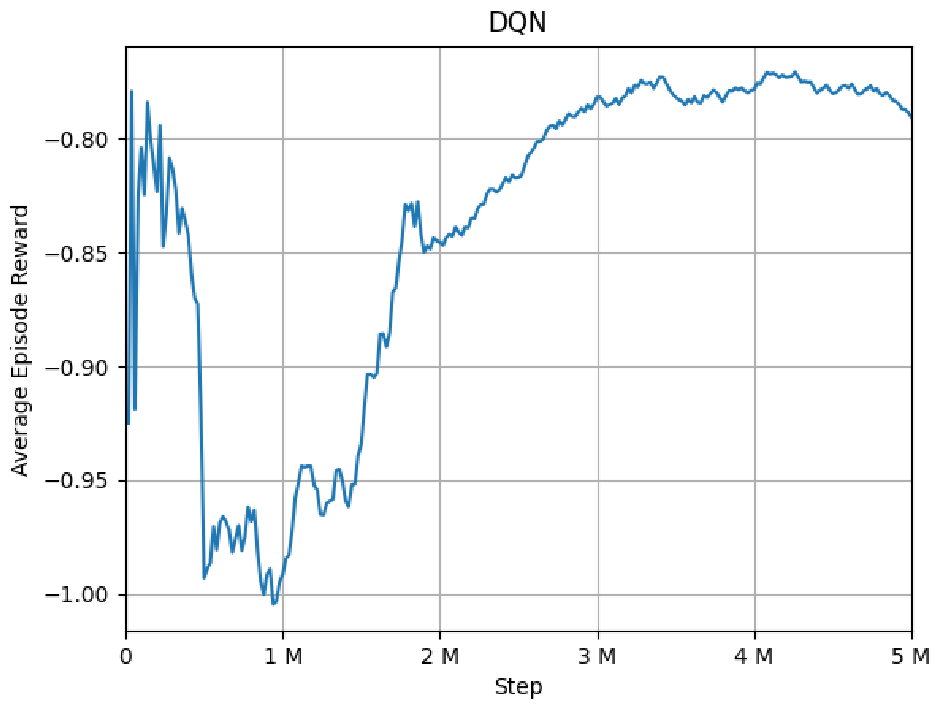


Figure.7 Result of DQN

We can see that DQN performs poorly in this environment. The average episode reward is around -0.78 even after the model is converged. This means that the agents not only fail to score in the game, they also waste a lot of time wandering around, as a result they don't get the reward of touching the ball and get a lot of time penalties (even own-goals). This might be because the Independent Learner paradigm does not work in this complex environment, and the DQN algorithm does not work as well because the non-stationarity makes the experience buffer unreliable.

PPO

From Figure.8 we can see that the naive PPO performs well in the purely cooperative settings. Due to the complexity of the environment, the naive PPO takes longer training steps to converge, but at the end it achieves 0.35 average evaluation episode reward.

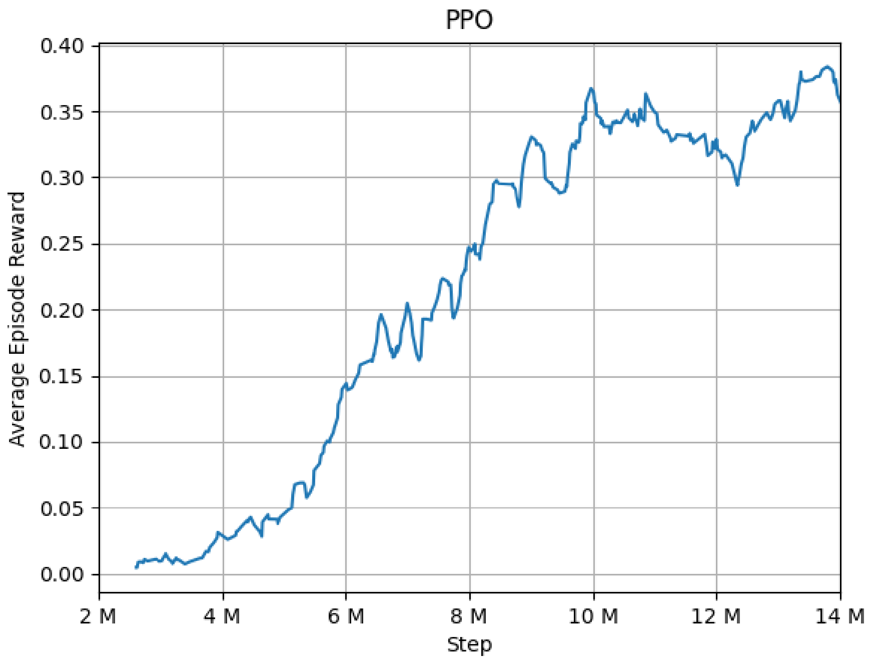


Figure.8 Result of naive PPO without self-play

However, this result seems too good and we suspected it is overfitting. Therefore, we run an extra experiment on this trained model, in which we evaluate this model in a competitive setting (play against itself). It turns out that the average episode reward becomes very poor, as shown in Figure. 9. This is because in the purely cooperative setting, even though the naive PPO is able to learn how to score, it overfits. During the training time, the agents can easily learn how to locate the ball, kick the ball along a specific path, circumvent the adversaries, and score; but if facing active adversaries, the agents fail to perform the correct action, as the environment is very different from the training environment. To address this problem, we use self-play to train the model in the later round of the experiment.

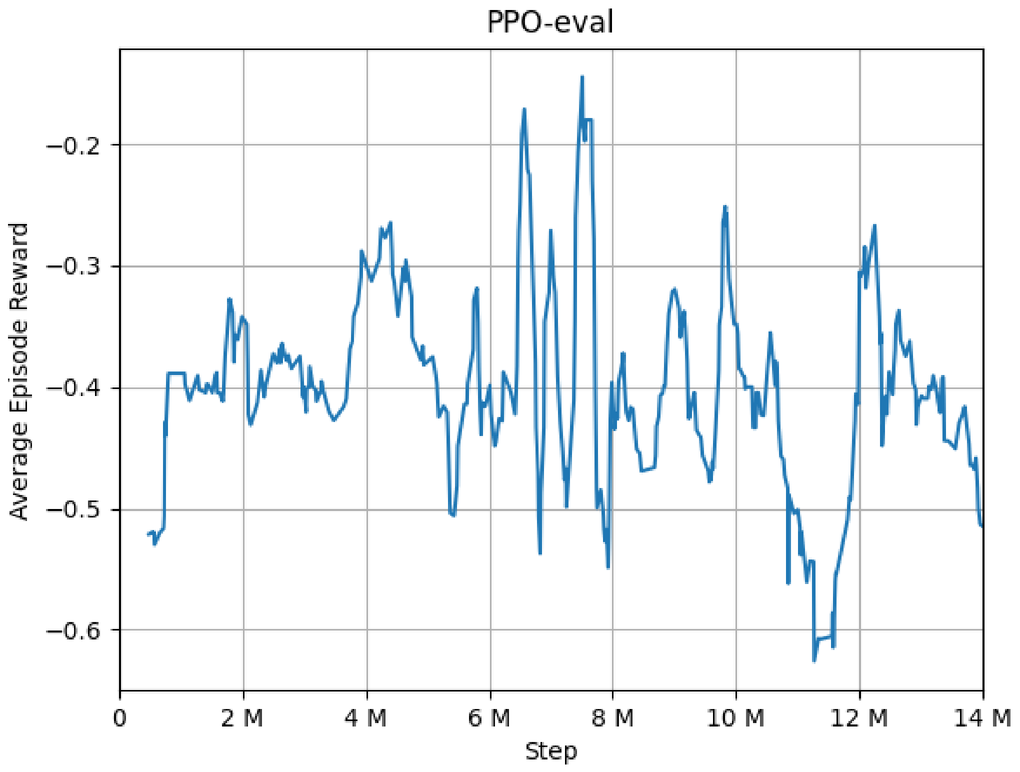


Figure.9 Result of evaluating naive PPO by playing against itself

MADDPG

When we conduct the experiment of MADDPG, we discard the Policy Ensembles feature, which was proposed by the original paper. This is because comparing to the environment used in the original paper, **Soccer Two** is far more complicated. As a result the corresponding networks for MADDPG is a lot more complicated as well, in terms of the dimension of inputs. Therefore, the training time of MADDPG is very long, which takes about 70 hours on a V100 GPU on Greene. Thus, we discard the Policy Ensembles feature in order to shorten the training time to a practical range.

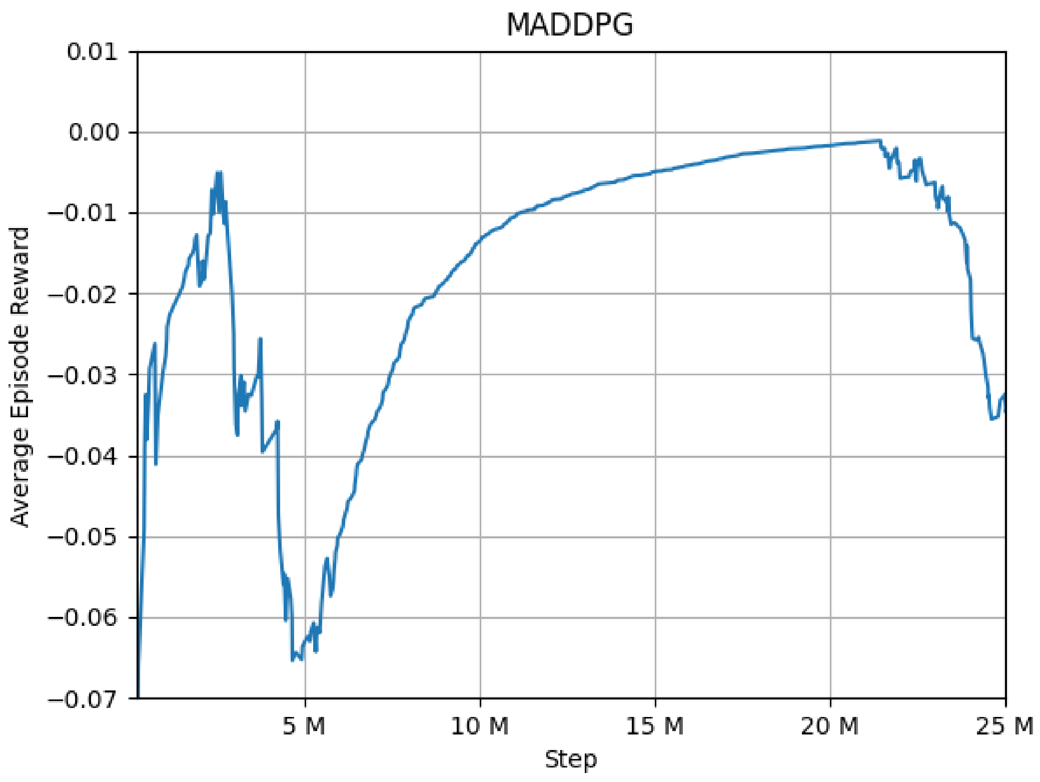


Figure.10 Result of naive MADDPG without self-play

From Figure.10, we can see that the naive MADDPG performs poorly, even in the purely cooperative settings. *(IMPORTANT NOTES: In the final presentation, we incorrectly claimed that the final demonstration video is the result of naive MADDPG by accident. In fact, that demo is the result of improved MADDPG, which will be discussed in a later section)*. After investigation, we found out that Policy Inference does not perform as expected in this environment. The inference is generally inaccurate because we observed that the inferring loss is high and fails to converge during the whole training process. The reason might be that our environment is far more complicated in terms of the observation space and action space compared to the environment used in the original paper. Therefore, the behavior of other agents is harder to predict. Moreover, due to the complexity of the environment, MADDPG needs far longer time to converge, as the scale of the centralized critic network increased dramatically; thus, an inaccurate inference of other agents has severe impact on the stability of the training process. Thus, we remove the Policy Inference feature from MADDPG in later stages of our project.

Self-paly

To improve the performances of our models, we train our models using self-play in the second round of experiment. More specifically, the trained models are saved every 20k steps, and a total of 10 policies will be stored at the same time. The adversaries during the training time will use one of these 10 past policies with a probability of 0.5, as a result we have a fully cooperative and competitive setting for the training time. And in order to mitigate the non-stationary effects, we extend the number of training steps for each adversary policy to 10k steps.

To examine the impact of self-play, we run an experiment using PPO, in which we train PPO with self-play and compare the result with that of the naive PPO. The results are shown in the figure below.

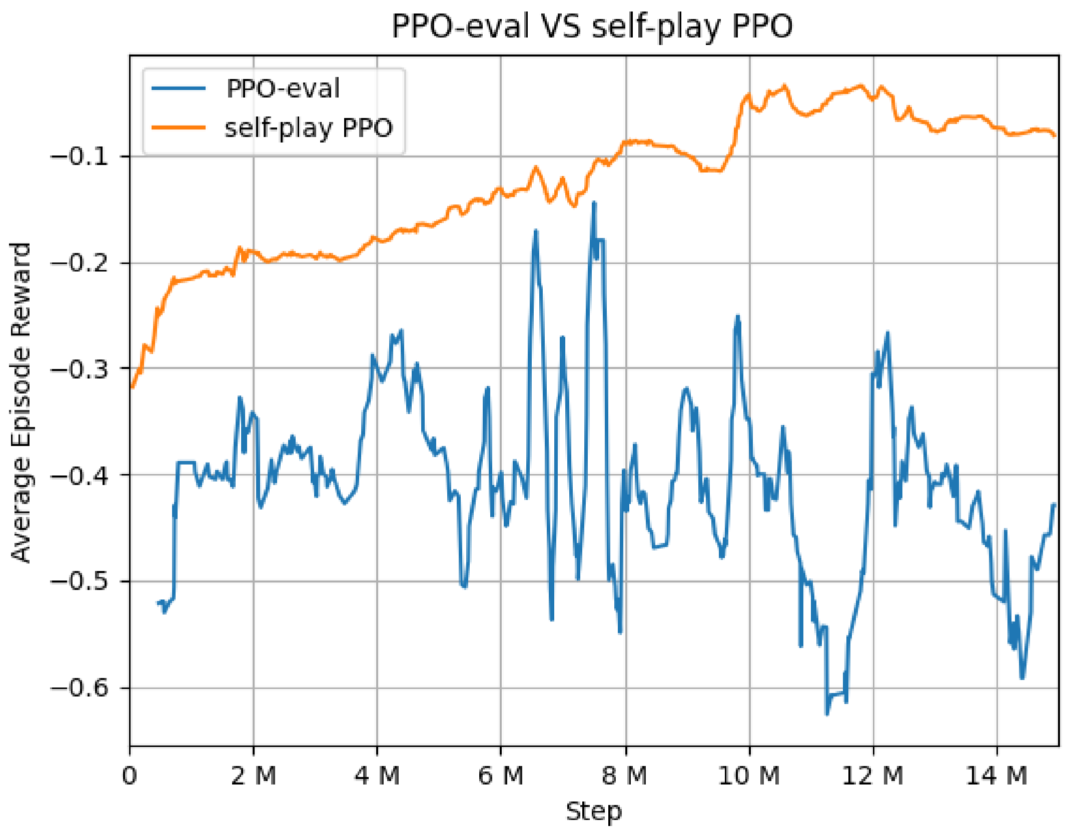


Figure.11 Naive PPO v.s. Self-play PPO

We can see that with self-play, the performance of PPO increased evidently.

Improved MADDPG

Based on the poor performance of the naive MADDPG, we decided to make further modifications to MADDPG so that it will adapt to the **Soccer Two** environment better. As mentioned in the previous part, we first discard the Policy Inference feature from the naive MADDPG as the inference is very inaccurate in our environment. Then we implement the separated input, as described in the Methods section. Last but not least, we use self-play during the training. And the comparison of the results are shown in the following figure.

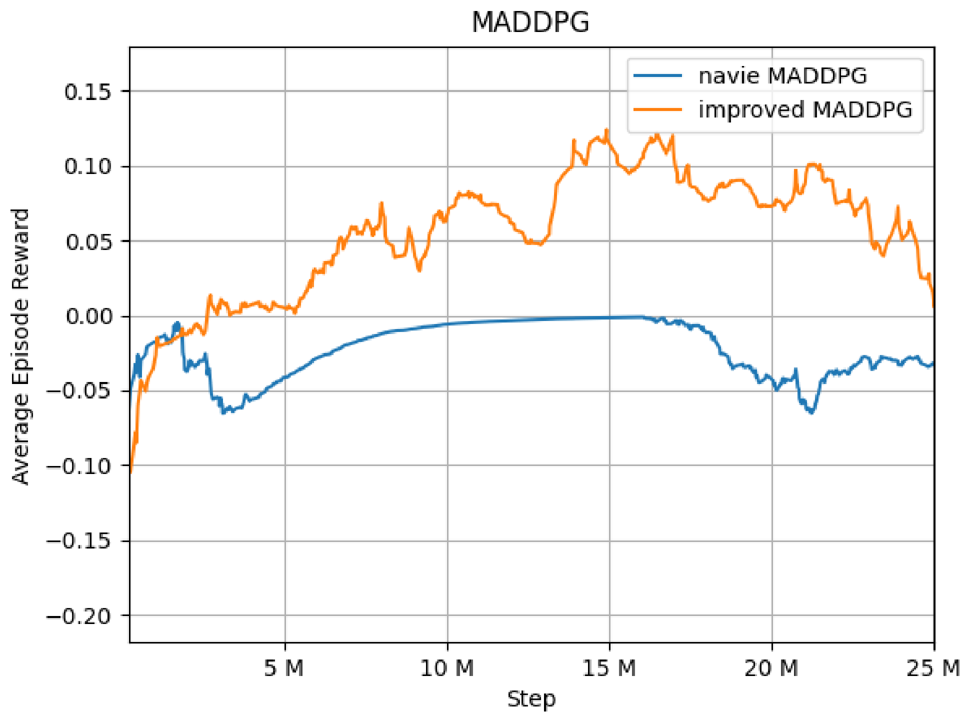
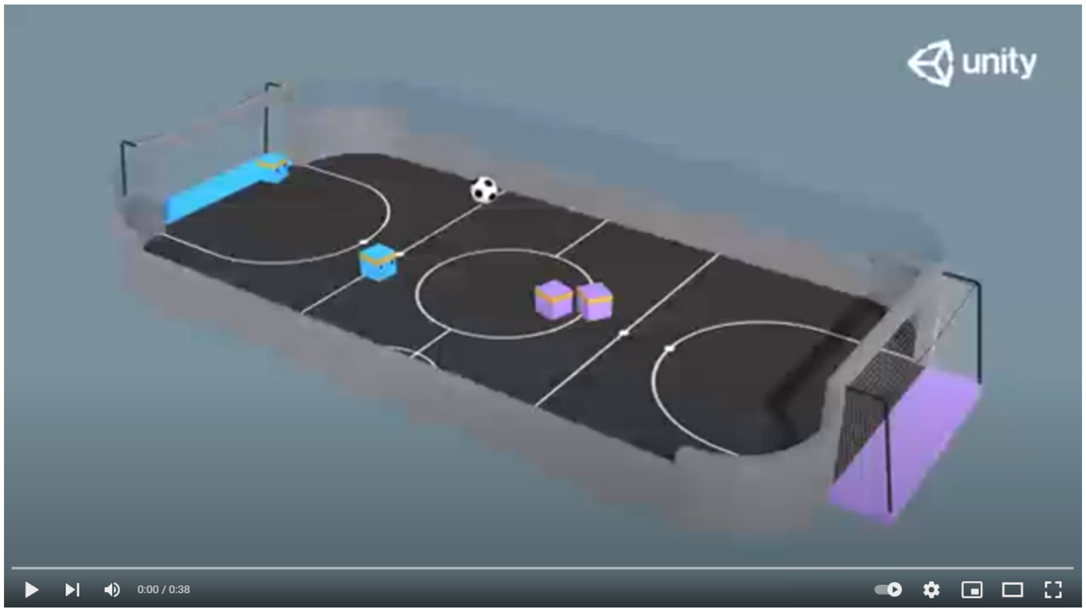


Figure.12 Naive MADDPG v.s. Improved MADDPG

As we can see, the improved MADDPG performs a lot better than the naive MADDPG. One reason is that applying self-play during the training helps the model to learn in a competitive setting. The other reason is that the improved version MADDPG is more stable, despite the fact that self-play is expected to add non-stationarity to the model. Therefore, by adding the separated critic input and discarding the Policy Inferences, the modification makes the MADDPG more stable in this complex environment, which outweighs the non-stationarity caused by the self-play.



Video.1 Improved MADDPG Gameplay Demo

(In case the video does not play, use this URL: https://youtu.be/w4O2FB2xyZE)

Match Day!

Finally, as opposed to the first round of experiment, we decided to evaluate these trained models under a more competitive environment, by letting them play against each other. Another important goal in this step is to check if these models are overfitted by self-play training. We chose the PPO with self-play and improved MADDPG as the two teams to fight against each other, and the result is shown in the following figure.

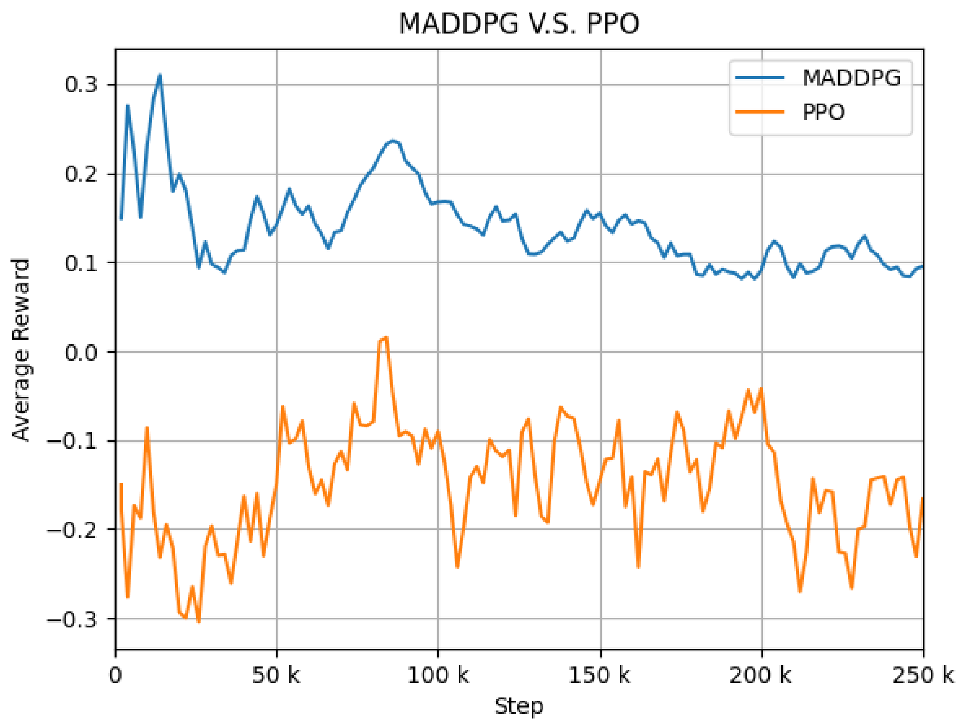
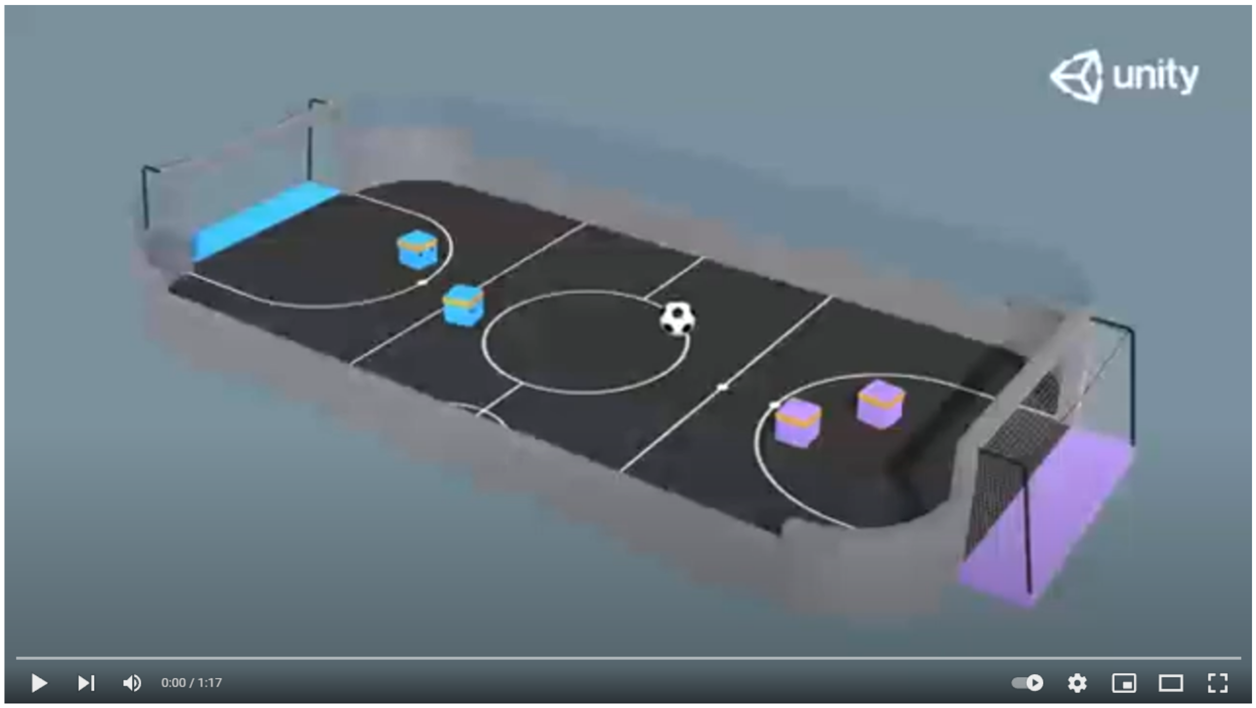


Figure.13 Match day: Self-play PPO v.s. Improved MADDPG

As we can see, the improved version of MADDPG performs better than the Self-play PPO evidently. However, both of them perform worse than they do during the training time. This is because during the training time, the playing styles of their opponents are relatively similar due to the use of self-play. As a result, both of these models are slightly overfitted. When evaluating them by playing against a policy that has not been seen in the training time, both of them perform worse. From the demo video shown below we can see that the improved version of MADDPG can win the game by scoring goal in most of the time.



Video.2 Match Day: Improved MADDPG v.s. Self-play PPO Gameplay Demo

(In case the video does not play, use this URL: https://youtu.be/CbHlplte5QM)

Conclusion

In this final project, we implemented and examined three most common multi-agent reinforcement learning paradigms in a complex cooperative and competitive environment. Moreover, we analyzed multiple features of MARL models, including but not limited to Policy Inferences and Self-play.

We have shown that Independent Learners perform weakly in a complex environment; Parameter Sharing performs well, even the logic behind the paradigm is relatively simple; the naive MADDPG model performs poorly because it fails to mitigate the non-stationarity of the problem, even worse, its relatively complicated architecture compromises the training stability. After discarding Policy Inferences, adding separated critic input, and using self-play, we managed to improve the performance of the MADDPG, which is the paradigm with best result in our experiment.

Future Work

One of the original objectives of our project is to improve the performances of the MARL models by modifying the reward functions. In this way, each agent can either have a different focus (i.e. play a different role in the team), or be rewarded more fairly which prevents the “lazy agent” issue. However, due to the limited support of the Python API of Unity, we were not able to finish it in this project. We would like to explore this topic further in the future.

Another algorithm we would like to explore is Counterfactual Multi-Agent Policy Gradient (COMA). (Foerster, 2017) It proposed a method to evaluate the impact of the action of an individual agent, and use it to measure if the acting agent is really doing the job. This is also known as the credit assignment issue.

Transformer has achieved great success in NLP and computer vision area. We would also like to implement a model using a transformer to see if it improves the result of MARL.

Appendix I: Source Code

Our source code of this project has been uploaded to Github. Please visit:

https://github.com/DariaXu/unity-soccorTwos

https://github.com/zhengtr/soccerTwos_MADDPG

Appendix II

References

Bansa, T. (2017). Emergent Complexity via Multi-Agent Competition. *CoRR*, *abs/1710.03748*.

Cohen, A. (2020, February 28). *Training intelligent adversaries using self-play with ML-Agents*. Unity Blog. Retrieved December 17, 2021, from https://blog.unity.com/technology/training-intelligent-adversaries-using-self-play-with-ml-agents

Foerster, J. (2017). Counterfactual Multi-Agent Policy Gradients. *CoRR*, *abs/1705.08926*. arXiv.

Lowe, R., & Wu, Y. (2017). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments.

Oroojlooy, A., & Hajinezhad, D. (2019). A review of cooperative multi-agent deep reinforcement learning. *CoRR*, *abs/1908.03963*.

Sunehag, P., & al., e. (2017). Value-Decomposition Networks For Cooperative Multi-Agent Learning. *1706.05296*.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. *Proceeding of the tenth international conference on machine learning*, 330-337.

Terry, J. K., & Grammel, N. (2020). Parameter Sharing is surprisingly useful for Multi-Agent Deep Reinforcement Learning. *CoRR*, *abs/2005.13625*.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*, 279-292.

Zhong, Y., Zhou, Y., & Peng, J. (2020). Efficient Competitive Self-Play Policy Optimization. *Corr*, *abs/2009.06086*. arXiv.