Interview Take Home Assignment Write-up

Applicant: Charles Yoo Submitted March 13, 2023

1 Introduction

For this assignment I created, the multi-agent environment that was specified in the problem statement. The remainder of this write-up is as follows:

1. Procedure: Going through the development and formulation of my submission. This includes a summary of my progression, the structure of my code, assumptions and decisions made, and the algorithm I tried to implement.
2. Extra Questions: Answering the extra questions posed in the problem statement.
3. Future Work and Potential Improvement: Discussing the potential directions that can be taken beyond the scope of the problem statement.

2 Procedure

2.1 Algorithm

The reinforcement learning algorithm that I chose to implement is Deep Q Learning. The reason that I chose this algorithm is because I had worked with it before and thought that it could potentially be used for this problem. For this submission, I created a basic network comprised of 2 fully-connected layers with ReLU activation functions, and 1 linear, fully-connected layer. The input to the network is the state representation of the environment, which corresponds to the state of the predator and the nearest prey. The output layer corresponds to the 9 possible actions that the predator can take. The intermediate layer has 512 nodes.

2.2 Reward Structure

The reward structure used for the DQN implementation is a joint reward to try to encourage cooperation between the predators. For every non-terminal step within the environment, the predators each receive a joint penalty corresponding to sum of the euclidean distances between each predator and their closest prey. When a predator dies, they are removed from the environment and the penalty incurred at each step is higher than any possible penalty while still alive. For the terminal case where all prey are removed from the environment, the reward is 0.

2.3 Assumptions and Decision Making

One important assumption is that the environment state is fully observable by each predator agent. In a more complicated (and realistic) environment, the state will only be partially observable and the predator will only be able to see a limited portion of the environment. To simplify movement behavior, only agents that have new positions that don't conflict with other agents are allowed to move. If a prey has conflicting movements with another prey, neither will move. Likewise with predators. If a prey and a predator have conflicting movements, then neither moves. However, in this case, the environment will check for an attack. If the prey moves into a stationary predator, then this is not an attack. If the prey moves into a moving predator or a predator moves into a stationary prey, then this is an attack.

2.4 Code Structure

The code is split up into 4 parts: the main function that runs through the environment, the environment, agents that act within the environment, and the framework for a DQN agent.

The main file defines the parameters for the environment and then creates it. Once this is done, the code is designed so that you can either run the environment with DQN predators or with non-learning predators that follow the fixed policy of moving towards the nearest prey. This can be specified by defining the mode parameter. The environment is then run for based on the episode and step parameters and the number of rewards and successes is recorded. For the DQN version, DQN agents are called to make decisions for the predators in the environment.

The DeepQNetwork file defines a Deep Q Network that will be used by the DQN agents (The structure of the network is described in the next section). A DQN agent is then defined using the Q network. This agent is able to choose an action for the predator following an epsilon-greedy policy, return the Q value of a given state, update epsilon, and update the Q network. During the Q network update, the network is updated using the Adam optimizer and MSE loss.

The environment file follows the OpenAI Gym interface with the initialization, step, and reset functions. During the initialization, the grid size of the map is defined to be 7x7 and 4 prey and predator instances are created. Additionally, the state and action spaces are defined. Finally, each agent is given a unique, random position within the map. In the reset function, the map is reset to the initial state that was defined in the initialization state. In the step function, tentative moves are generated for every agent. During move generation, if the move will cause the agent to leave the map, the agent is forced to stay at its current position and will not move during the step. Only agents that do not collide with other agents are allowed to perform the generated move. If a prey's tentative new position causes it to collide with the tentative new position of a predator, then the intent of the predator must be checked. If the predator's tentative new position is the same as its old position then it is not attacking the prey and nothing happens. However, if the predator is determined to have been making a move away from its old position then it is considered to be attacked. Then it must be checked to see if other predators are also attacking the same prey. If there is at least one other predator attacking, then the predators are successful and the prey dies. Otherwise, the predator fails and dies. The death flag for the deceased agent is raised and the agent is removed from the map. There are 2 terminal states. If all the prey are removed from the map, then the episode ends and the predators are successful. On the other hand, if all the predators are removed from the map, then the episode ends and the predators fail. Functions are also implemented to get the current state of the environment and to check whether an attack is successful. The render function has not been implemented.

The agents file defines the prey and predator instances that act within the environment. The both classes are initialized to their initial position in the environment. They each have functions to update their position within the environment, return their current position, and to generate their potential next move within the environment. There is also a function within the move generation function that prevents movements that leave the map. The prey can choose to move north, south, east, west, or stay at its current position. The Prey class move function chooses randomly from these options with uniform distribution. On the other hand, the predator can choose to move north, south, east, west, northeast, northwest, southeast, southwest, or stay at its current position. The Predator class move function chooses a move based on its current policy. If the predator is non-learning, then the policy is to move towards the nearest prey using the euclidean distance. If the predator is learning, then the move is chosen by the network policy. Effectively, the learning predator should learn to behave similar to the non-learning predator.

3 Extra Questions

1. Two multi-learning agents that have been used in other research papers for the predator-prey problem are Multi-Agent Deep Deterministic Policy Gradient (MADDPG) and Multi-agent Proximal Policy Approximation (MAPPO). According to [1], MAPPO is better than MADDPG because predators are better able to learn cooperative behavior. Even for environments where the prey is able to learn, MAPPO still performs better than MADDPG but there is a constant shift in policies as predator and prey learn better policies against each other. [1] does not actually explain why MAPPO outperforms MADDPG in the predator-prey game. Additionally, [2] tests both algorithms for self-driving

simulations and finds that MADDPG considerably outperforms MAPPO. Some context for the two algorithms could help understand why MAPPO may be better than MADDPG in some environments but not in others. For this, [3] has useful information regarding the structure of both algorithms. MAPPO and MADDPG are extensions of PPO and DDPG to multi-agent learning, respectively. In MAPPO, the critic is learned using just the joint state of the agents. In MADDPG, the centralized critic is learned using the joint state and actions of the agents. In both algorithms, the decentralized agents learn on local observations. Based on the learning structures, one reason that MAPPO achieves better performance than MADDPG in the predator-prey game is because the joint actions do not give useful information regarding the cooperation between predators. Rather, it is more likely that the joint positioning of the predators is much more important for coordination than their joint actions. Intuitively, if the predators are all relatively close to a prey, they are more likely to successfully eliminate the prey. The actual actions the predators took to eliminate the prey does not appear to be as learnable and might add noise that can impede learning cooperative behaviors. This type of reasoning would also explain why MADDPG outperforms MAPPO for self-driving simulations. In these environments, the actions of other agents (other vehicles) are much more important for learning appropriate actions.

2. For this problem, I encountered multiple challenges while trying to train predators using learning techniques. The first of which is a scalability issue. With larger numbers of agents, the joint action space can be very large. To deal with this, I chose to use a decentralized learning algorithm for each predator rather than one centralized learning algorithm. However, one disadvantage of using decentralized learning instead of centralized learning is that it can be difficult to encourage cooperate behavior. One potential compromise for this is to used a centralized learner and decentralized execution by agents. This works well in environments where the state is fully observable. In this case, the learning goal is relatively straightforward. In the case where it is not so clear what the goal should be for each agent, it can be difficult to assess the performance of the learning models as convergence may not necessarily be indicative of good performance. One final problem that is specific to multi-agent reinforcement learning is the non-stationarity of the environment. This is because, as the agents in the environment learn, the environment begins to change. This is especially important for competitive and mixed environment structures. [1] actually explains that DQN has trouble finding cooperative behavior in the predator-prey environment. Based on this knowledge, I would likely have better results looking into some of the algorithms they have tested such as MAPPO.

4 Future Work and Potential Improvement

Due to time constraints and other work commitments, there are multiple areas that can be improved or further expanded upon. Currently, there is no framework set up to render the environment during each episode. While this is not a necessity, it would be nice to implement this to get a visualization of how the agents are behaving in the environment. This would make it easier to see if the agents are performing/learning the expected behaviors. Additionally, there is still a lot of room left to explore beyond the scope of this assignment. For this submission only tested one value-based method (DQN). Even this implementation has not been fully tested for performance and convergence due to the long execution time. One way to solve this problem, as mentioned previously, is to parallelize the computations using Ray to decrease execution time. Furthermore, testing on other value-based methods as well as on policy-based methods such as Proximal Policy Optimization would be useful to see which methods are most effective for this problem. Another area to explore would be how changes to the environment affect the behavior of the agents. For example, one could explore the effects of changing the number of predators/prey or adding obstacles to the environment. This problem can be made more complex by introducing a prey that is also capable of learning behaviors. One final area that could be tested is the effect of different reward structures. For this submission, every predator is penalized by the sum of euclidean distances between each predator and the nearest prey. When all the prey have been removed from the environment, the reward is 0. Another reward structure could include this joint penalty for each agent along with an individual penalty that adds an additional penalty for each agent based on their own euclidean distance to the nearest prey. In doing so, this could further encourage predators to more consistently move towards the nearest prey.

References

[1] Ken Ming Lee, Sriram Ganapathi Subramanian, and Mark Crowley. Investigation of independent reinforcement learning algorithms in multi-agent environments. 2021.

[2] Ansh Mittal and Aditya Malte. On multi-agent deep deterministic policy gradients and their explainability for smarts environment, 2023.

[3] Joshua John Wilkins. Multi-agent deep reinforcement learning: Revisiting maddpg. 2021.