Project Report

Rishabh Khanna- 2019113025, Bhumika Joshi -2022121006 , Aadarsh Raghunathan -2019113021

Learning Inter-Agent Synergies in Asymmetric Multiagent Systems

Gaurav Dixit, Kagan Turner - AAMAS 2023

Contributions of the Base Paper

The base paper provides the following contributions:

- Discusses the shortcomings of existing cooperative multiagent learning algorithms. Learning Individual Intrinsic Rewards (LIIR) is a method which uses a linear combination of parameterized intrinsic reward and team reward to facilitate learning. However, its gradient based alignment between the two rewards makes it unscalable to problem with sparse team rewards. Similarly, Multiagent DDPG which uses a centralized critic to optimize the joint actions of all the agents to address this, also relies on dense team rewards and becomes intractable as the no. of agents increases. Multiagent Evolutionary Reinforcement Learning (MERL) combines gradient based learning on individual rewards with an evolutionary algorithm to maximize team rewards while also ensuring alignment. However, the shared buffer replay architecture limits it to homogenous agents.
- Introduces Quality Diversity (QD) methods as a solution to learning diverse policies that are required in multiagent systems that require agents to cooperate with asymmetric agents which perform different tasks. A QD method is an iterative process that mutates policies in a population and organizes them in a behavior space for refinement. It allows the maintenance of a diverse set of well performing policies in a population of policies.
- Introduces the Asymmetrical Island Model (AIM) as a multiagent learning framework that learns inter-agent synergies (cooperation) between asymmetric agents across a wide spectrum of tasks. AIM has three sub-parts: Islands, Mainlands and Policy Migration. 'Islands' maintain a set of diverse agent class specific policies using the QD method and are trained to maximize agent-specific rewards. 'Mainlands' represent distinct team tasks and maintains a population of teams which comprise of agents (policies) received from their respective islands. The mainland optimizes the population of teams using an evolutionary algorithm. Policy Migration transfers the policies from their respective islands to the mainlands based on a SoftMax distribution utilizing weights which get updated based on the performance of the policies and the teams. It also transfers policies from the best performing teams in the mainlands back to their respective islands to allow the QD process to focus on policies similar to these well performing ones.
- Tests the performance of AIM on the habitat problem. The habitat problem is a scenario where three different types of agents, namely, rovers, excavators and drones get deployed in a remote environment where they must work together to find suitable dig sites, excavate regolith and communicate the no. of excavated dig sites back to the ground station. The rovers are equipped with sensing devices which they can use to locate, visit and mark dig sites. A site gets marked when c rovers visit it simultaneously. The excavators possess the equipment to visit marked sites. If c excavators visit a marked site simultaneously, then it can get excavated. Drones are

equipped with communication infrastructure to communicate the excavated dig sites within their observation radius. The AIM model is trained on various environments like decay, where the value associated with a dig site decreases over time, volatile, where the sites become unmarked after a certain period if they are not excavated, constrained, where the size of the environment is half its usual value, sparse which doubles the size of the environment while keeping the no. of sites constant and mixed, which combines the decay and volatile scenarios.

Implementation

As part of the implementation of the base paper, the AIM model for the Habitat Problem has been coded in python.

Agent Classes

Each of the three agent types: rovers, drone and excavators, have been implemented in the form of classes.

Rover

Each rover is initialized at a random position inside the habitat boundaries. The habitat environment is divided into 4 quadrants. Using the compute_quadrant function, given a position, the quadrant it belongs to is calculated. This is useful in computing the site and agent density for each quadrant.

Each object of the rover class has a neural network architecture associated with it. It consists of an input layer of size 20 connected to a hidden layer of size 32. This hidden layer further connects to an output layer of size 2. This 2D output corresponds to the amount movement in x(dx) and y(dy) directions that the rover should implement. In the get_next_move function of the rover class, the site and the agent densities are passed as the input data to the neural network model and the movement coordinates are received. The rover object proceeds to change its position by dx in the x direction and dy in the y direction.

The rollout function takes in the locations of the sites and agents and the no. of timesteps as parameters. The site density and the agent density at a particular timestep form the state at that timestamp. The working of the rover object for the given number of timesteps is simulated where the rover object calculates the next move given the sites and agents and moves at each timestep. The current state, the move and the associated reward (which is the inverse of the distance of the rover from the closest dig site and is calculated by the reward function) is stored in the trajectory at each timestep.

Excavator

The excavator class is similar to the rover class and possesses the same functions as the rover class. The only difference is that the neural network associated with the excavator class for calculating the next move has the input layer of size 16 instead of 20.

Drone

The drone class is similar to the excavator class and has the same functions.

Island Class

An instance of the island class contains agents (policies) of a particular class, that is, either rover, excavator or drone class. Each island object starts with a certain number of agents (numAgent) or policies which get initialized by the initialize_agents function and added to the list of agents on that island.

The update_policies function runs N iterations. In each iteration a random policy is selected from among all the policies (agents) on the island. The weights associated with the neural network responsible for moving the agent are perturbed by adding some small gaussian noise. A rollout of the agent with these perturbed policy weights is performed using the rollout function of the agent class and the returned trajectories are stored as the rollout data. The neural network model is then optimized by performing PPO with the help of the train_ppo function which takes in the neural network model and the rollout data and performs the optimization, returning the updated model. The policy with this new neural network model gets appended to the list of policies in that island.

The update_latent_space function gathers all the neural network models for each policy on the island and extracts the weight data of each model into a weight vector. This vector is used to train a PCA model which reduces the dimensionality of the weight vector data to 20 and stores them as the reduced weight vector. The distances between the model parameters are calculated and the top numAgent most distant policies are retained thus eliminating similar policies to maintain diversity.

Team Class

A team consists of multiple agents from the various classes. They are stored in the form of lists, one each for rovers, excavators and drones.

The mark_site function marks an unmarked dig site if the coupling requirement is satisfied, that is, c rovers have the site within their observation radius.

The excavate_site function excavated a dig site if the coupling requirement of c excavators having the marked dig site within their observation radius is satisfied.

The rollout function simulates the operation of a team given the habitat dig sites for the given amount of timesteps. At each timestep, all agents move according to the output of their policies neural network given the current state and try to mark and/or excavate any dig site possible.

The get_fitness function performs a rollout for the team and calculates the fitness of the team with the help of the calc_fitness function. The fitness of a team is the summation of the values associated with all the dig sites which have been excavated and have at least one drone covering them (has the site within its observation radius).

Mainland Class

Each mainland object has a team size (that is, the number of agents in a team), a population size (number of teams in the mainland) and the number of teams to be considered as elite teams. It is also provided with data associated with the habitat where the teams will be operating like the habitat extents and the number of dig sites.

The habitat is to be generated by the initialize function which places the dig sites within the habitat at some random locations. Agents are received through the update teams function and teams are formed by random sampling of the agents to create the required no. of teams of the required size.

In the crossover function, an agent of either the rover class, the drone class or the excavator class gets exchanged between the two teams (one elite and the other non-elite selected using tournament selection) and the of the two resulting children, the child with the higher fitness is returned. The crossover_possible function checks whether a crossover between two given teams is even possible. For a crossover to be possible, the two teams must have at least one common agent which has a common agent class.

The get_elite function sorts the teams that form the population by order of their fitness and selects the e fittest teams as the elite teams. The rest of the teams are considered non-elite teams.

The evolutionary algorithm is performed in each mainland. It goes on for N generations or iterations. In each iteration the elite teams in the population are identified and cross-overed with non-elite teams selected through tournament selection. The returned child of the crossover is added into the population and the algorithm moves on to the next iteration.

Policy Migration

The policy migration process is responsible for integrating the working of the Islands and the Mainlands and for the overall operation of the Asymmetrical Island Model. It consists of four main parts.

The first is to define a SoftMax function. Using init_weights function, each island is assigned an equal weight across each mainland. The distribution_of_weights function calculates the SoftMax probability associated with each island-mainland pair.

Secondly, after initializing the islands and the mainlands, for N iterations, the policies of each island are optimized using the functions that are part of the island class as stated above. Then, the policies from each island get sampled into each mainland following the SoftMax probability associated with each island-mainland pair using the select_agents function. The teams are optimized with the help of an evolutionary algorithm with the help of the mainland class functions discussed previously.

Next, the e elite teams get selected from each mainland and the agent policies comprising these teams get appended to the current population of the islands. The latent space of each island gets updated using the update_latent_space function of the island class allowing the island optimization to focus on the vicinity of the successful policies while discarding any similar policies to help maintain diversity in the island population.

Lastly, the weights associated with the SoftMax get updated using the update_weights function based on a gradient based on the agents' performances on the mainland in the current iteration. An entropy regularization term makes sure that the contribution of any island to a mainland does not reduce to zero, while the alpha value determines how much preference the current iteration's performance is given over the existing weight value.

The non-elite teams of each mainland get replaced by new teams created by sampling from the next input of the agent policies from the islands and the process goes to the next iteration and continues forward in a similar manner.

Contributions Covered from the Base Paper:

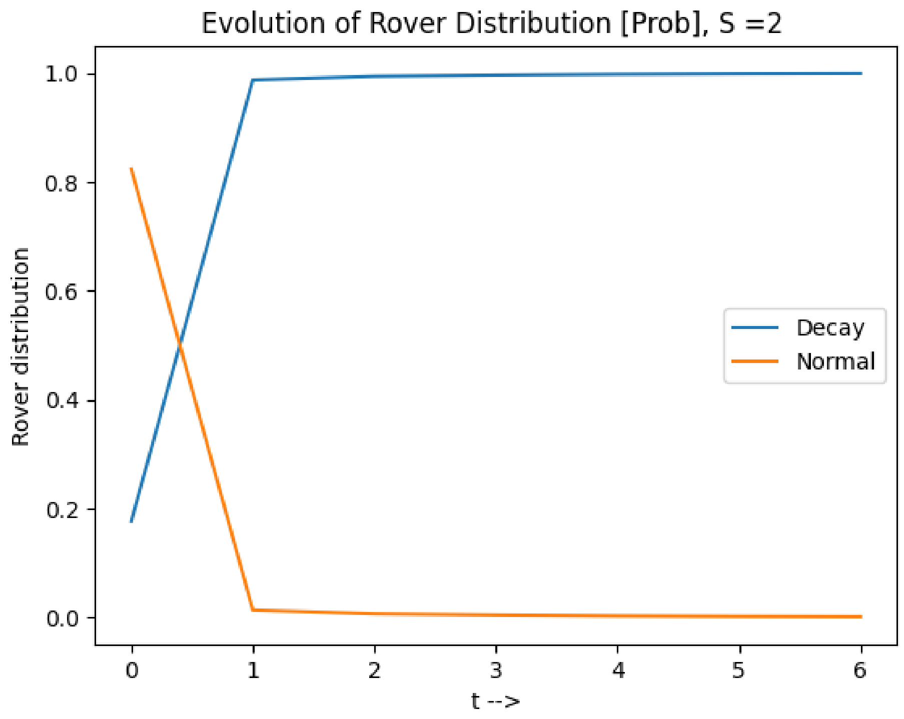
From the paper, we implemented the three-step algorithm to perform quality diversity on islands, evolutionary optimization on mainlands and policy migration between islands and mainlands. We also implemented various game modes (Decay, volatile, mixed) to test our agents in challenging environments.

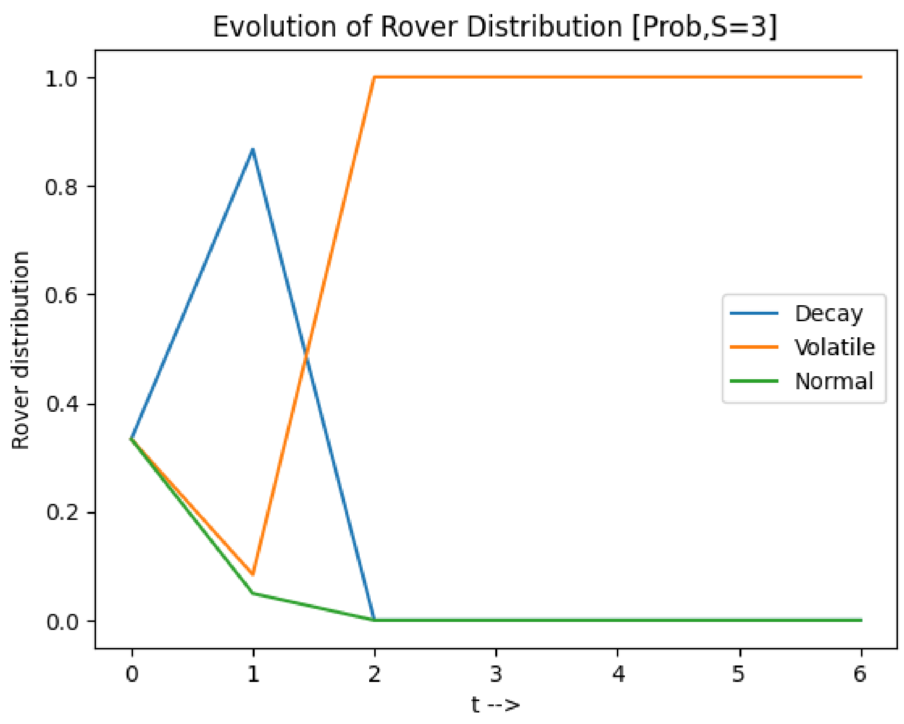
The paper proposed three metrics to test the efficacy of the model. These were asymmetric coordination, adaptability to unseen tasks and understanding the correlations between behaviors and tasks. We have performed rollouts based on the algorithm to test the first 2 of these metrics. The main bottleneck that we faced was training time for the model. The authors performed approximately 6000 iterations of policy migration process with a very slow learning rate to obtain converged results. Training the model for a comparable amount of time was not feasible for us.

Given this limitation, it was hard for us to try and understand the correlations between actions and tasks. Moreover, due to time constraints, we were unable to implement the baseline algorithms that the paper used, such as MERL and multiagent DDPG.

Results:

To understand the flow of agents across between the mainlands as a function of training time, we looked at the evolution of the softmax distribution for the rovers. This graph determines how crucial the rovers are to the different game modes on the islands. We considered 2 different training settings, one with S= 2 and the other with S =3, where S is the number of distinct team tasks that we train using.





Here, we can observe the relative value of different agents to different game modes. In the volatile game mode, marked sites become unmarked after a short period, so more rovers are required to continuously mark the sites. Therefore, after a short training period, the distribution completely skews towards sending rovers to the volatile mainland. Similarly, in S =2, we see that decay captures most of the rovers, as they need to quickly mark the sites before they lose value.

Comparison of Results from the Base Paper:

The inferences made in the above section don't exactly correspond with the paper. According to the authors, rovers are most important in decay, as demonstrated in the case of S=2. However, in volatile, excavators become the most important because they can quickly dig up marked sites before the sites reset, which we don't see in our simulations. We suggest this is because of our extremely short training period compared to the authors. We also see very quick changes as compared to the base paper, because we significantly increased the learning rate (0.1 for us, 0.00001 for the authors) due to limited training time.

Learnings

Through this project, the additional complexity and requirements for learning cooperative strategies in the case of asymmetric multi-agent systems was understood. We learnt about the Quality Diversity methods which help maintain a diverse set of well performing policies so that there are multiple choices to choose from when trying to learn a set of policies from different asymmetric agents that are synergetic (help enhance the performance of each other).

We also learnt about the model proposed by the base paper, the Asymmetric Island Model, which has three broad subparts. Firstly, best performing policies are identified and optimized for each individual agent class using the Quality Diversity method. We also learned about Proximal Policy Optimization (PPO) which is a reinforcement learning algorithm based on the policy gradient method which iteratively updates the policy by sampling trajectories from the environment and computing the policy gradient, which is the direction that improves the expected reward. It improves the training stability of the policy by limiting the change one can make with the help of the clip function, clipping the ratio of the old policy and the new one between [1- ε, 1+ ε].

Through this project, the evolutionary algorithm was realized as a possible way to optimize team rewards in the case of asymmetric agents. The evolutionary algorithm samples the policies of different agent classes to form a team of agents that need to cooperate to solve a given problem. These teams are assigned a fitness score based on their combined performance in the simulation of the task. The fittest teams are selected and cross-overed with other teams selected through tournament selection (where the probability of selection of a team is proportional to their fitness score). The child of this crossover is added to the population of teams and this process continues until the terminating condition (number of iterations in this case) at the end of which we get the best performing teams.

We also learned of a way to bridge these two separate optimization processes through the policy migration process. Under this, the final elite teams provide feedback to the individual agent class optimization process (the islands) by returning the policies that comprise the best performing teams. This helps optimization of the specific agent class policies to continue further. Also, the probability distribution for sending over the policies from the individual agent classes to the evolutionary optimization process (the mainlands) is updated with the help of a gradient based on the performances of these policies in the team tasks, thus allowing the team formation process to learn the ideal team composition for a given task.

Through the process of implementation of this model, we also learned the technicalities of implementing the procedures and algorithms used, like implementing individual agent policies in the form of neural networks, performing PPO on a given policy, implementing evolutionary algorithms to optimize team performances, maintaining and sampling from SoftMax distributions and updating these distributions with the help of policy gradients, etc.