Swarm Robotics

Romika Sairam
romikasa@buffalo.edu

Michael Mu
msmu@buffalo.edu

Sachin George
sachinge@buffalo.edu

Abstract

Our project will relate the fields of swarm robotics and reinforcement learning, resulting in a multi-agent environment problem. We will have multiple agents that each are controlled by the same reinforcement learning algorithm. They will have to work together to accomplish the goal of finding a target, then working together to move the target. The objective of the project is to design an algorithm that is able to control agents to robustly navigate an unknown environment to reach a target location and to work together to accomplish a greater task. In the real world, this sort of problem has several practical applications. An example would be locating an incapacitated child in a dangerous/cramped environment and then using multiple, small agents to pull the child to a safer location. This concept can be expanded to any object, animal, etc. as well as any environment from earthquakes to wreckages.

Background

The most relevant prior work and a major inspiration of our work originates from the swarm-bot project in 2002 [1]. This project later evolved to the swarmanoid project [2] and continued into ASCENS. Swarm-bots originated as a project to imitate the cooperation of ant-like groups with different agents and roles, and goals were often accomplished in a predefined manner. This can be seen in the original paper when the goal was to probabilistically form chains. In future implementations, neural networks and genetic algorithms begin to be utilized for control [3]. However, these control systems were specifically designed for motion control like trap avoidance, moving towards a target, and successfully grabbing. Our application, on the other hand, will focus on applying reinforcement learning to the control problem. Rather than focusing on implementation, we wish to explore the ability of many multi-agents being able to learn to work in tandem to accomplish a goal. In doing so, our implementation should generalize better. For example, our agents should be able to search cooperatively and create plans and paths without specification. The agents aren't limited to act in a manner we set, but will rather learn to interact in a way that best accomplishes the overall goal. Multi-agent reinforcement learning applied to swarm robotics is a relatively unexplored area. Cooperative agents in previous works generally operate in smaller numbers and can act more complexly on their own.

Real-world RL application

A real-world RL Application for this project and the algorithm would be in a situation where it is practically unsafe for humans to accomplish tasks such as dragging a victim from one place to another. For instance, in case of a fire accident and it is unsafe for other human beings to enter, we can send these agents to safely recover a victim stuck. Another scenario would be if a building falls apart and there are victims stuck inside, these agents can easily navigate the victims and drag them to a safe destination. Overall, the swarm bots can be utilized in different scenarios involving natural and man made disasters and will be very helpful in the recovery process.

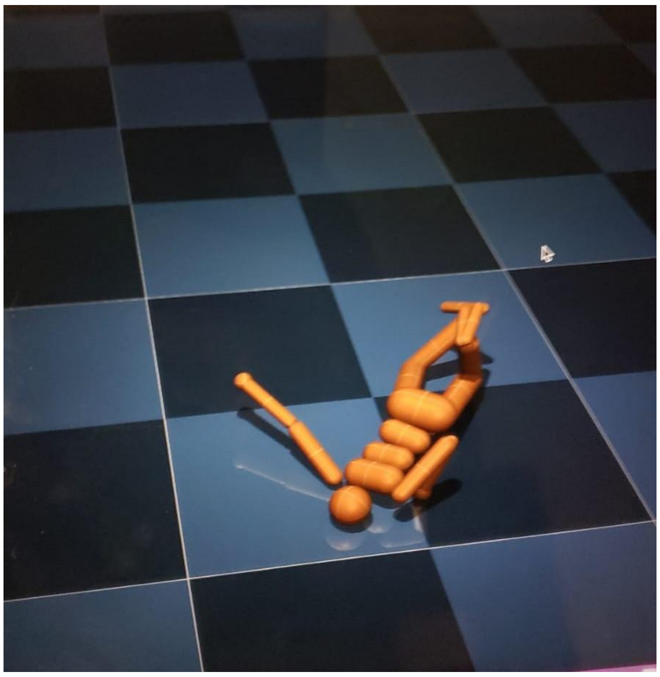
Implementation

Environment Description

The environment is a continuous environment consisting of a flat plane with enclosures surrounding the borders. There are four agents, which are trained separately and approximated with cylinder shapes. The target is a part of the environment which we wish to direct our agents to find and move. In our default environment, this is a ragdoll humanoid. A destination is selected in the environment, which is where we wish the target to be brought to. The agents are initially randomly located in a region about 2 meters away from the target. They are tasked with working together to navigate the environment to locate the target. Once they reach the target, they work together and cooperate to push the object to the destination. We custom designed our rewards to fit this goal. There is a small punishment that accumulates over time for failing to accomplish anything. There are rewards for finding and approaching the target. The majority of major rewards are received by pushing the target towards the destination.

Design Process

We have explored algorithms like Advanced Actor-Critic and DQN to compare how the two algorithms work with our multi-agent environment. Each agent is trained with its own instantiated version of the same algorithm. For the simulations, and to get a better understanding of our environment while training, we have utilized MuJoCo. The objects, physics, and movement mechanisms of the target and the agents are defined and designed using an XML syntax designed by MuJoCo.



Visualization of our environment where the victim is placed somewhere on the grid

Environment with agent bots

The victim is moved by the agents, to the target location

DQN

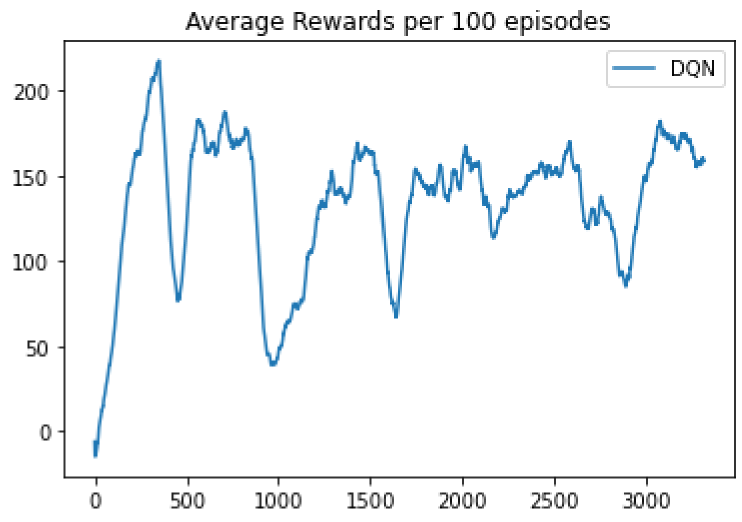
The Deep Q- Network (DQN) is a value-based algorithm which we use as a baseline common method of solving environments in deep reinforcement learning. It is able to stably solve deterministic environments quite consistently for simple problems. DQN uses a replay memory which saves relevant training information such as the agent's previous states, actions, rewards, and next states. We also save a conditional determining whether the state is terminal. In this implementation, two neural networks are created with identical structure but one of them will be frozen. Then, the normal epsilon-greedy algorithm is still used to decide between exploiting the highest value action or exploring a random action. A core part of this algorithm is that the values for each action are derived from the output of the unfrozen, policy neural network, which tries its best to predict the value of each action resulting from the input state. The resulting reward and state are then observed and compiled with the previous state and action to create a transition sample that we store to our replay memory list. Then, we sample from our replay memory randomly, evaluating our neural networks based on the ability to properly estimate the value of states. Using a frozen, target neural network allows us to more accurately define a trend in training that allows for more stable convergence. Stochastic gradient descent is then performed on the unfrozen neural network based on the loss.

Advantage Actor-Critic

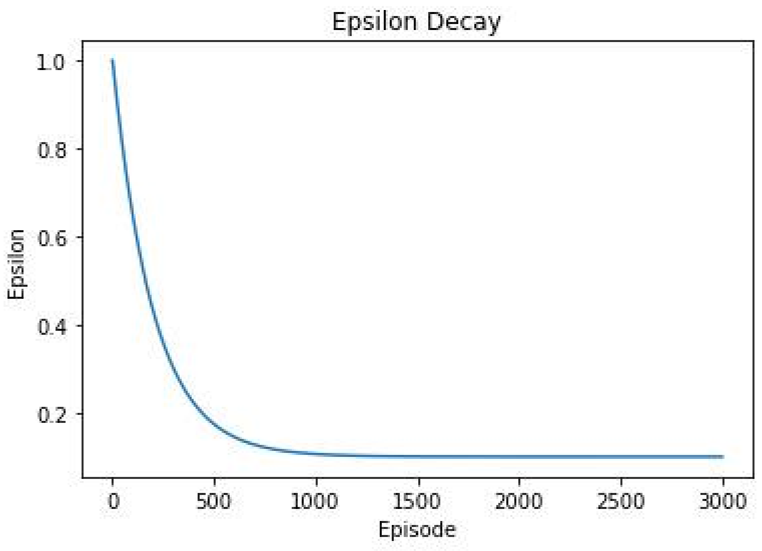
In general, actor-critic algorithms perform by combining the concept of a policy gradient algorithm with the state evaluation of a value-based algorithm. There are two neural networks, called the actor and the critic. The actor produces a probability distribution of what is likely the best optimal action for a given state. The critic rates the state and actions that the agent is taking. In our implementation, we use the advantage actor critic algorithm, so the critic rates the advantage of a move. Using the critic's rating, the actor is then able to train its performance by backpropagating through its neural network to maximize the advantage. The critic improves by having the agent keep track of the actual rewards that follow and calculating the actual advantage, and then the loss is determined by computing the mean squared error between the actual advantage and the predicted advantage. The actor-critic has an advantage over value-based algorithms because it does not need to base its performance solely on the value of future states, but can actively take part in designing a plan for the highest performance. However, this algorithm is on-policy, meaning that it needs to generate new training data as its policy changes. Because of this, the algorithm requires more sample-intensive training.

Results

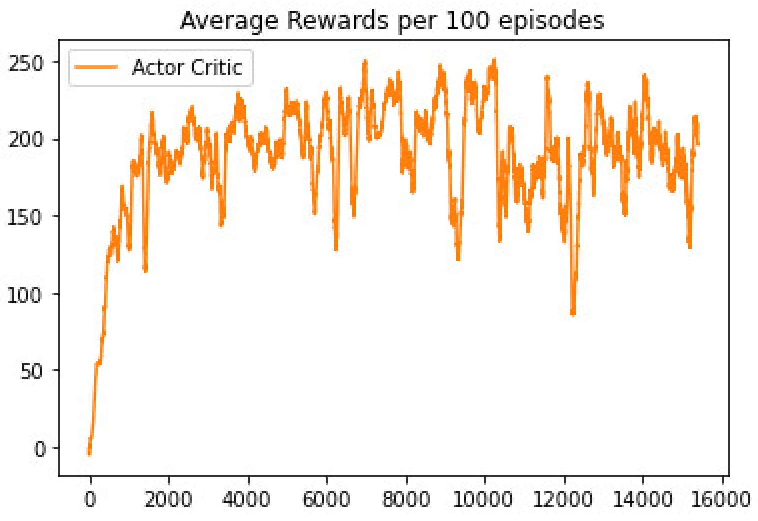
For our implementation, we have run our environment over the DQN algorithm, and the Advantage Actor-Critic algorithm. The results are portrayed and discussed below:



The average rewards over DQN algorithm

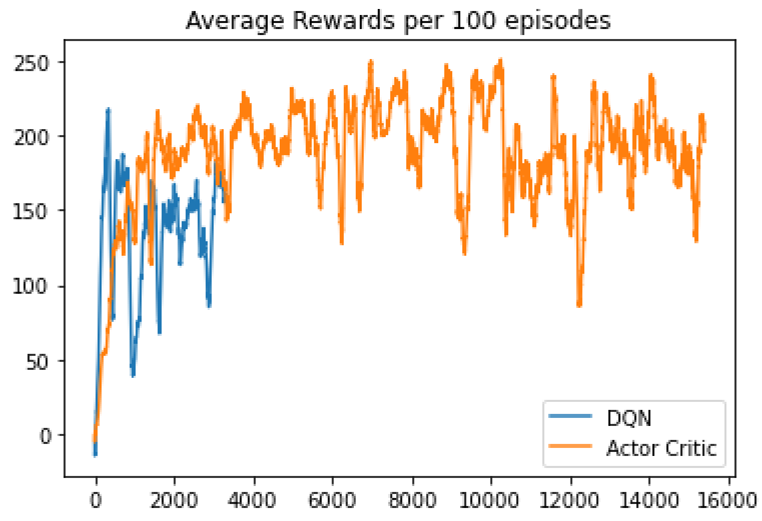


Epsilon Decay graph for the DQN Algorithm over 3000 episodes



The average rewards over Actor Critic algorithm

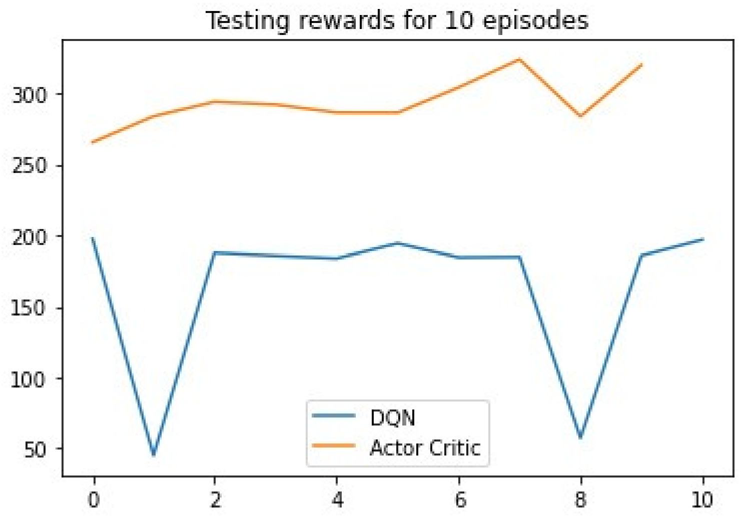
This graph above shows how the two algorithms impact the average rewards in our environment. While the DQN algorithm starts off learning faster and achieving higher rewards than the A2C algorithm, the performance begins to degrade after reaching a peak. The average rewards for the A2C algorithm, however, performs consistently with a higher reward.



The average rewards over both algorithms

When following the optimal policy, the comparison of the best performing DQN and the best performing actor-critic is clear. The actor-critic is able to formulate a complete plan and solve the environment quite well. The DQN algorithm is able to acquire many initial points by finding the target and beginning to push the agent. However, it is unable to completely finish pushing the target to the destination.

The reason for this is because the DQN learns from random movements resulting from the epsilon greedy algorithm. While theoretically, this algorithm has the potential to also solve the environment, we discovered the DQN algorithm only trains to get the agent about halfway to the destination. From there, random motions should work to train the algorithm the rest of the way. However, we also have to contend with catastrophic forgetting, where the agent begins to fail on state observations that the neural network has forgotten. In future work, increasing the neural network and replay buffer significantly can combat this issue, at the cost of significant training time and computational cost. However, we focused on algorithms that are relatively comparable in size to the actor-critic.



Conclusion

Overall, we found that the A2C algorithm works best for this environment, but the DQN algorithm shows potential for better performance if we provide more resources to it. By showing the successful application of algorithms in this environment, we show the potential reinforcement learning has in the field of swarm robotics. Applications of this work can be used in real-world situations, where it may be too cramped or dangerous for humans to go in. By introducing many small, robusts agents, the process of search and retrieval can be much more efficient and safer. In the future, if we were to improve this algorithm more, we would like to introduce more complicated environments to force the agent to more proactively search for targets/victims and avoid obstacles. We would also like to add more functionality to the agents, such as adding a pulling features, which would allow the agents to be more versatile while working collaboratively.

Contribution Summary

Michael Mu

- Created the base environment required.
- Coded and ran the A2C algorithm over the environment.
- Researched on the required methods to implement the project.
- Worked on the demo for the actor-critic algorithm.
- Tried and failed to implement PPO

Sachin George

- Coded and ran the DQN algorithm over the environment
- Worked on the report for the checkpoint, and gathered necessary information with research.
- Gathered all visuals required for the presentation and the report.
- Worked on the demo for the DQN Algorithm.

Romika Sairam

- Prepared the presentation slides, researched and gathered adequate information.
- Prepared and summarized the final report
- Gather necessary visuals and information for the final report.
- Edited the demos.

References

PPO: https://arxiv.org/pdf/1707.06347.pdf
Policy Gradient: https://www.youtube.com/watch?v=5P7I-xPq8u8&ab_channel=ArxivInsights
MuJoCo Github: https://github.com/openai/mujoco-py
MuJoCo Documentation: https://mujoco.readthedocs.io/en/latest/overview.html

[1] **SWARM-BOT: Pattern Formation in a Swarm of Self-Assembling Mobile Robots**
Sahin E., Labella T.H., Trianni V., Deneubourg J.-L., Rasse P., Floreano D., Gambardella L.M., Mondada F., Nolfi S., Dorigo M.
In A. El Kamel, K. Mellouli, and P. Borne, editors, *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisia, October 6-9, 2002. Piscataway, NJ: IEEE Press.

[2] **Swarmanoid, The Movie**
Dorigo M., Birattari M., O'Grady R., Gambardella L.M., Mondada F., Floreano D., Nolfi S., Baaboura T., Bonani M., Brambilla M., Brutschy A., Burnier D., Campo A., Christensen A.L., Decugnière A., Di Caro G.A., Ducatelle F., Ferrante E., Guzzi J., Longchamp V., Magnenat S., Martinez Gonzalez J., Mathews N., Montes de Oca M.A., Pinciroli C., Pini G., Rétornaz P., Rey F., Roberts J., Rochat F., Sperati V., Stirling T., Stranieri A., Stuetzle T., Trianni V., Tuci E., Turgut A.E., Vaussard F.
In *AAAI-11 Video Proceedings*, 2011, Winner of the AAAI-2011 Best AI Video Award

[3] **Autonomous Self-Assembly in Swarm-Bots**
Groß R., Bonani M., Mondada F., Dorigo M.
IEEE Transactions on Robotics, volume 22(6), 1115-1130, 2006