Multi-Agent Reinforcement Learning and Cooperation

Cade Bruce, Aditya Retnanto

TTIC 31170: : Planning, Learning, and Estimation for Robotics and Artificial Intelligence

Professor Matthew Walter

Abstract

We explored deep Q-learning for a variety of Multi-Agent Particle Environments (MPE) environments: including collaborative and mixed environments. We implemented centralized, decentralized, and value decomposition techniques.

1 Introduction

In this project, we wish to explore algorithmic approaches multi-agent systems undertake to achieve a collective goal and further extend concepts beyond the scope of the class. Swarm robots have been proposed in lieu of more complicated single-agent systems as they are often cheaper and can have multiple modes of failure.

We are motivated by *Swarm User-Interfaces* - where miniature robots act as a physical embodiment of a pixel. In the digital realm, "cooperating" and "planning" are trivial tasks. Current implementations of the system utilize planning algorithms such as Hungarian Goal/HRVO Global Control[2] and Conflict Based Search[5] but both are computationally expensive for real-time realizations of such systems and are not able to handle modularity (replacing robots) and scalability (changing the number of robots) during human interaction. Rather than rely on centralized control, we hypothesize that agents themselves can: 1) learn the optimal trajectories to avoid collisions, 2) reach a goal state In this project, we explore how decentralized decision-making agents can cooperate to perform global objectives.

2 Theory

2.1 Preliminaries: POMDP and Deep Q-Learning

Our environment is represented by a partially observable Markov decision process (POMDP)[8]. Formally, a POMDP is a tuple $(S, A, P, R, \Omega, O, \gamma)$, where

- S is a set of states $\{s_t\}$.
- A is a set of actions $\{a_t\}$.
- $P : \mathcal{S} \times \prod_{i\in[N]} \mathcal{A}_i \times \mathcal{S} \to [0, 1]$ is the transition function. It has the property that for all $s \in \mathcal{S}$, for all $(a_1, a_2, \ldots, a_N) \in \prod_{i\in[N]} \mathcal{A}_i, \sum_{s'\in\mathcal{S}} P\left(s, a_1, a_2, \ldots, a_N, s'\right) = 1$
- $R : S \times A \to \mathbb{R}$ is the reward function.
- Ω is a set of observations.
- O is a set of conditional observation probabilities.

Preprint. Under review.

- $\gamma \in [0, 1)$ is the discount factor.

Recall that Q-Learning estimates the value of executing an action in a given state with respect to the reward. We use an off-policy algorithm, so we learn to estimate and evaluate one policy based on experience obtained by following a different policy. The Q-values are updated iteratively, bootstrapping based on the estimated optimal value function:

$$Q(s_t, a_t) \rightarrow Q(s_t, a_t) + \alpha \left(\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') \right) - Q(s_t, a_t) \right)$$

Then the action with the highest Q value is taken. In order to compensate for the exploration v. exploitation tradeoff, we utilize epsilon greedy actions, so that there always remains a small probability of taking a random action. This ensures the entire space of actions is explored in the limit, so that early convergence is avoided.

We implement Deep Q-Learning[1], where the model is a neural network parameterized by weights and biases collectively denoted as θ. Q-values are estimated online by querying the output nodes of the network after performing a forward pass given a state input. The resulting Q-values are denoted $Q(s, a|\theta)$. Instead of updating individual Q-values, updates are made to the parameters of the network to minimize the loss function.

Another feature of Deep Q-Learning is the use of a stored replay buffer or experience buffer; which is a collection $\{e_t\}$ where each

$$e_t = (s_t, a_t, r_t, s_{t+1}).$$

This storage facilitates batch learning where we randomly sample from different time steps. The result is a break in the temporal structure, making the data more independent; in contrast with standard temporal difference learning, which is prone to bias from temporal correlations between consecutive experiences. Additionally, the replay buffer enables us to learn from the same experiences multiple times in different ways (since it can be resampled); enabling greater sample efficiency.

2.2 Independent Q-Learning

For our first model, we implemented Deep Q-Learning independently for each agent. That is, each agent is trained and acts greedily on their own with respect to their learned Q-value.

2.3 Advantage Actor Critic (A2C)

We also experimented with an actor-critic algorithm.

The actor is the policy that conducts actions in an environment, while the critic computes value functions to assist the actor in learning.

The advantage function represents the state-action value minus the state value:

$$A(s, a) = Q(s, a) - V(s)$$

where $Q(s, a)$ is the state-action value function and $V(s)$ is the state value function. The policy gradient update for the actor, (θ), is given by:

$$\nabla_\theta J \approx \mathbb{E}\tau \sim \mathcal{D}[\nabla\theta \log \pi_\theta(a \mid s) A(s, a)]$$

where $\mathcal{D}$ is the replay buffer, τ is a trajectory, and $\pi_\theta(a \mid s)$ is the policy.

2.4 The Problem with Centralized Control

To deal with the issue of cooperation and conflicting agent goals, it is natural to attempt the use of a centralized learner that controls the actions of all agents. However, in practice, our attempts at this saw drastic failure. To gain a better theoretical understanding of why this is (a question posed by Professor Matthew Walter in our presentation), we provide some commentary and then briefly review some of the literature. Intuitively, a centralized actor has many degrees of freedom. Let n be the

number of agents, each with k-many actions. Then the number of possible action combinations is n^k. Considering a group of sequential actions, the number quickly becomes extraordinarily large.

This would take many trials to explore; where each single agent individually need only explore k actions and can more efficiently converge to the optimum (at the sacrifice of potentially interfering with the other agents).

Additionally, in different real-world applications, a centralized actor may not be feasible. For all the methods we discuss, it is possible to apply a pre-trained model to robots without communication (of their planned actions).

In [4], Lyu et al show theoretically that the centralized critic results in higher variance updates of the decentralized actors assuming converged on-policy value functions. They further emphasize that the stability of value function learning does not directly translate to a reduced variance in policy learning.

2.5 Value Decomposition Networks (VDN)[7]

To make each agents more aware of their impact on the performance of the other agents, we introduce a want to instead optimize with respect to a global Q value function. A natural idea is to maximize the sum of the Q value functions for each agent

$$Q_{tot}(\{a_i\}, \{s_i\}) = \sum_i Q_i(a_i, s_i).$$

The idea behind VDN is that we can learn each Q_i by backpropogating using a loss on the joint sum, and each agent acts independently with respect to the learned Q_i value functions.

2.6 QMIX[6]

Note that for VDN, consistent learning is contingent on the fact that maximizing the joint value function also performs an argmax for the individual Q value functions of each agent:

$$\underset{\mathbf{u}}{\text{argmax}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \text{argmax}_{u^1} Q_1\left(\tau^1, u^1\right) \\ \vdots \\ \text{argmax}_{u^n} Q_n\left(\tau^n, u^n\right) \end{pmatrix}.$$

Then rather than working with the sum, letting Q_{tot} be any other monotone combination of Q_i will also be a valid methodology in the VDN framework. In particular, we can let Q_{tot} be any neural network that implements the constraints that the weights must be positive.

3 Environments

To ensure practicality of the system We utilize *Pettingzoo*[8] to simulate our results rather than deploy them on robots. We selected the Multiple Particle Environment [3] . The selected environment for pure cooperation is *simple spread* in which N agents must cover N landmarks. Agents are homogenous with the same physics model and observation capabilities. Further, agents can not communicate with each other (can not view other agent's intended actions). Each agent is awarded a global reward that is calculated by how close an agent is to a landmark and locally penalized by the number of collisions an agent makes. We weigh this through the parameter which we set as 0.5. We summarize the environment in table 1.

4 Experiments

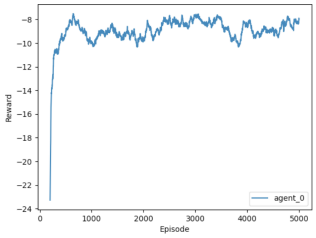
We first test each algorithm's performance in a base case - the simple environment. In a single agent environment it is expected that the agent learns an optimal. Next, we test the performance of the algorithm in a purely cooperative multi-agent setting where we evaluate its rewards over time and qualitative behavior. The first set of algorithms implemented is purely decentralized - each agent will greedily perform an action. The second set of algorithms introduces a centralized network to better.

Table 1: Multi Particle Environment

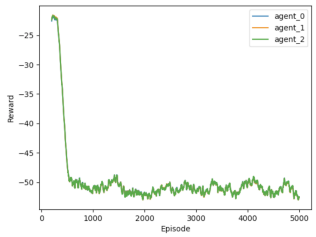
	Description	Type
Action	{No Action, Left, Right, Up, Down}	Discrete
Observations	{Vel, Pos, Landmark Pos, Other Pos, Communication}	Continuous
Reward	β*(Euclidean Distance to landmark) + (1-β)* number of collisions	Continuous
State		Continuous

Table 2: Agent parameter

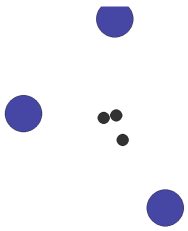
γ	0.99
ϵ	0.1
batch size	64
lr	0.001



(a) DQN Baseline



(b) DQN Mutliagent



(c) Particle Position at final step at Episode 4000

Figure 1: Deep Q-Learning

For all off-policy agents, we employ an $\epsilon-$greedy approach to encourage exploration. We report the parameter in table 2

Additionally all neural architectures are variations of a 3-layer MLP with observations as inputs, a hidden layer and an action output layers with intermediary rectified linear units.

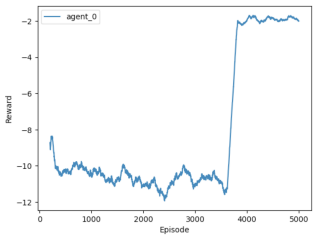
4.1 Independent Q-Learning

The baseline single agent-model clearly learns and improves with more episodes. However when introducing multiple agents to the environment the agents perform worse than random policy.

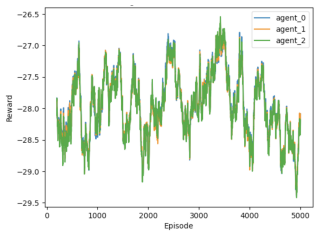
When observing the qualitative1c behavior of the agents, they initially approach the landmarks haphazardly and over time learn to avoid this sequence of action. This ultimately results in timid agents as they wish to avoid the collision penalty.

4.2 Independent Advantage Actor Critic

In our implementation of an independent advantage actor critic we likewise find successful learning of the policy albiet at a later episode 2a. In the multi-agent setting, the independent actor-critic



(a) A2C Baseline



(b) A2C Multi

(c) Particle Position at final step at Episode 4000

Figure 2: Advantage Actor Critic

model had improved results compared to the DQN in terms of the reward but still has not learned a cooperative policy2b.

When observing the simulation run, we find that a singular agent approaches the landmarks while the other two remain approximately near their starting location. While such strategy may lead to higher rewards it does not result in the ideal cooperative outcome of each agent approaching a landmark.

4.3 Value Decomposition Networks

For the implementation details of our value decomposition network, they were exactly the same as the independent Q-Learning; apart from training the agent Q networks using the loss associated with their sum. We also only updated the target every 128 steps. The other parameters are: batch size$= 64$, buffer size$= 512$.

Qualitatively, in the last observable episode the agents spread out from each other but then begin to move closer to the landmarks. They are unable to get close to the target within the allotted maximum steps in an episode.

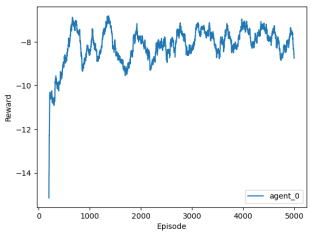
4.4 QMIX

For the implimentation of QMIX, to ensure the mixer has monotone weights (optimizing Q_{tot} also necessarily optimizes each Q_i), we used three hyper networks with absolute activation functions as the hidden layers in the neural network, followed by ReLU activation. The size of the hidden dimension was set to 24. The other parameters are: batch size$= 64$, buffer size$= 512$, target update frequency$= 1284$a.

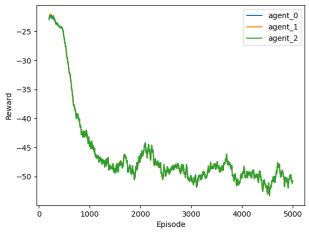
Qualitatively, we find that the agents do exhibit cooperative behavior. Here we find that two agents approach the landmarks compared.

5 Discussion and Extensions

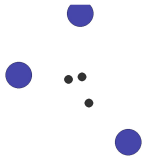
Neither QMIX nor VDN were able to learn and consistently converge in improving the reward. The 'spread' task was more difficult than anticipated; although we saw it as sort of a prerequisite before we could hope to achieve strong performance in mixed competitive/cooperative environments. In particular, it is due to of non-stationarity as the actions of other agents also impact the environment. At this stage of the project we chose to survey multi-agent reinforcement learning algorithms so we implemented shallow and mostly linear networks to represent actors, critics, q functions and mixing functions. The authors of QMix [6] utilize GRU layers which enable larger flexibility in capturing



(a) VDN Baseline

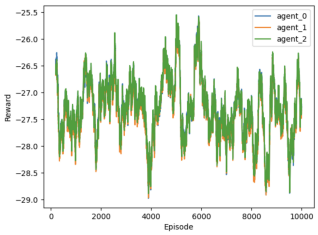


(b) VDN Multi

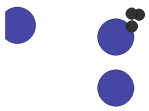


(c) Particle Position at final step at Episode 4000

Figure 3: VDN



(a) QMIX Multiagent



(b) Particle Position at final step at Episode 4000

Figure 4: QMIX

the linearity of the function. Additionally, we can further run the algorithms for longer durations as the agents may learn in later episodes. Testing parameters such as gamma, buffer size, batch size, epsilon, and target update frequency on the results may also be desired.

In each approach to tackling the multi-agent setting, we find that the average reward is not an appropriate metric to measure a desired global outcome. The results between individual learners and

one with shared critics have similar total rewards yet different behaviors when observed qualitatively. This outcome may suggest that the problem is under-specified and that custom alterations in the reward function are needed.

6 Code

Our implementations of each algorithm can be found in the github: `https://github.com/aretnanto/MultiAgentRL/tree/main`. We used the Pytorch and PettingZoo repositories, otherwise implementing everything ourself.

References

[1] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[2] Mathieu Le Goc, Lawrence H. Kim, Ali Parsaei, Jean-Daniel Fekete, Pierre Dragicevic, and Sean Follmer. Zooids: Building blocks for swarm user interfaces. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, page 97–109, New York, NY, USA, 2016. Association for Computing Machinery.

[3] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

[4] Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning, 2021.

[5] Ken Nakagaki, Jordan L Tappa, Yi Zheng, Jack Forman, Joanne Leong, Sven Koenig, and Hiroshi Ishii. (dis)appearables: A concept and method for actuated tangible uis to appear and disappear based on stages. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[6] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning, 2018.

[7] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017.

[8] Justin K. Terry, Benjamin Black, Ananth Hari, Luis S. Santos, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, Caroline Horsch, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. *CoRR*, abs/2009.14471, 2020.