

Cleaning Agents with DeepQLearning

A performance comparison between MLP and RNN networks

Filippo Cavallari filippo.cavallari2@studio.unibo.it

Department of Computer Science and Engineering - DISI
Alma Mater Studiorum – University of Bologna

Intelligent Systems Engineering

20/11/2023



- 1 Background
- 2 Project



MARL systems [1] I

First type of division

- *Cooperative*
 - Homogenous
 - Heterogeneous
- *Competitive*
- *Mixed*

Second type of division

- *Centralized training centralized execution*
- *Decentralized training decentralized execution*
- *Centralized training decentralized execution*

Deep Q Learning I

In DQL, the Q-function is represented as a deep neural network. This network takes the current state s as input and outputs a Q-value for each possible action a . The agent selects the action with the highest Q-value, guiding its decision-making process [3].



Deep Q Learning II

Algorithm

- Initialize Q-network and target network with random weights.
- Initialize replay buffer.
- For each episode:
 - Observe the current state s .
 - Select an action a using ϵ -greedy policy.
 - Execute action a , observe reward r and next state s' .
 - Store the experience (s, a, r, s') in the replay buffer.
 - Sample a mini-batch from the replay buffer.
 - Calculate the target Q-values using the target network.
 - Calculate the loss between predicted and target Q-values using the Bellman equation.
 - Update the Q-network weights using backpropagation.
 - Periodically update the target network weights.
 - Repeat until convergence or a predetermined number of episodes.

Deep Q Learning III

Applications of Deep Q Learning

- **Atari Games:** DQL achieved human-level performance on various Atari 2600 games, demonstrating its capability to learn complex strategies.
- **Robotics:** DQL has been used to train robots for tasks like object manipulation, navigation, and control.
- **Autonomous Vehicles:** DQL is employed in training autonomous vehicles for safe and efficient navigation.
- **Recommendation Systems:** It is utilized to optimize recommendations for users on online platforms.
- **Healthcare:** DQL is applied for optimizing treatment plans and resource allocation in healthcare.

VMAS I

VMAS is a sophisticated framework tailored for Multi-Agent Reinforcement Learning (MARL) applications. It offers a unique blend of features and capabilities, making it a valuable tool for researchers and practitioners in the field [2].

Main components

- Environments
- Scenario
- Agents
- Targets
- Sensors (e.g. LIDAR)

VMAS II

Tensors structure

The tensor structure is described as follows:

$$X \in \mathbb{R}^{I \times J \times K}$$

Where:

I : The number of environments

J : The number of agents

K : The dimension of the agent observation

Cleaning Agents I

Goal

The primary goal of the agents in this scenario is to efficiently remove all M targets from the 2D space. To successfully remove a target, an agent must be in close proximity to it, ensuring effective cleaning. This proximity requirement adds complexity to the task, as agents must navigate and coordinate to reach and clean targets efficiently.



Cleaning Agents II

Agents

- Agents do not know the absolute location of the targets
- Agents use a LIDAR to detect targets inside a specific range
- Agents can move in eight direction (N, W, E, S and in diagonal)

Targets

- Targets are static, they don't move
- Targets have a radius of interaction
- When an agent moves inside the target radius the target is deleted



Cleaning Agents III

The reward function in the "Cleaning Agents" scenario is designed to encourage agents to make optimal decisions while cleaning the targets. The reward function operates as follows:

- 1 When an agent's distance from a target falls below a predefined threshold (K), indicating successful cleaning, the agent is rewarded positively: $\text{Reward} = 1 + \text{number of previously removed targets}$
- 2 If an agent's LIDAR sensor does not detect any targets in its vicinity, it receives a negative reward (-1): $\text{Reward} = -1$
- 3 When an agent's LIDAR sensor detects a target, and the distance from the agent's previous position to the target decreases, the agent is rewarded based on the function: $\text{Reward} = \frac{-\text{LIDAR range}}{x}$ (normalized to fall within the range of -0.5 to 0)

Cleaning Agents IV

- ④ Conversely, if the distance from the agent's previous position to the target increases, indicating a suboptimal move, the agent is penalized based on the same function: $\text{Reward} = \frac{-\text{LIDAR range}}{x}$ (normalized to fall within the range of -1 to -0.5)



Custom Implementation for LSTM RNN Networks I

How do you address the LSTM networks requirement for temporally consecutive states?

The solution is to extend the functionality of the Replay Buffer: in my modified setup, states are no longer solely composed of LIDAR measurements but are now augmented with the action taken by the agent and the corresponding reward received.



Custom Implementation for LSTM RNN Networks II

New use of the ReplayBuffer

To feed sequences of states into the LSTM network, I require N consecutive states. I accomplish this by selecting the $N-1$ states stored in the Replay Buffer, which were observed before the current state. The current state, however, does not yet possess an associated action and reward, as it awaits processing by the behavioral policy. To accommodate this, I initialize tensors of the appropriate shape, filled with zeros, representing no action performed and no reward received.



Custom Implementation for LSTM RNN Networks III

Generating Sequences for Network Improvement

To train the LSTM-based DQL effectively, I need to generate batches of sequences that preserve the temporal structure of the data. This ensures that the LSTM can learn and generalize from the sequential patterns in the state-action-reward sequences. To achieve this, I employ a two-step process.

- First, I randomly select a BATCH SIZE number of lists of consecutive random indices from the Replay Buffer.
- Secondly, these indices are used to retrieve the corresponding states from the Replay Buffer, forming sequences that encapsulate the temporal progression of the agent's experiences.

DQL implementation for MLP networks

State representation

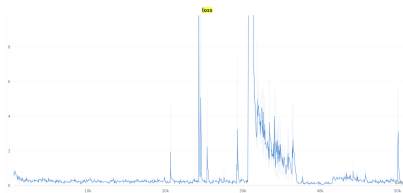
The state of each agent is represented as a tensor of shape $[50, 1]$. This choice is informed by the fact that each agent's LIDAR sensor in VMAS emits 50 rays, returning float distances. The final tensor also includes a dimension for the environment ID: $[NENV, 50, 1]$.



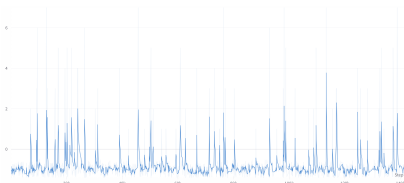
MLP results I



(a) Total reward after 50 epochs



(b) Total loss after 50 epochs



(c) Total reward after 150 epochs



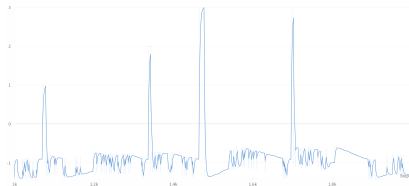
(d) Total loss after 150 epochs

MLP results II

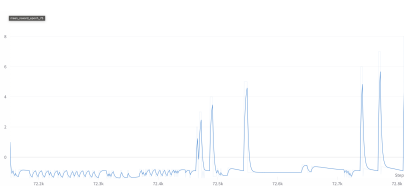
Figure: Reward and Loss comparison



MLP results III



(a) Mean Reward Epoch 1



(b) Mean Reward Epoch 70

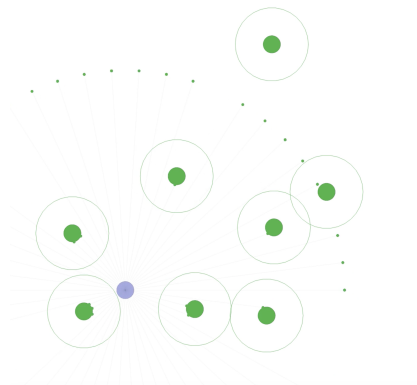


(c) Mean Reward Epoch 145

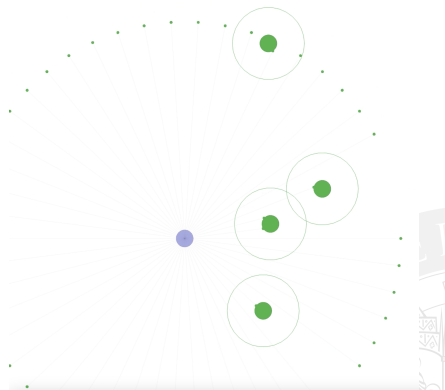
Figure: Mean rewards



MLP results IV

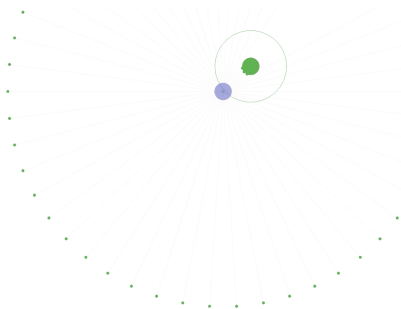


(a) Beginning of simulation



(b) Half simulation

MLP results V



(c) End of simulation

Figure: Different stages of a simulation with one agent and 8 targets

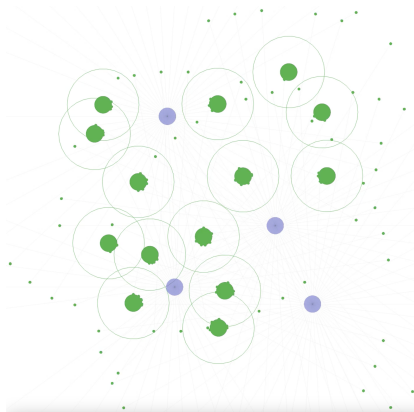
MLP results VI

Evaluation with one agent

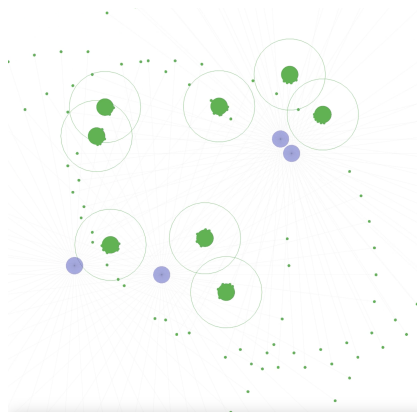
To assess the agents' performance in practical scenarios, I conducted evaluations with varying numbers of agents and targets. In a scenario involving one agent and eight targets [3], the results indicated the following:

- On average, it took approximately 10 steps for the agent to remove the first target, reflecting a quick initiation of the cleaning process [2a].
- Removal of half the targets occurred, on average, in approximately 170 steps, demonstrating consistent progress [2b].
- To remove all eight targets, agents required an average of around 1440 steps, highlighting the complexity of clearing the entire environment [2c].

MLP results VII

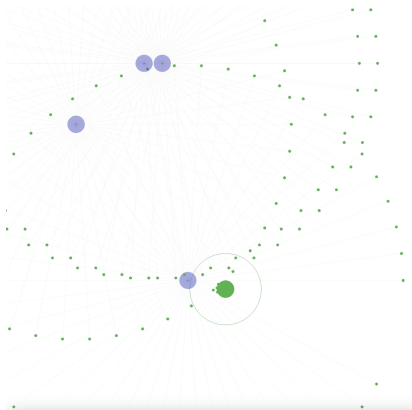


(a) Beginning of simulation



(b) Half simulation

MLP results VIII



(c) End of simulation

Figure: Different stages of a simulation with four agent and 14 targets

MLP results IX

Evaluation with many agents

In a more challenging scenario with four agents and fourteen targets [4], the outcomes were as follows:

- Agents demonstrated improved collaboration, taking an average of about 400 steps to clear all fourteen targets.
- The first target was removed in an average of 10 steps, emphasizing swift task initiation.
- Removal of half the targets was achieved in approximately 240 steps, showcasing efficient teamwork among agents.

MLP results X

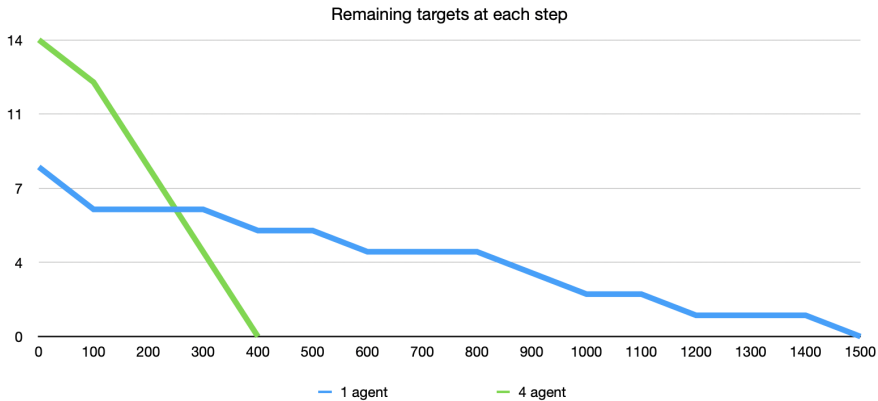


Figure: Remaining targets at each step

Cleaning Agents with DeepQLearning

A performance comparison between MLP and RNN networks

Filippo Cavallari filippo.cavallari2@studio.unibo.it

Department of Computer Science and Engineering - DISI
Alma Mater Studiorum – University of Bologna

Intelligent Systems Engineering

20/11/2023



References I

- [BBD08] Lucian Busoniu, Robert Babuska, and Bart De Schutter. “A Comprehensive Survey of Multiagent Reinforcement Learning”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), pp. 156–172. doi: 10.1109/TSMCC.2007.913919.
- [Bet+22] Matteo Bettini et al. “VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning”. In: *The 16th International Symposium on Distributed Autonomous Robotic Systems* (2022).
- [Mni+13] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. url: <http://arxiv.org/abs/1312.5602>.