City, University Of London

MSc Data Science

INM707 Deep Reinforcement Learning

Lahcen El Hayani

Antonio Jose Lopez Roldan

1. Define an environment and the problem to be solved.

The purpose of this project is to develop and evaluate a Q-learning algorithm. We consider a robot (agent) that will navigate across a 10x10 virtual grid space, to deliver items to the designated points. Since our main goal is to understand how the agent learns to find the best way within the maze space, we are only considering the robot's movements, hence there won't be any transportation of items.

Given our case, we aim to maximise the efficiency of the agent by finding itself the fastest route from any random allocation to the delivery points. The reason behind is that we are looking for the quickest route from picking any item from the shelves to be delivered to the delivery points.

For extra guidance and experimentation with the different parameters, apart from the labs and lecture we use the book "Reinforcement Learning: An Introduction" [1].

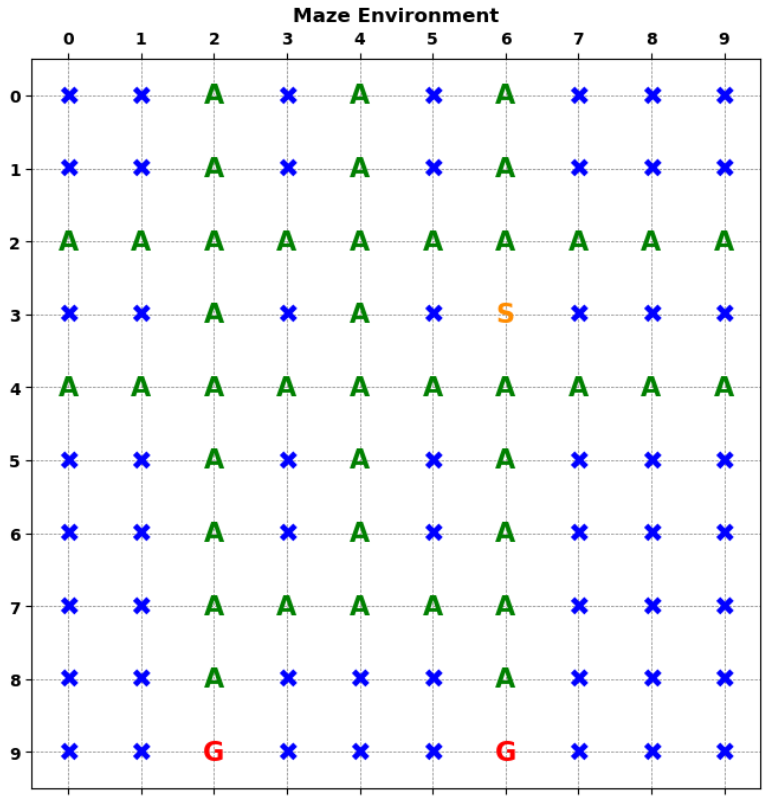
Environment description:

a. Grid layout: A10 x 10 matrix represents the space, this space includes paths, obstacles, and delivery points.

b. Delivery points: The environment has 2 fixed points located at (0, 9) and (9, 9). These fixed points represent the delivery points and the robot's goal is to reach them. Delivery points are represented by the "red G".

c. Obstacles: The obstacles of our environment mimic shelves and the warehouse hallway is the space where the robot can move through.

Figure 1. Maze environment, example 1.



d. Starting points: The robot's starting positions are randomly allocated at the beginning of each episode. This variation aids thoroughly in modelling the best path to the delivery points within the environment.

e. Actions, The robot can move:

- Up
- Down
- Left
- Right

Any move that leads into walls or exceeds the grid boundaries, results in penalties and is counted as "invalid".

Specifically:

- "Green A" represents the space where the robot is allowed to move.
- "Blue X" represents obstacles.
- "Orange S" represents the starting point.
- "Red G" represents the goals, and delivery points.

2. Define a state transition function and the reward function.

State transition function

In Reinforcement Learning, how likely it is for an agent to move from one state (to another is mostly determined by what we call the state transition function. This function updates the agent's current state based on the action it chooses. Mathematically, we can represent this function as $s_{t+1} = \delta(s_t, a_t)$ where a_t is the action the agent decides on (up, down, left, right) and s_t the state where the agent currently is (s_{xy}).

The aforementioned function is a generalization of all the states our agent can experience. In this line, the agent has 44 possible states (green A + red G). The agent starts at a random location s_{xy}, for instance s_{63} like in the image we are sharing and to reach the goals can move, lef s_{53} or right s_{73} for which moves would be penalized, up s_{62}, or down s_{64} for which case would get the higher reward as that is the direction towards the goal.

Reward Function

The reward function is designed to encourage the warehouse robot to find the shortest path from a random starting point. The reward function assigns positive rewards and penalties defining the robot behaviour. For our environment and the presented problem the following are our rewards/penalties:

- Rewards:
    - Fast Delivery: The robot receives a reward of +100 for reaching the goal quickly.
    - Slow Delivery: The robot receives a reward of +50 for reaching the goal at a slower pace.
    - Fast Delivery Limit: Is a threshold that define the limit between fast/slow delivery.
- Penalties, we allocate three kinds:
    - Hit Obstacle: The robot receives a penalty of -15 for hitting an obstacle (shelves).
    - Move Penalty: The robot receives a penalty of -1 in each move to avoid unnecessary moves.

We can generalize the reward function with the following mathematical expression:

$r' = r(s, a)$; Where r is the reward transition function that takes as input the current state represented by s, and the action executed by the robot, represented as a, finally r' is the output for that certain state, action.

3. Set up the Q-learning parameters (gamma, alpha) and policy.

The Bellman equation for updating the Q-value function is presented next:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma * \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Where:

$Q(s_t, a_t)$ represent the current Q-value for state s_t and action A_t; α represents the learning rate, how important are new policies in comparison to the already known; R_{t+1} represent the immediate reward; γ represents the discount factor, how important is the immediate reward in comparison with to the future rewards; $\max_a Q(S_{t+1}, a)$ both represent the maximum Q-value for the next state S' (or St+1) across all possible actions

a. Discount Factor, Gamma

Gamma, represented by "γ " and values between 0 and 1, set how important are the future rewards. If gamma 0, we only care about the immediate rewards and don't consider future rewards at all. This turns the agent into a "short-sighted". When gamma is 1, we give equal importance to immediate rewards and future rewards, to make a decision we consider both equally.

b. Alpha, Learning Rate

Is a ratio used to indicate how much new information is weighted in contrast to old information when updating what the agent knows about the environment. When alpha is 1, we completely override the old knowledge and only use the new information from the most recent experience to make updates. The other way around, doesn't consider new experiences to update the knowledge of the environment.

c. Policy.

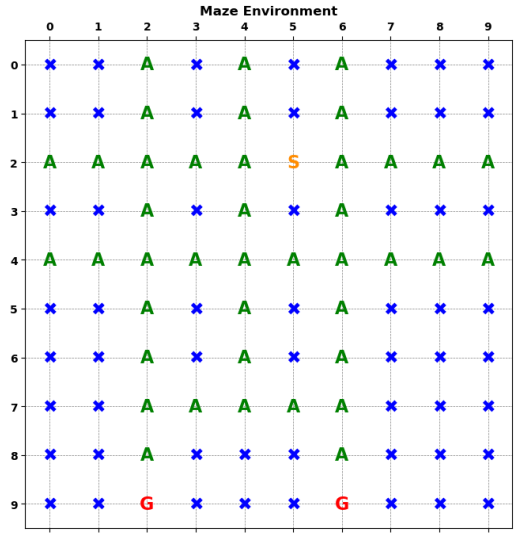
The Epsilon-Greedy policy helps to balance between exploring new options and using what the agent already knows. Epsilon is a value between 0 and 1, a probability. When exploring, agent tries to learn more, whereas when exploiting, tries to make the best decision based on the knowledge stored. At first, the agent explores far more than exploit, then is the other way around.

4. Run the Q-learning algorithm and represent its performance.

In this part, we will run the Q-learning algorithm. We will choose the following configuration:

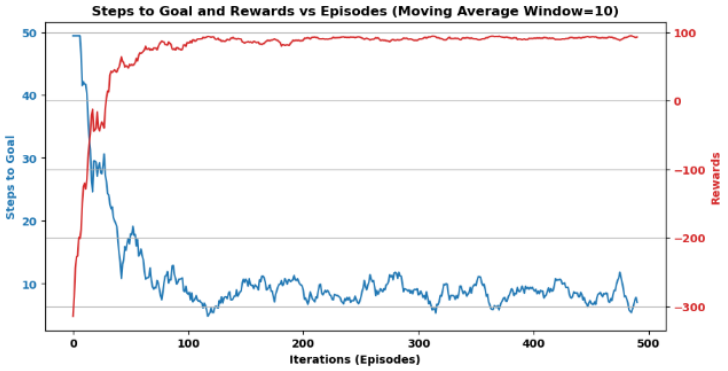
- Number of episodes: 500
- Epsilon: 0.5
- Gamma: 0.8
- Alpha: 0.8

Figure 2. Maze environment, example 2.



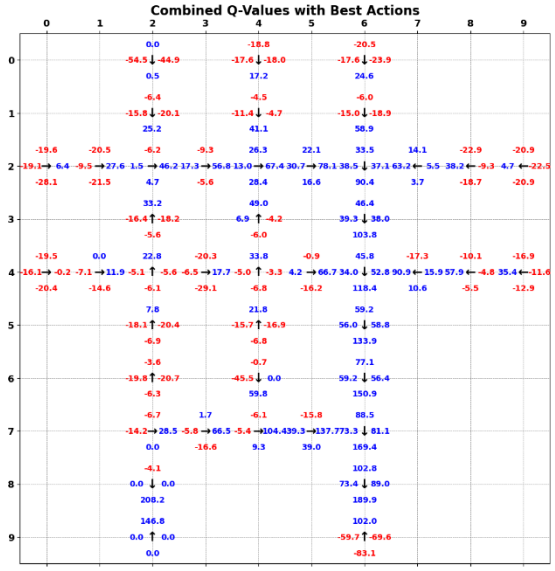
As we said our starting point is different every time. On this occasion, as we see in Figure 2 we start at S_{52}.

Figure 3. Steps to goal/rewards vs epsilon



In Figure 3 we appreciate how the lines converge towards their respective sides. The necessary steps (blue line) to reach the goals diminish considerably until episode 100, then, the line remains around 10 (remember that each time the starting point is different so this is why it might keep varying). The rewards (red line) also converge towards 100.

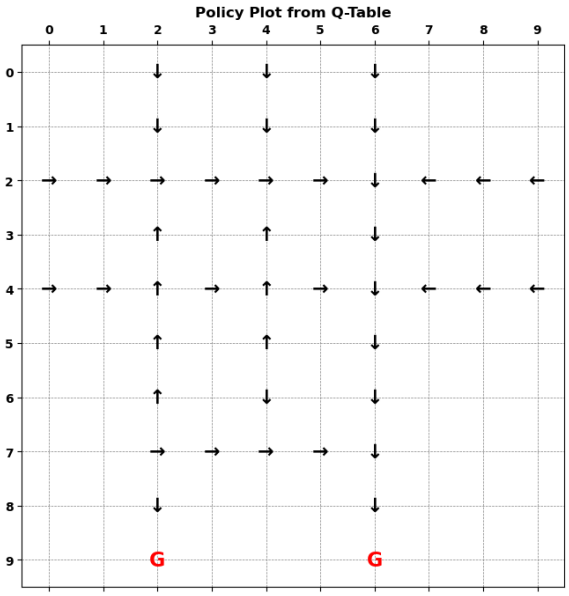
Figure 4. Combined Q-values with best actions.



In Figure 4 we can see what the agent learned. If we pay attention to the blue set of numbers, we see an "imaginary blue path" towards the goal, once the agent reaches the goal, two arrows are facing each other, which indicates the loop reward, so the agent tends to remain at the goal once this has been reached.

Figure 5. Policy plot from Q-table.

Figure 5 represents a pragmatic picture of the best policy learned by the agent. Each Arrow in the grid points in the direction that represents the best action to take from each state according to the learned Q-values. If we follow the arrow at any starting point, we will end up at our delivery point marked as “red G”.



5. Repeat the experiment with different parameter values, and policies.

a. Different parameters

Below, we display the results for the different values applied on epsilon, as well as on Alpha. Best results are for the higher epsilon whereas the lowest shows major variability. Alpha, produces a more stable line with a mild value of 0.5.

Figure 6. Different parameters for epsilon.

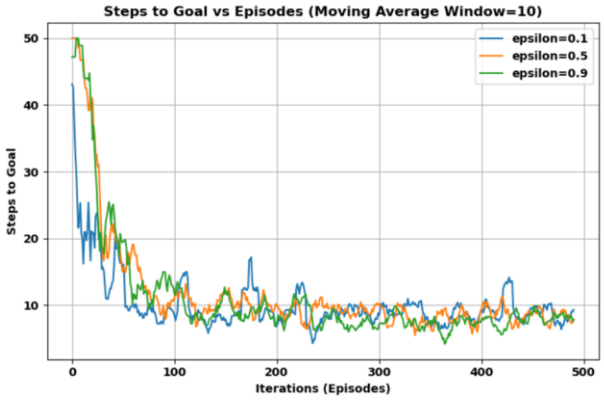
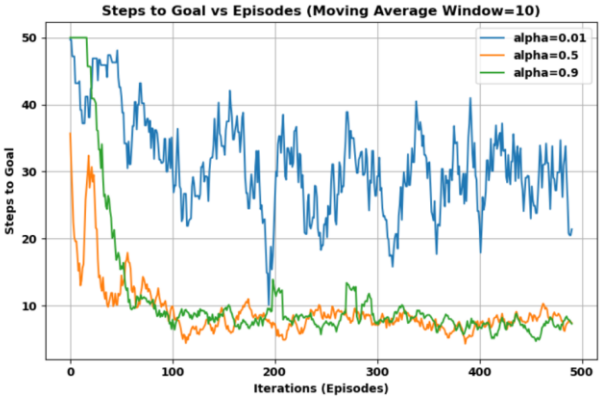
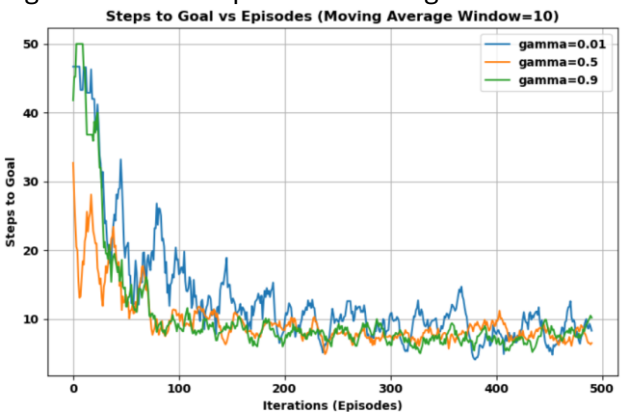


Figure 7. Different parameters for Alpha.



Gamma, below in Figure 8, has the most stables output in the long run with a value of 0.5, with 0.01 produces high variability.

Figure 8. Different parameters for gamma.



b. Different Policies

Below we show 3 different polices the agent learned after starting from different points. Each of the charts represents the policy plotted with the score for each action.

Figure 6. Starting at S_{47}

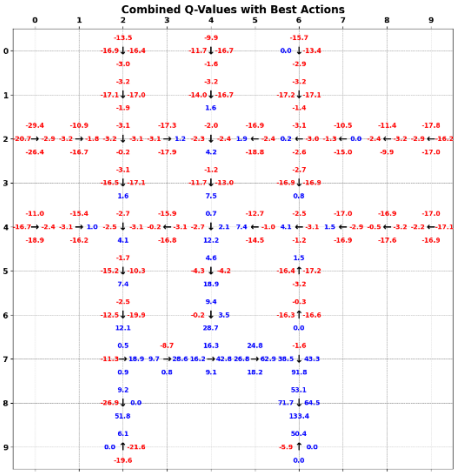


Figure 7. Starting at sS_{28}

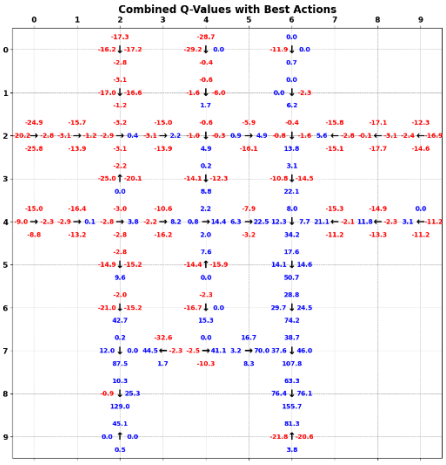
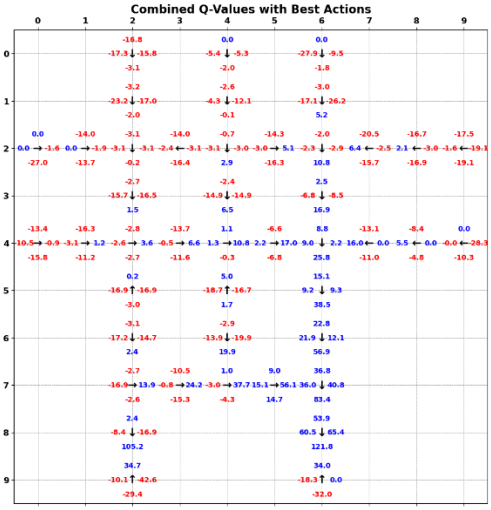


Figure 8. Starting at S_{40}



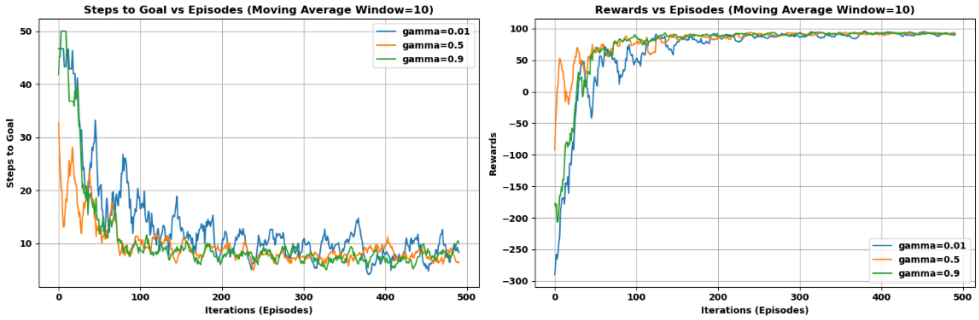
6. Analyze the results quantitatively and qualitatively.

a. Analysis of Gamma

In Figure 9 we can see the different behaviour in the lines drawn by the different values for Gamma, next, we show the reward values gained by the agent across the episodes. Let's underline that in our analysis the reward values gained by our agent over time are relevant in the analysis of our results, as the agent might need to take a different amount of steps to reach the goal not because hasn't learned a stable policy but because our agent started every time at a different point, and that is why the number of steps/episodes is more volatile than the rewards/episodes after a certain amount of steps. Said that:

- The number of steps taken to reach the goal decreases as increases the number of episodes, this means that the agent is learning and becoming more efficient. Comparing the different gamma values, we confirm the already theory seen, the higher the values for Gamma the more steady and fastest the agent reaches a lower number of steps/episodes.
- This suggests that a higher discount factor allows the agent to consider future rewards more effectively, leading to better performance and higher cumulative rewards.

Figure 9. Gamma; to goal/episodes and rewards/episodes.

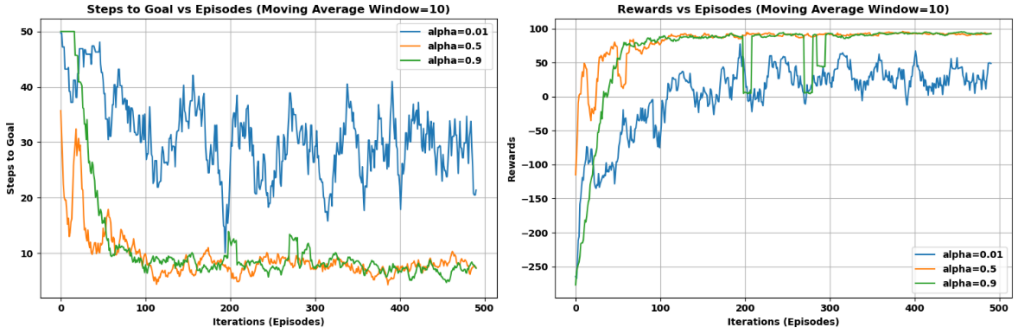


b. Analysis of Alpha

The optimal learning rate depends on the environment, it is represented by alpha. From Figure 10, below, we conclude:

- The high variability in the steps to goal/episodes reflects the struggle of the agent to learn from the environment.
- The best behaviour seems to be achieved by an Alpha = 0.5 with a rapid descent in the ratio steps/episode and a solid increase in the reward/episode ratio.

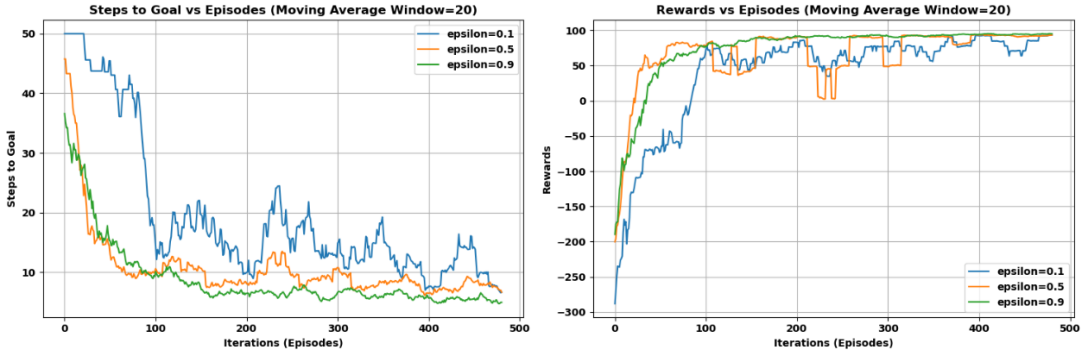
Figure 10. Alpha; Steps to goal/episodes and rewards/episodes.



c. Analysis of Epsilon

Figure 11 demonstrates how a medium value offers the best balance between exploring new actions and already known strategies. The lowest value (high exploitation) doesn't benefit to the agent in our environment, it takes extra steps until drastically decreases the steps needed to reach the goals, although with a high variability. In this case, seems that exploring>exploiting seems to produce the best of the results.

Figure 11. Epsilon; Steps to goal/episodes and rewards/episodes.



Advanced Part

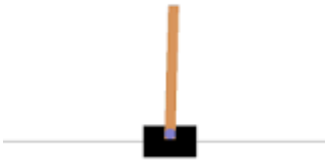
7. Implement DQN with two improvements.

Deep Q-Network (DQN)

Essentially, DQN swaps the traditional Q-table for a neural network. Rather than associating a state-action pair with a Q-value through a simple table, a neural network is employed to link these pairs with their respective Q-values. This setup involves two neural networks that share the same design but hold different weights. In every "N" step, the values from the primary network are transferred to the secondary network, known as the target network.

CartPole-v1 Environment Description

Figure 12. Cart Pole.



In the CartPole-v1 game, Figure 12, the goal is to prevent a pole from falling over by moving a cart left or right. The game measures how well you are doing by counting how long the pole stays upright.

a. Motivate your choice and your expectations for choosing a particular improvement.

Standard DQN tends to overestimate Q-values because of the max operation used in the Bellman equation to update these values. This can make the agent overly optimistic, leading it to believe some actions are better than they actually are. Double DQN, or DDQN, addresses this issue by splitting the process into two steps: one to choose the best action and another to accurately evaluate how good that action really is. This separation helps make the learning process more reliable and prevents the agent from being too optimistic about the value of its actions.

b. Apply it in an environment that justifies the use of Deep Reinforcement Learning.

Why we chose it: Dueling DQN uses a special trick where it breaks down its predictions into two parts: one part that estimates the general quality of being in any given situation, and another part that evaluates the additional benefit of each possible action in that situation. This helps it make more nuanced decisions, especially in scenarios where the choice of action doesn't matter much.

8. Analyse the results:

Baseline DQN Results: Initially, the robot had a good learning curve, and kept the pole straight for a relatively long time. However, after a while, the performance was variable. This might be indicative of the agent getting confused at times due to its overestimation of how good some actions truly are.

Double DQN Results: The results with DDQN showed fewer variability. The trend line (moving average) in DDQN results often showed more stability compared to the baseline DQN. This likely means that by splitting the decision-making process into two parts, the robot didn't overestimate the value of its actions as much, leading to more stable learning.

Dueling DQN Results: With Dueling DQN, the performance was generally steadier. This improvement suggests that understanding the value of just being in a particular situation (regardless of the action taken) helped the robot to make better decisions, leading to keeping the pole upright more consistently.

Conclusion

We tested two modifications to the standard DQN approach to see if they would help the robot in learning better the Cart Pole game.

- Double DQN was used to try to stop the robot from being too optimistic about its actions.
- Dueling DQN was used to help the robot understand better when it's in a good situation or when an action might really make a difference.

The experiments showed that these methods could help stabilize and improve the robot's performance, making it a smarter learner.

9. Apply the RL algorithm of your choice (from rllib) to one of the Atari Learning Environments.

Deep Q-Network (DQN) combines Q-learning with deep neural networks. Being more specific, It learns what is the optimal decisions by estimating the Q-values of the state-action pairs using deep neural networks.

a. Briefly present the algorithm.

- I use a default configuration of the DQN algorithm in RLlib, including experience replay. (The agent's experiences in a buffer and samples from it during training).
- DQN uses a separate target network to estimate the Q-Values of the next states. The target network is updated with the weights of the main network.
- Epsilon-Greedy Exploration, DQN employs an epsilon-greedy exploration. The agent takes random action with probability epsilon and the best action according to the Q-values with pb 1-epsilon.

b. Justify your choice

- DQN has human-level performance in Atari games. It is well-established.

- DQN can handle the raw pixel input of Atari games by utilizing deep neural networks. Hence the agent can learn directly from the “screen”.
- Finally, it relatively has a good sample efficiency compared to other methods so we can obtain a better understanding of the agent according to the different parameters values.

10. Analyse the results quantitatively and qualitatively

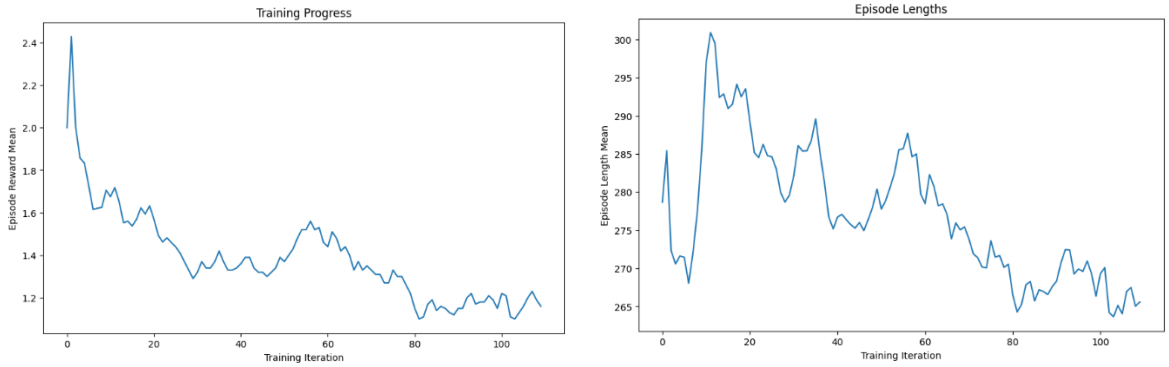
Quantitative Analysis

- For the **episode reward mean**, initially peak and later decline. This peak suggests that during the early exploration, the agent found some good actions. It reaches a max of 2.4 and later drops to a minimum of under 1.2 in the late training period, getting stabilised after the 60^{Th} iteration around the value of 1.4 the stabilization could mean that the agent converged to a certain policy, althouth not the best one.
- For the **episode length mean**, although at the beginning there is a noticeable variation from 270 to 300 then consistently mark lower values until in the end seems to be around 270 to 265. This is a reflection of the agent exploring different activities, the different strategies that the agent applies lead to that variation in the chart.

Qualitative Analysis

- For the **episode reward mean**, seems that the agent is capable of learning somehow due to the stabilization. Without any hesitation, there is room for improvement in terms of both efficiency and effectiveness in the learning process. The gradual decrease in the episode reward mean is a clear sign that supports that.
- For the **episode length mean**, the behaviour in this chart shows a need to tweaking the exploration/exploitation balance as indicates the fluctuations and initial high rewards followed by a decrease.

Figure 13. DQN, ATARI results. Epsilon reward mean, episode length mean, vs iterations.



[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. in Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.
[2] M. Hessel et al., 'Rainbow: Combining Improvements in Deep Reinforcement Learning', AAAI, vol. 32, no. 1, Apr. 2018, doi: 10.1609/aaai.v32i1.11796.
[3] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236
[4] PriyankaVelagala, "Deep-reinforcement-learning-projects/atari-breakout/breakout_model_training.ipynb at main · Priyankavelagala/deep-reinforcement-learning-projects," GitHub, https://github.com/PriyankaVelagala/Deep-Reinforcement-Learning-Projects/blob/main/Atari-Breakout/Breakout_Model_Training.ipynb (accessed May 1, 2024). Special note, the code is based on this piece of work, yet crafted and updated by our own based on the rllib website.
[5] "RLlib: Industry-grade reinforcement learning#," RLlib: Industry-Grade Reinforcement Learning - Ray 2.21.0, https://docs.ray.io/en/latest/rllib/index.html (accessed May 12, 2024).