Cairo University

Faculty of Engineering

Computer Engineering Department

CMPS458 - Reinforcement Learning

Assignment 2

Team Name/Number:

Abdelrahman Tarek

Ziad Hesham

Supervisor: Ayman AboElhassan

November 12, 2025

Deliverables

Repository Link: GitHub Repository
Video Record Link: Google Drive - Videos
Weights & Biases Graphs: Google Drive - Weights & Biases Figures

Experiment Summary

During this experiment we witnessed a lot of the strengths and weaknesses of the **DQN** and **DDQN** approaches to reinforcement learning, they generally reduced the necessary storage for bigger problems as we just store weights and biases of our model currently. They do have flaws especially when it in comes to handling complex long sequence of actions problems as well as continuous problems. They do however, handle simpler discrete problems pretty well. We noticed that increasing the number of weights for our Q-Networks does increase training time a lot however it tends to provide more refined results. Another major factor that affected training is the amount of episodes we train for and the limit for steps to be taken during a training episode. A higher step limit does allow us to try longer sequences of actions however it heavily increases our training times.

Question Answers

Question 1: What is the difference between DQN and DDQN in terms of training time and performance?

As shown in Figure 1, the training time difference between the **DDQN** and the **DQN** is not that significant (8m30s vs 10m), however in the following question, we can clearly see that the DDQN provides more stable runs on average so it is a performace gain with minimal cost.

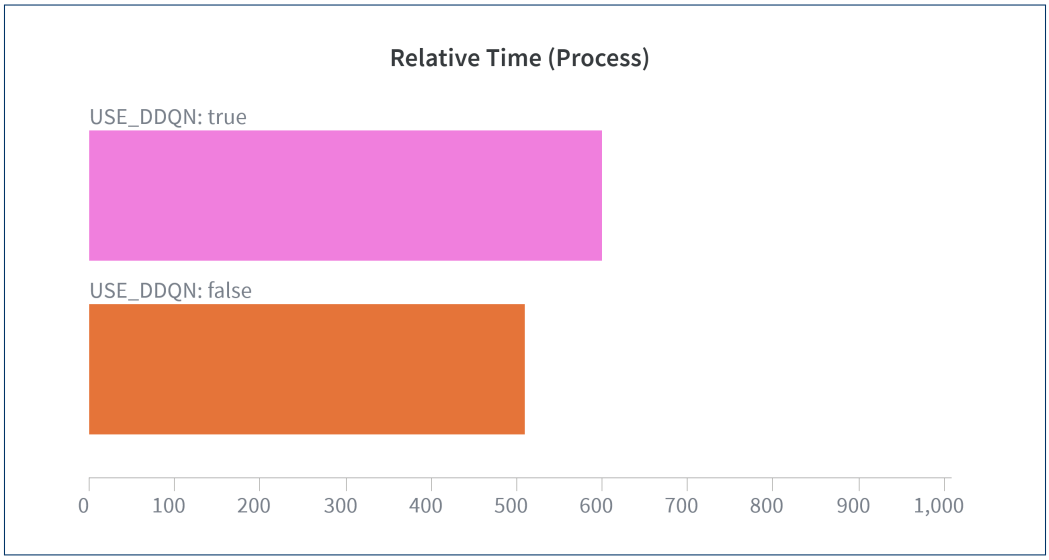

Figure 1: ***DDQN*** *vs* ***DQN*** *Average Training Times.*

Question 2: How stable are the trained agents?

The agents' stability varies heavily depending on 2 things, being **DQN/DDQN** and the nature of the problem, down below is each figure showing the rewards per episode over 100 episodes for each environment showing the difference in stability for each agent over each environment.

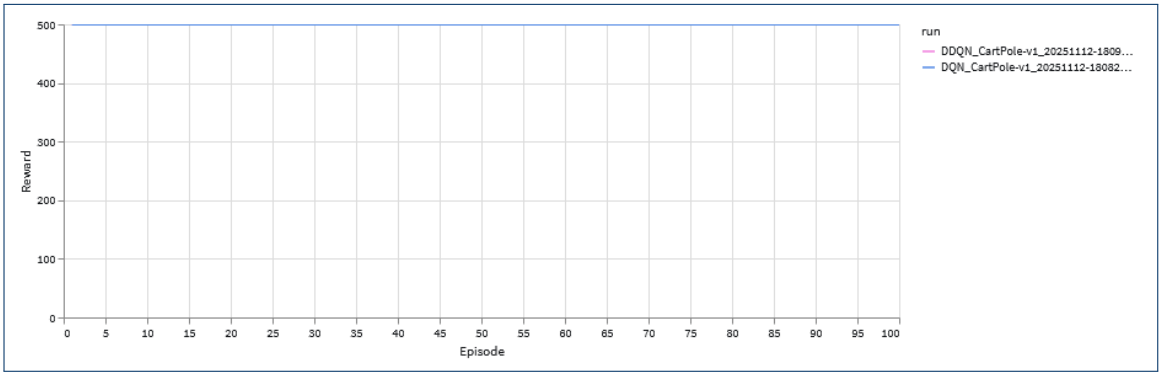

Figure 2: *Reward/Episode Graph for the Cart-Pole Environment.*

As shown in Figure 2, Since this environment is easy both agents succeed all the time so they are perfectly stable.

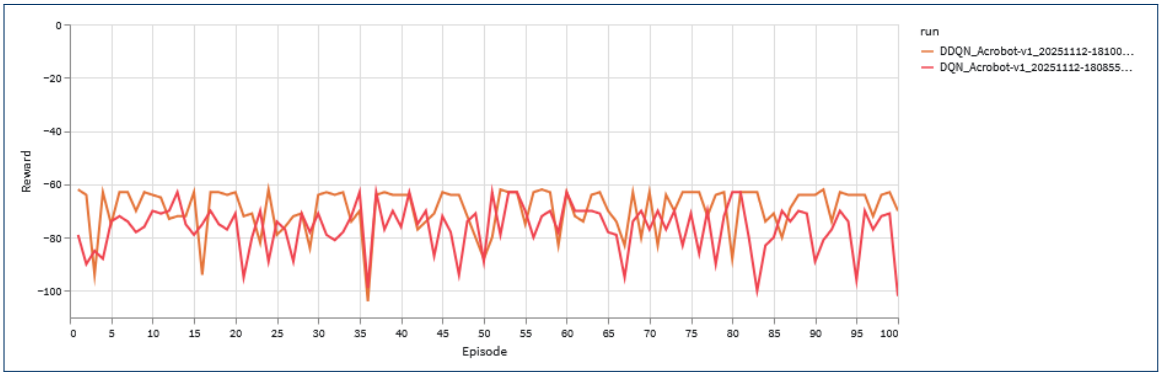

Figure 3: *Reward/Episode Graph for the Acrobot environment.*

As shown in Figure 3, This environment is more challenging, showing instability for both agents. However we can notice that the **DDQN** agent is more stable on average, having an overall lower standard deviation than its **DQN** counterpart.

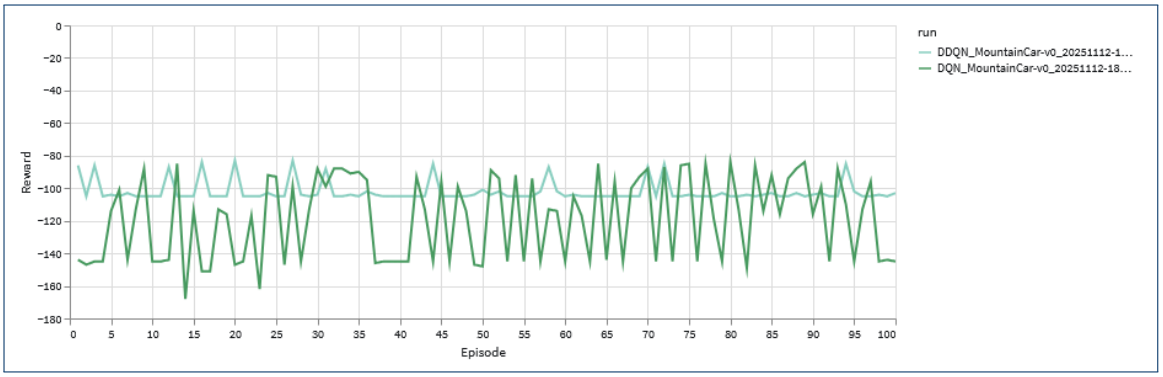

Figure 4: *Reward/Episode Graph for Mountain-Car Environment.*

As expected in Figure 4, we found that DDQN was more stable and the average rewards were closer to each other than the DQN.

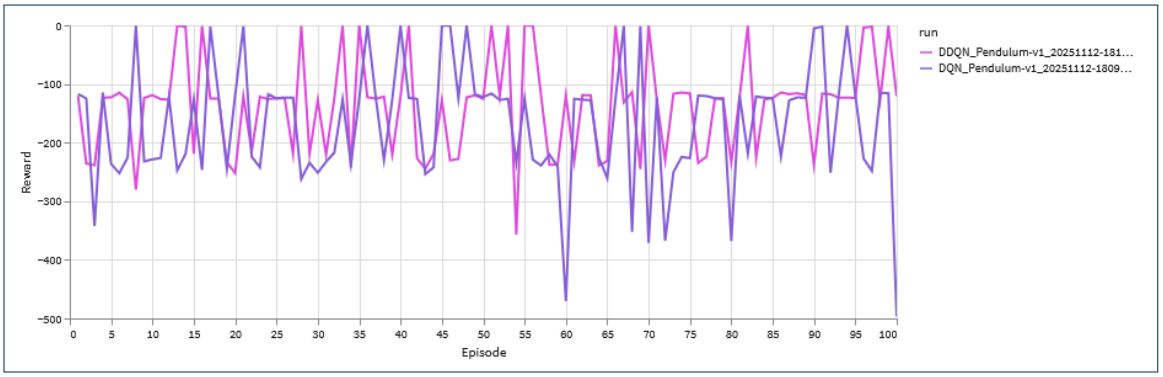

Figure 5: *Reward/Episode Graph for Pendulum Environment*

We noticed that in Figure 5, since the environment which was continuous and then discretized; even though DDQN is more stable than the DQN, its still not producing the best results.

Question 3: What is the effect of each hyperparameter value on the RL training and performance?

During training and evaluation we got to try many different hyperparameters, trying to fit each case for getting proper results. Down below are some of our observations on how each hyper paramter affected both our training and also our results

Table 1: *Hyperparameter Trade-offs in RL Training*

Hyperparameter	Value / Setting	Effect on Training & Performance
Discount Factor (γ)	High (e.g., 0.99)	The agent is "patient" and values future rewards highly. This is good for complex problems where the reward comes at the end.
	Low (e.g., 0.9)	The agent is "short-sighted" and prioritizes immediate rewards.
Epsilon Decay Rate	Fast Decay	The agent stops exploring and starts exploiting very quickly. Good for simple problems like Cart Pole, bad for problems that need heavy exploration like Mountain Car.
	Slow Decay	The agent explores for a long time. This can help find a better overall policy but can make training take much longer.
NN Learning Rate	Too High	The Q-network updates are too large, making training unstable. The reward plot will likely be "spiky" and may never converge.
	Too Low	Training will be very slow, as the network weights change only tiny amounts with each step.
Replay Memory Size	Large Memory	Helps break correlation between sequential steps, leading to more stable training. May keep old, "bad" experiences which slow down later learning.
	Small Memory	The agent learns mostly from recent experiences. This can make it adapt faster but can also lead to instability or "catastrophic forgetting".
Learning Batch Size	Large Batch	Provides a more stable, accurate gradient for the network update, but each update takes longer to compute.
	Small Batch	Updates are faster and "noisier." This noise can sometimes help the agent escape local optima, but it can also make training unstable.

Question 4: Explain from your point of view how well suited DQN/DDQN is to solve this problem.

The first and second examples (**Cart-Pole, Acrobot**) have discrete actions by nature, so they are well suited for being solved by **DQN** and **DDQN** problems 3 and 4. However, (**Mountain-Car, Pendulum**) both face difficulties being solved using these methods. Problem 3 has the issue that its rewards are always negative except when it reaches the goal, which causes the agent to struggle to find the correct sequence of actions since its pretty long and for the most part all actions seem to lead to the same negative reward. Problem 4 faces a different problem, it does not have a discrete action space, but rather a continuous one. This would not work given that **DQN** and **DDQN** are made to handle discrete problems. This issue can be solved by discretising the action space which leads to the problem becoming solvable, albeit not having the best possible results.