CSE 4/546: Reinforcement Learning
Assignment 2 - Deep Q-Networks

Vignesh Prakash
UBID : 50478782
prakash8@buffalo.edu

Mohit Sai Aravind Nunna
UBID : 50468322
mnunna@buffalo.edu

Part 2

1 Benefits

1.1 Using Experience Replay in DQN

Using experience replay helps in sampling efficiency by allowing agent to learn from past experiences. It also helps learning from rare events as these rare events get stored in replay buffer. It also helps in improving stability. Modifying the size of replay buffer can also influence the results of learning process. A large replay buffer allows the agent to store more experiences and learn from more diverse set of experiences, which can improve quality of learned policy. However increasing the replay buffer too much can require more memory and computation time.

1.2 Introducing Target Network

1. Improvement in stability: By introducing a target network, the correlation between the target Q-values and the predicted Q-values is reduced, thus stabilizing the training process.
2. Convergence : By using Deep Q Network, the agent can learn to find the optimal Q-values more accurately, thus improving the convergence
3. Reduces Overestimation : As the Q-values of the target network are learned using the same network, the target network helps to reduce overestimation of the Q-values. The rate at which the target network is updated can also affect the performance of the DQN. Updating too often can lead to instability and updating infrequently can slow down the convergence. Thus, the frequency of updating the target network has to be done for every few thousand steps.

1.3 Representing the Q Function as $\hat{q}(s, w)$

1. Flexibility : For large state spaces and complex dynamics, Neural networks can be used as it is very flexible and can accurately learn to represent complex functions
2. Generalization : By using a Neural network, DQN algorithm can map the states to Q-values and thus can generalize its model to new and unseen states

2 Environments

2.1 Grid Environment

In our project, we have used a basic grid world with the following parameters and objective

2.1.1 Deterministic Environment Parameters

1. **Number of states** : 12
2. **Number of Actions**: 4
3. **Number of Rewards**: 6 (0,10,-1,-0.5,-1.5,-2)

2.1.2 Stochastic Environment Parameters

1. **Number of states** : 12
2. **Number of Actions**: 4
3. **Number of Rewards**: 6 (0,1,2,3,4,-2.5)

In the stochastic environment, the agent moves to the proper position as per the action with only 0.4 probability, and it moves to the other 3 positions at 0.2 probability.

To illustrate this, If right action is provided to the agent, the agent moves right with a 0.4 probability and it might move to left, top, down with a probability of 0.2

2.2 CartPole-v1

CartPole-v1 is one of the Classic Control Environments from the Gymnasium library.

1. The action space of this environments is a discrete set of 2 numbers, indicating the left and the right movement of the cart.
2. The observation space consists of values like Cart Position, Cart Velocity, Pole angle,Pole Angular Velocity.
3. The goal of this model is to stabilize the pole supported in the cart and make it stand upright.
4. The rewards will be +1 as long as the pole stays upright.

2.3 Acrobot-v1

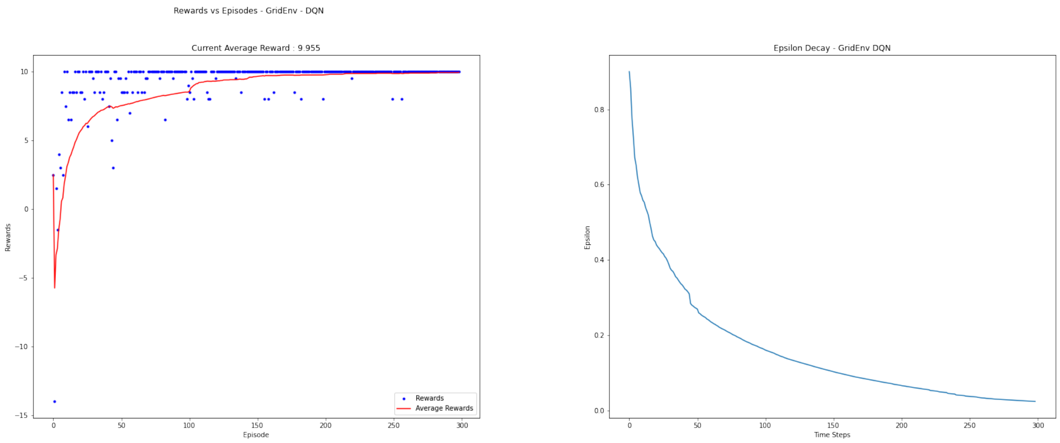
Acrobot-v1 is one of the Classic Control Environments from the Gymnasium library.

1. The action space of this environments is a discrete set of 3 numbers, torque in clockwise direction (-1 Nm), zero torque (0 Nm) and torque in anticlockwise direction (1 Nm)
2. The observation space is a 6 dimensional vector, containing the Cosine of theta1, Sine of theta1, Cosine of theta2, Sine of theta2, Angular velocity of theta1 and Angular velocity of theta2
3. The goal of this model is to make the free end of the acrobat to reach a certain height in as few steps as possible which is constructed as: $-cos(\theta_1) - cos(\theta_2 + \theta_1) > 1.0$
4. The reward function is discrete, if the step doesn't reach a goal, then the reward will be -1.When the free end reaches the target height, the reward will be 0 and the model will terminate. The threshold for the reward is -100 and if the model has an average value of rewards less than -100 would be considered as solved.

3 Training Results for DQN

3.1 Hyper Parameters used for Grid Environment

Epsilon	0.9
Minimum Epsilon	0.001
Epsilon Decay Rate	1000
Memory Size	1000
Learning Rate (α)	1e-4
Batch Size	128
Discount Factor (γ)	0.99
Target Network Update Frequency	100
No Of Episodes	300



(a) Total Rewards vs Episodes

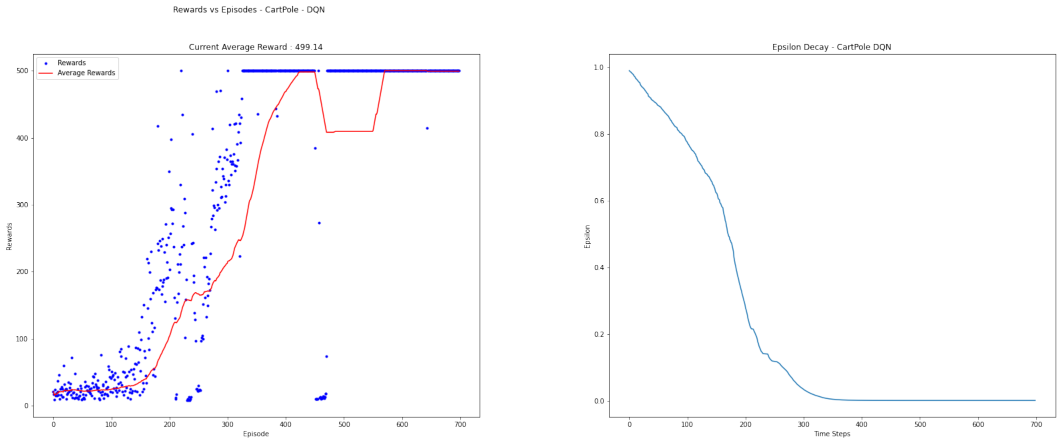
(b) Epsilon Decay

Figure 1: DQN Implementation in Grid Environment

Model converged after 250 episodes with optimal policy learned for given hyperparameters, achieving 9.955 average reward over the last 100 episodes. Low reward values at the first few epsiodes might have been due to low Epsilon decay rate.

3.2 Hyper Parameters used for Cartpole-v1 Environment

Epsilon	0.99
Minimum Epsilon	0.001
Epsilon Decay Rate	10000
Memory Size	5000
Learning Rate (α)	1e-4
Batch Size	128
Discount Factor (γ)	0.99
Target Network Update Frequency	30
No Of Episodes	700



(a) Total Rewards vs Episodes

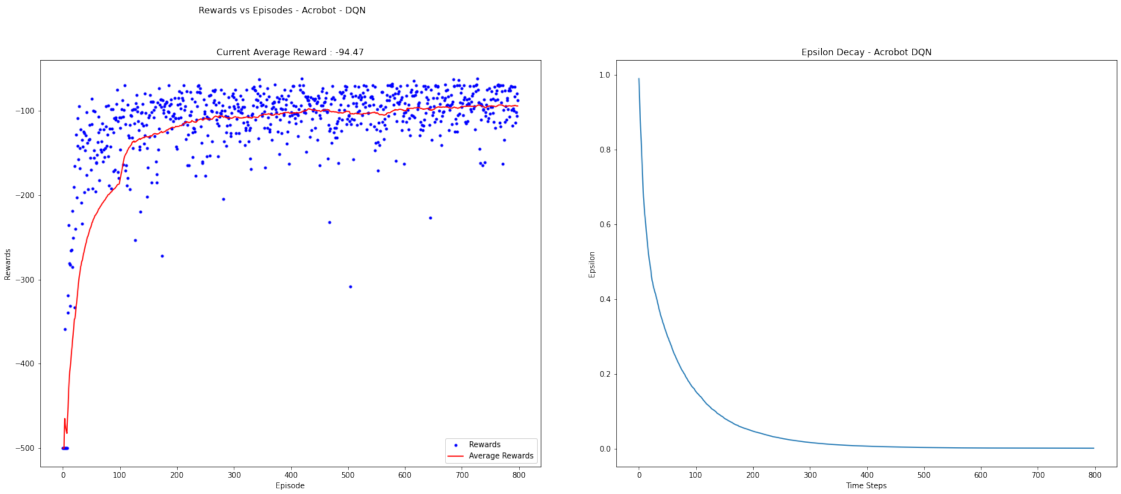
(b) Epsilon Decay

Figure 2: DQN Implementation in CartPole-v1

In the Cartpole environment, the initial output of the model for the first 200 episodes may have been random, possibly due to a low Epsilon Decay Rate. However, after 200 episodes, there was a significant increase in rewards. Although there was a slight dip after 400 episodes, which could indicate overfitting, the model quickly recovered and ultimately converged after 600 episodes, achieving an average reward of 499.14 over the last 100 episodes. These results demonstrate that the model has learned the optimal policy for the given hyperparameters after extensive training.

3.3 Hyper Parameters used for Acrobot-v1 Environment

Epsilon	0.99
Minimum Epsilon	0.001
Epsilon Decay Rate	10000
Memory Size	10000
Learning Rate (α)	1e-3
Batch Size	256
Discount Factor (γ)	0.99
Target Network Update Frequency	30
No Of Episodes	800



(a) Total Rewards vs Episodes

(b) Epsilon Decay

Figure 3: DQN Implementation in Acrobat-v1

In the Acrobat environment, the model exhibited rapid convergence early on, quickly reaching a value near the maximum threshold (-100) within the first 200 episodes. However, the model's progress stagnated thereafter, with rewards increasing at a slow and steady pace. By the end of the 800-episode training run, the model's average reward over the last 100 episodes was -94.47. Although this is a decent performance, it is not as impressive as the algorithm's past performances in other environments. This might be due to the difficulty of precisely controlling the acrobat's unpredictable movements in this environment.

4 Evaluation & Interpretation of the Results

4.1 Grid Environment

The grid environment's model has converged, resulting in a maximum reward of 10 across all 15 time steps when running with a greedy epsilon value. This indicates that the model has learned the optimal policy and is consistently making the best decisions to maximize the reward.

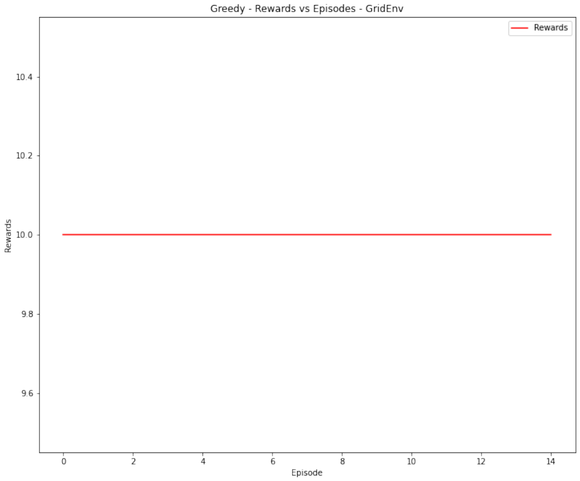


Figure 4: Total Rewards vs Episodes Grid Environment

4.2 CartPole-v1 Environment

The model in the Cartpole environment converged after 700 episodes, and when tested with greedy actions for 10 timesteps, it achieved the highest possible reward of 500. These results demonstrate that the model has learned the optimal policy and consistently makes the best decisions to maximize reward.

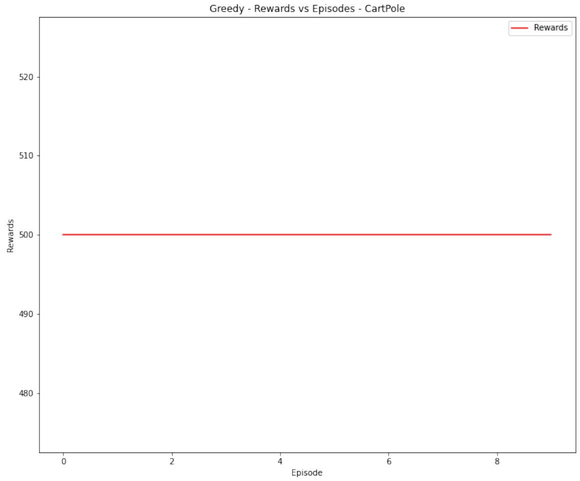


Figure 5: Total Rewards vs Episodes CartPole-v1

4.3 Acrobat-v1 Environment

After 800 episodes in the Acrobat environment, the model converged and consistently achieved a reward value less than the threshold of -100 when tested with greedy actions for 5 timesteps. These results demonstrate that the model has learned the optimal policy and meets the reward criteria for the Acrobat environment.

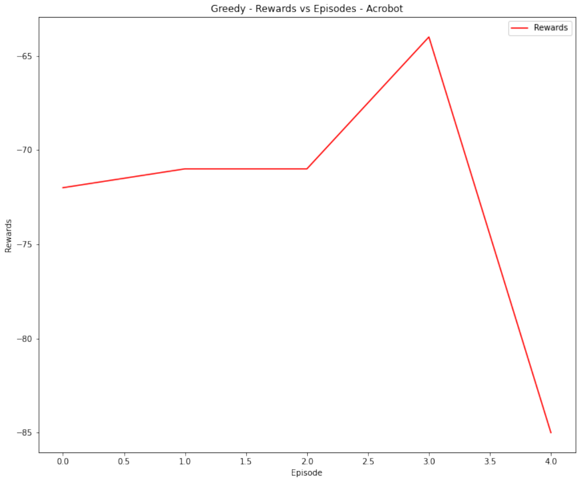


Figure 6: Total Rewards vs Episodes Acrobat-v1

Part 3

5 Report

5.1 Double DQN Algorithm

The Double DQN algorithm is an improvement over the vanilla DQN algorithm that uses a target neural network as another Q-network. This helps in reducing the overestimation of the Q-values, which is a common problem in DQN. The main difference between DQN and Double DQN is that in Double DQN, the Q-value estimation is split into two parts: one part that selects the action and another part that evaluates the Q-value of that action. This way, the action selection and evaluation are decoupled, preventing the maximization bias that can occur in DQN. The update functions are also different in DQN and Double DQN.

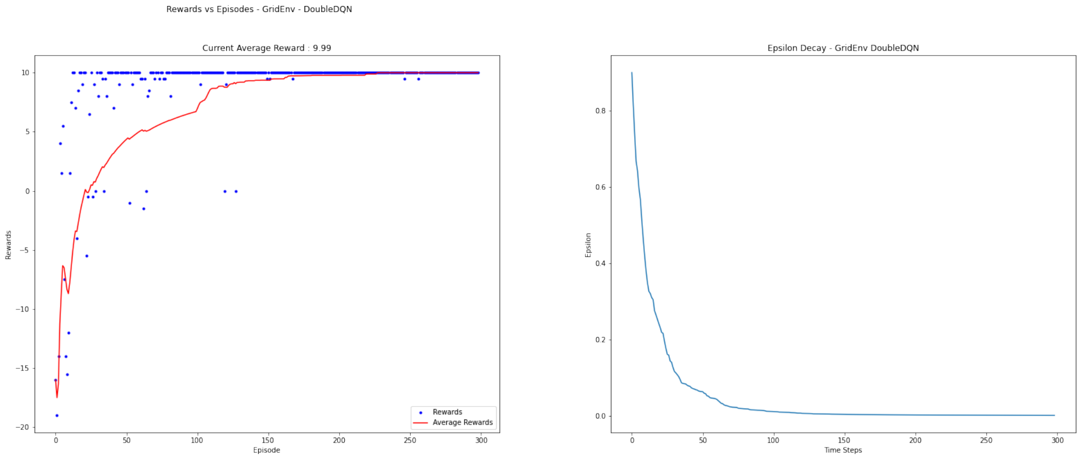
5.2 Main improvement of Double DQN over the vanilla DQN

Double DQN helps in reducing the over-estimation of Q-values, thus improving stability and performance of the algorithm. Double DQN has more efficient update algorithm than vanilla DQN, allowing it to converge to an optimal policy faster. However, it should be noted that the performance of Double DQN is highly dependent on the choice of hyperparameters and the complexity of the environment.

5.3 Results after applying Double DQN on the environments

5.3.1 Grid Environment

Epsilon	0.9
Minimum Epsilon	0.001
Epsilon Decay Rate	1000
Memory Size	1000
Learning Rate (α)	0.001
Batch Size	64
Discount Factor (γ)	0.99
Target Network Update Frequency	20
No Of Episodes	300
tau (τ)	0.05



(a) Total Rewards vs Episodes

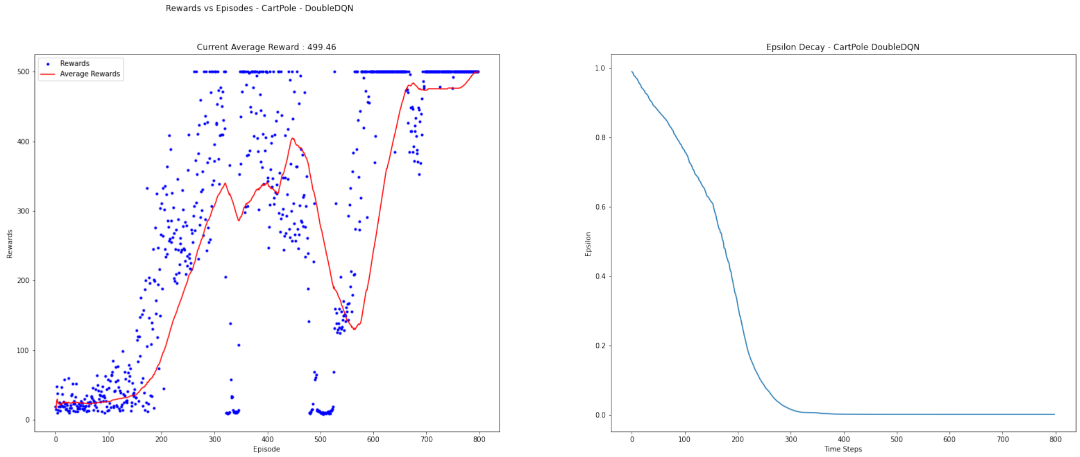
(b) Epsilon Decay

Figure 7: Double DQN Implementation in Grid Environment

Double DQN was run for 300 episodes, with a steep increase in rewards from the beginning. The model achieved its maximum average reward of 9.99 after 200 episodes, indicating that it learned the environment and found the optimal solution.

5.3.2 Cartpole

Epsilon	0.99
Minimum Epsilon	0.001
Epsilon Decay Rate	10000
Memory Size	5000
Learning Rate (α)	1e-4
Batch Size	128
Discount Factor (γ)	0.99
Target Network Update Frequency	30
No Of Episodes	800
tau (τ)	0.03



(a) Total Rewards vs Episodes

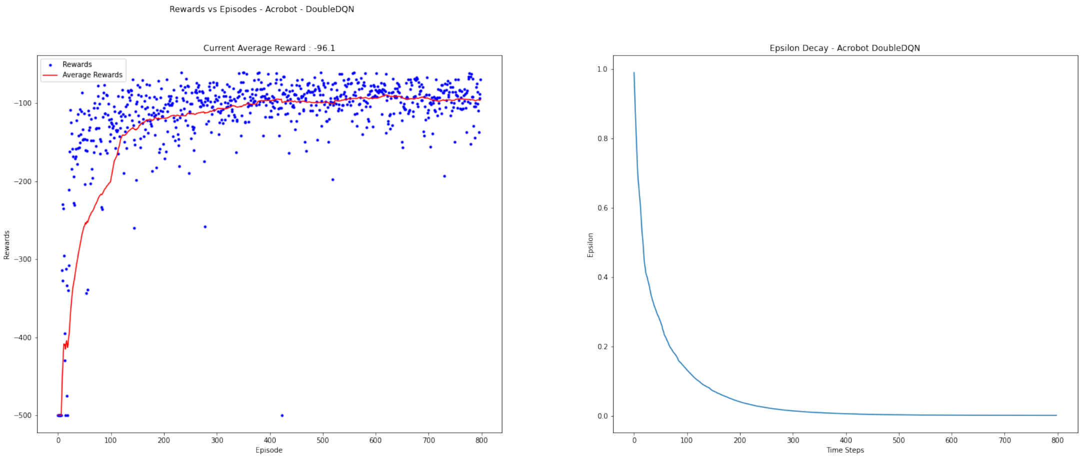
(b) Epsilon Decay

Figure 8: Double DQN Implementation in CartPole-v1

Initially, the model trained on the Cartpole environment did not improve significantly in the first 200 episodes, potentially due to a high epsilon value resulting in more exploration. However, after 200 episodes, the rewards steadily increased, indicating that the model began to learn the environment. Around 500 episodes, there was a sudden decrease in rewards, possibly due to overfitting of the model to some environment parameters. However, the model quickly recovered and achieved a maximum average reward of 499.46 over the last 100 episodes (around episode 800). This peak in average reward demonstrates that the model has converged and found an optimal solution.

5.3.3 Acrobat

Epsilon	0.99
Minimum Epsilon	0.001
Epsilon Decay Rate	10000
Memory Size	5000
Learning Rate (α)	1e-3
Batch Size	64
Discount Factor (γ)	0.999
Target Network Update Frequency	20
No Of Episodes	800
tau (τ)	0.03



(a) Total Rewards vs Episodes

(b) Epsilon Decay

Figure 9: Double DQN Implementation in Acrobat-v1

When training the model on the Acrobat environment, we observed that it converged relatively quickly, with a noticeable increase in rewards from -500 to over -150 within 200 episodes. Interestingly, we also noted that the epsilon decay graph reached a minimum value at this point, suggesting that the model had explored the environment extensively and was ready to start exploiting what it had learned. However, due to the low learning rate, the model's progress was slow from this point onwards. Despite this, by the end of the 800-episode training run, the model's average reward over the last 100 episodes was -96.1, indicating that it had made significant progress and was performing much better than at the start of training.

5.4 Evaluation Results

5.4.1 Grid Environment

The grid environment's model has converged, and gives a total reward of 10 for every episode on running with greedy approach. This means that the model has learned the optimal policy and is able to come up with the best ways to reach the goal.

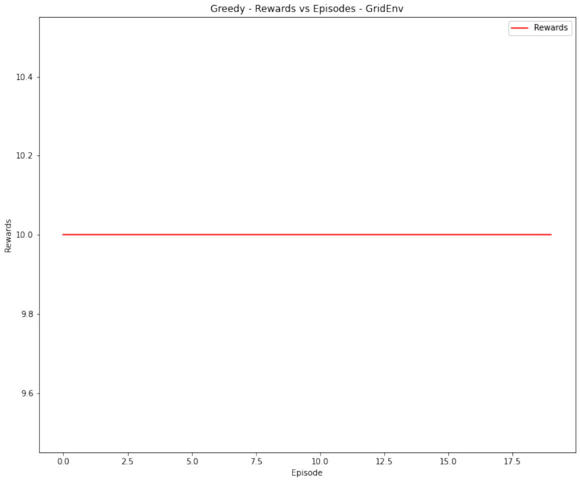


Figure 10: Total Rewards vs Episodes Grid Environment

5.5 CartPole-v1 Environment

On testing for 10 episodes with Cartpole Double DQN model, we notice that it reaches a highest reward of 500, which means it's able to maintain the pole for 500 timesteps after which the environment itself terminates. This indicates the model has learnt how to balance a pole in the cart.

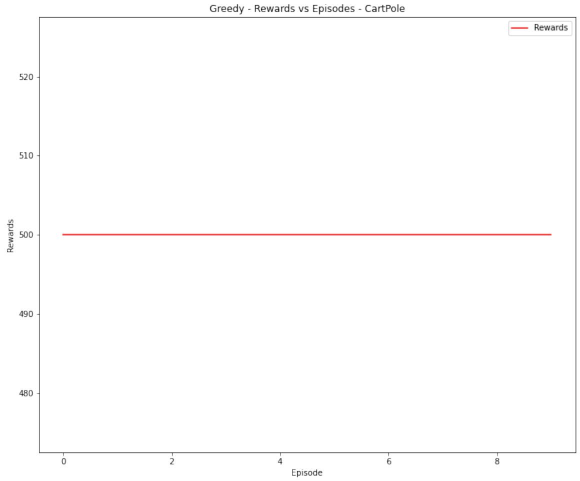


Figure 11: Total Rewards vs Episodes CartPole-v1

5.6 Acrobat-v1 Environment

We have trained the Double DQN for 1000 episodes in the acrobat environment. On testing for 5 timesteps with greedy method, the model consistently achieves a reward values less than the threshold of -100. This indicates model learnt the optimal policy and is able to rotate the acrobat above a certain height.

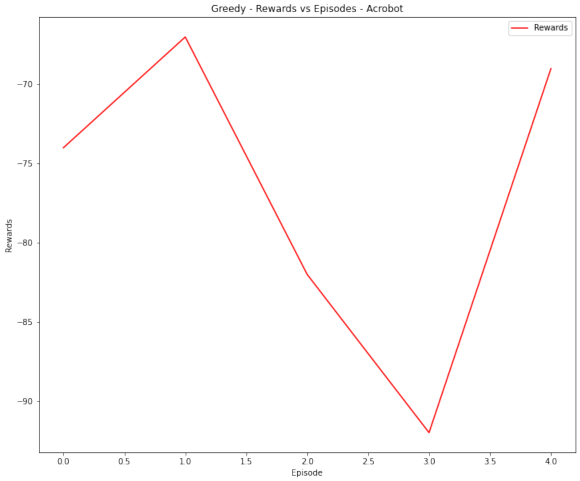


Figure 12: Total Rewards vs Episodes Acrobat-v1

6 Intrepretation of the Results across environments

6.1 Grid Environment

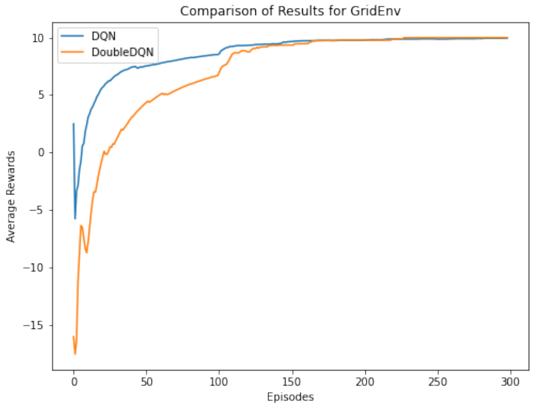


Figure 13: Comparison of Grid Environment

Upon analyzing the performance of the Double DQN algorithm, we observed that it started from a relatively low reward value due to its poor initial performance. However, it managed to converge and eventually reach the same maximum reward value of 10 as the DQN algorithm. Interestingly, after 250 episodes, the average reward graphs of both algorithms were almost identical and overlapped each other. Furthermore, both DQN and Double DQN algorithms exhibited a steep ascent in the beginning, indicating that the model was able to reach the maximum rewards at a high pace. This observation may be attributed to the simplicity of the grid environment in which the algorithms were trained.

6.2 CartPole-V1

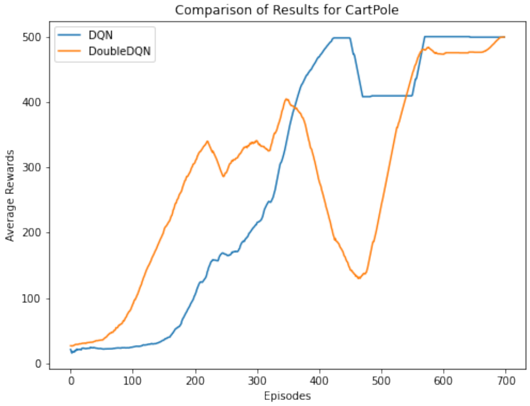


Figure 14: Comparison of CartPole Environment

On comparing DQN and Double DQN algorithm for Cartpole environment, we observed that DQN converged faster than Double DQN. DQN converged for a brief time period from episode 400 to 450, whereas there was a steep decrease in rewards from episodes 300 to 500 in Double DQN. However,both algorithms converged at around episode 550 with DQN having a maximum reward of 500, whereas Double DQN had a maximum reward of 480. Notably, at 700 episodes, Double DQN also achieved a maximum reward of 500. In general, Double DQN outperforms DQN, but for this current set of hyperparameters, DQN performs better.

6.3 Acrobat-v1

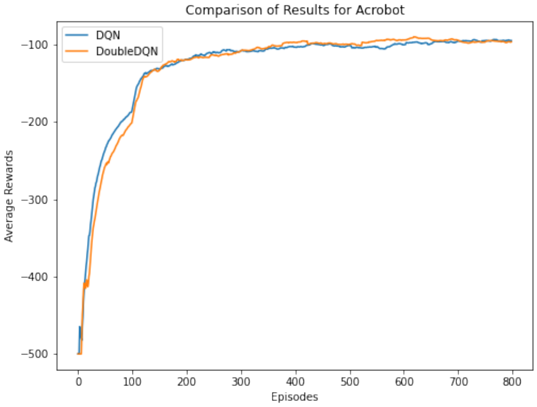


Figure 15: Comparison of Acrobot Environment

Upon comparing the performance of DQN and Double DQN algorithms on the Acrobot environment, we noted that both algorithms had very similar performance. Both algorithms demonstrated a steep ascent and quick convergence starting from around 200 episodes, and maintained a steady state throughout the remaining training period of 800 episodes. This observation may be attributed to the low value of the hyperparameter tau (τ), which resulted in both algorithms performing similarly.

7 Github Link

UB_CSE546_RL_Assignment_2

8 Contribution

Team member	**Assignment Part**	**Contribution (%)**
Mohit Sai Aravind Nunna	Part 1	50(%)
Vignesh Prakash	Part 1	50 (%)
Mohit Sai Aravind Nunna	Part 2	50(%)
Vignesh Prakash	Part 2	50(%)
Mohit Sai Aravind Nunna	Part 3	50(%)
Vignesh Prakash	Part 3	50(%)

Table 1: Team members' contributions to the project

9 References

Gym Documentation

Human-level control through deep reinforcement learning