

# Assignment 2 - Implementing DQN & Solving RL Environments

Indra teja Pidathala, Team 25

November 6, 2022

## 1 Introduction

In our Last Assignment(1), we have helped our friend ,Agent, Agent-ZERO-ZERO-VISION to reach find the infinity gauntlet location in the grid vis-a-v reaching the goal optimally. But from our understanding we came to know that q-learning and other algorithmic approximations are a bit slow and can be improved upon so, this time we are going a little complex and will be implementing Double Deep-Q-Neural Network on our grid.

In addition to applying Double-DQN to Grid, we'll also use it to solve [Cartpole-v1](#) and [LunarLander-v2](#) environments from the [open AI Gym](#).

## 2 Definition

### 2.1 Grid Environment

Our Grid Environment(MDP) is defined below.

$States \rightarrow S : \{S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15\}.$

$Actions \rightarrow A : \{LEFT : 0, RIGHT : 1, NORTH : 2, SOUTH : 3\}.$

$Rewards \rightarrow R : \{-1, -1, 25, -1, -1, -1, -1, -1, -1, -1, -1, -20, -1, -1, 100\}.$

$Policy - Deterministic \rightarrow \pi_d(s) : a.$

All the states allow all the actions. the rewards for each state has been shown in the below.Figures shown has captions to describe what it represents(Fig 1-7). in case of a reward, the value of reward has been shown just beside the reward name.



Figure 1: Agent 00Vision



Figure 2: Reward - Wanda +25



Figure 3: Reward - Thanos -20

## 2.2 CartPole

The description of the CartPole-v1 as given on the OpenAI gym website is given below

*A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The*



Figure 4: Reward - Gauntlet +100





<b>S0 (0,0)</b> 	<b>S1 (0,1)</b>	<b>S2 (0,2)</b>	<b>S3 (0,3)</b> 
<b>S4 (1,0)</b>	<b>S5 (1,1)</b>	<b>S6 (1,2)</b>	<b>S7 (1,3)</b>
<b>S8 (2,0)</b>	<b>S9 (2,1)</b>	<b>S10 (2,2)</b>	<b>S11 (2,3)</b>
<b>S12 (3,0)</b> 	<b>S13 (3,1)</b>	<b>S14 (3,0)</b>	<b>S15 (3,3)</b> 

Figure 5: Environment - Grid - Initial - State

*episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.*

Our Cartpole Environment(MDP) is defined below(Fig 8).

*States*  $\rightarrow S : [(0 : 1), (0 : 1), (0 : 1), (0 : 1)]4 - \text{continuous} - \text{values}.$

*Actions*  $\rightarrow A : \{LEFT : 0, RIGHT : 1\}.$

*Rewards*  $\rightarrow R : \{-100, 1\}.$

*Policy - Deterministic*  $\rightarrow \pi_d(s) : a.$

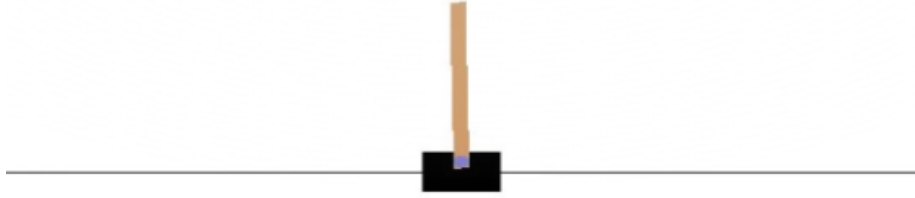


Figure 6: CartPole

### 2.3 LunarLander

Our LunarLander Environment(MDP) is defined below(Fig 8).

*States*  $\rightarrow S : [(0 : 1), (0 : 1), (0 : 1), (0 : 1), (0 : 1), (0 : 1), (0 : 1), (0 : 1)].$  8-continuous-values

*Actions*  $\rightarrow A : \{0, 1, 2, 3\}.$  [do-nothing, fire-left-engine, fire-right-engine, fire-main-engine]

*Rewards*  $\rightarrow R : \{-100, -1, 100\}.$

*Policy - Deterministic*  $\rightarrow \pi_d(s) : a.$

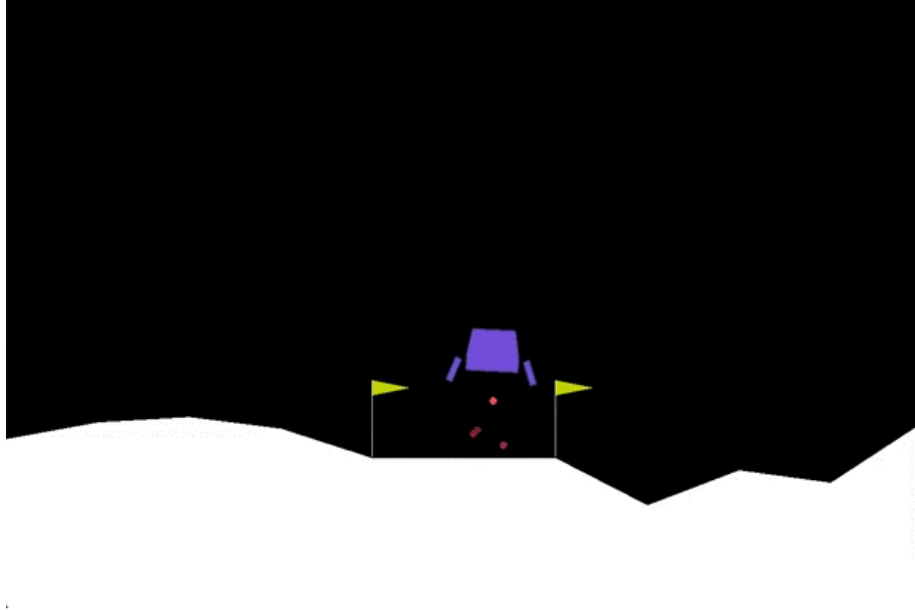


Figure 7: Lunar Lander

### 3 Double DQN

#### Double Q-learning

---

##### Algorithm 1 Double Q-learning

---

```

1: Initialize  $Q^A, Q^B, s$ 
2: repeat
3:   Choose  $a$ , based on  $Q^A(s, \cdot)$  and  $Q^B(s, \cdot)$ , observe  $r, s'$ 
4:   Choose (e.g. random) either UPDATE(A) or UPDATE(B)
5:   if UPDATE(A) then
6:     Define  $a^* = \arg \max_a Q^A(s', a)$ 
7:      $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a) (r + \gamma Q^B(s', a^*) - Q^A(s, a))$ 
8:   else if UPDATE(B) then
9:     Define  $b^* = \arg \max_a Q^B(s', a)$ 
10:     $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a) (r + \gamma Q^A(s', b^*) - Q^B(s, a))$ 
11:   end if
12:    $s \leftarrow s'$ 
13: until end

```

---

Figure 8: Double-DQN Implementation Algorithm

A Double Deep Q-Network is nothing but the cross between Double Q Learn-

ing and DQN. we combine both of them to make a more robust algorithm. In this we have 2 deep neural networks (instead of one in case of DQN) which act as estimators where one network obtains best actions and the other network evaluates these best actions.

the difference between DQN and DDQN is in the calculation of the target Q-values of the next states. In DQN, we simply take the maximum of all the Q-values over all possible actions. This is likely to select over-estimated values, hence DDPG proposed to estimate the value of the chosen action instead. The chosen action is the one selected by our policy model. other than having 2 networks and the update policy, Double-DQN is same as DQN.

### 3.1 Improvements

The main Advantage of Double DQN over vanilla DQN is that DDQN handles the problem of the overestimation of Q-values(Maximisation Bias).it helps us reduce the overestimation of q values (by having 2 estimators) and, as a consequence, helps us train faster and have more stable learning

## 4 DQN Results Evaluation & Visualizations

Finally, we have implemented the DQN on the 3 RL environemnts and observed the following results(plots).

### 4.1 Grid Environment

Following are our parameters/hyper-parameters for all of our cases

- Total No of Episodes = 1000
- Discount Factor = 0.99
- Epsilon(initial) = 1
- Epsilon Decay Rate = 0.001
- Max Time steps = 50
- Replay Memory Size = 3000
- Mini Batch Size = 128
- Model Fit Freq = 10
- Weights Sync Freq = 5

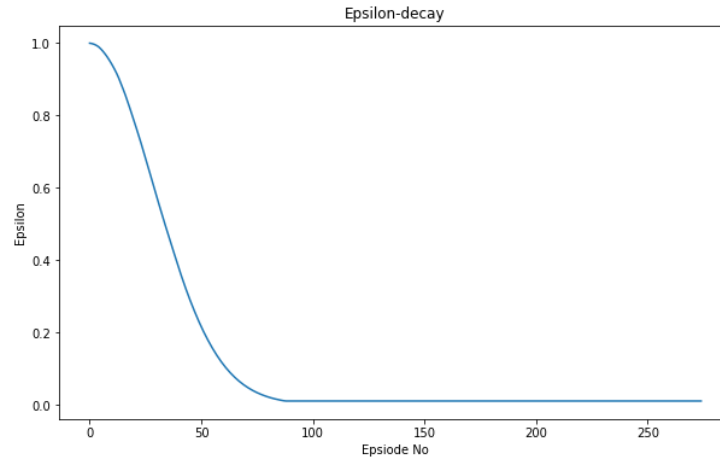


Figure 9: DQN Grid - Epsilon Decay Plot

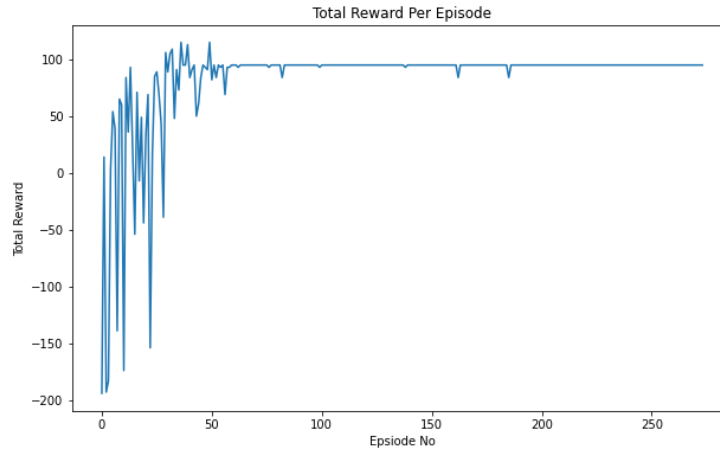


Figure 10: DQN Grid - Episode Rewards Plot

Following are our Results Figures 10-12 (Descriptions of plots explain themselves)

**Training Analysis** from the plots, we observed that epsilon (Fig-9) has been decayed after just 100 iterations. the total reward per episode (Fig-11) has been converged after around 60 episodes and stabilised thereafter. the converged reward is around 96.

**Evaluation** Fig-12, the agent is ow only taking greedy actions and is getting the expected reward.

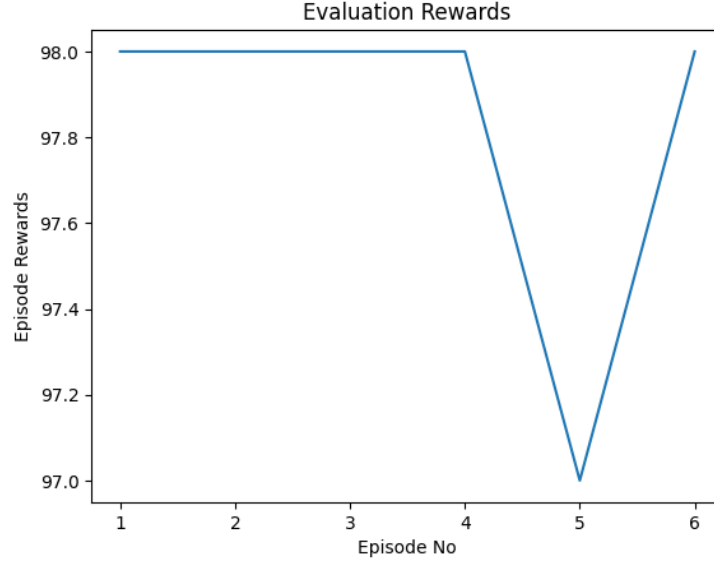


Figure 11: DQN Grid - Evaluation Rewards for 6 episodes

## 4.2 Cartpole-v1

Following are our parameters/hyper-parameters for all of our cases

- Total No of Episodes = 500
- Discount Factor = 0.99
- Epsilon(initial) = 1
- Epsilon Decay Rate = 0.001
- Replay Memory Size = 3000
- Mini Batch Size = 64
- Model Fit Freq = 10
- Weights Sync Freq = 5

Following are our Results Figures 13-16 (Descriptions of plots explain themselves)

**Training Analysis** After training for 500 episodes, from the plots, we observed that epsilon (Fig-13) has been decayed after just 100 iterations. the total reward per episode (Fig-14) goes as high as 140 but averages 100 through out the training. overall, the rewards are not converged yet. but no of steps made in an episode (Fig-15) are going as high as 500 and averages around 250 which is expected



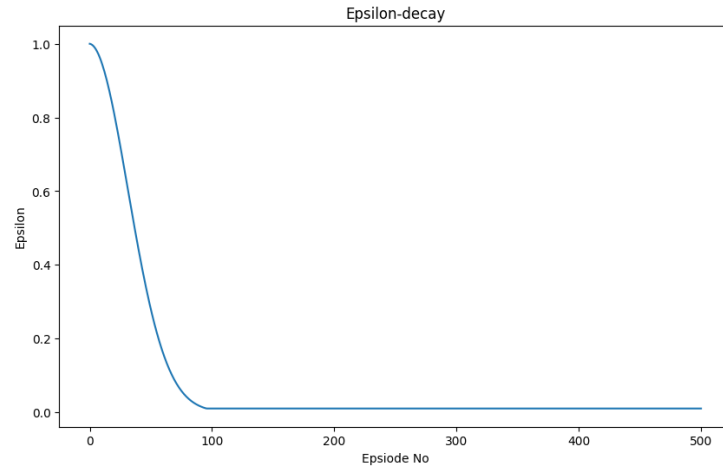


Figure 12: Cartpole - Epsilon Decay Plot

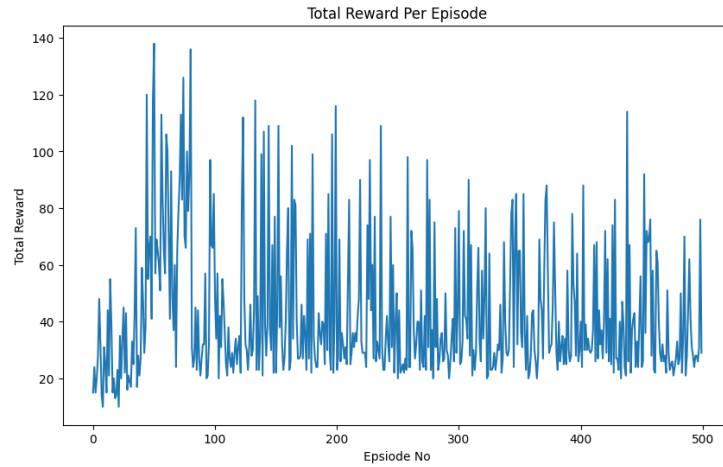


Figure 13: Cartpole - Episode Rewards Plot

**Evaluation**(Fig-16) we can observe that the reward has been fluctuating but not went to the expected maximum(470)

### 4.3 LunarLander-v2

Following are our parameters/hyper-parameters for all of our cases

- Total No of Episodes = 300

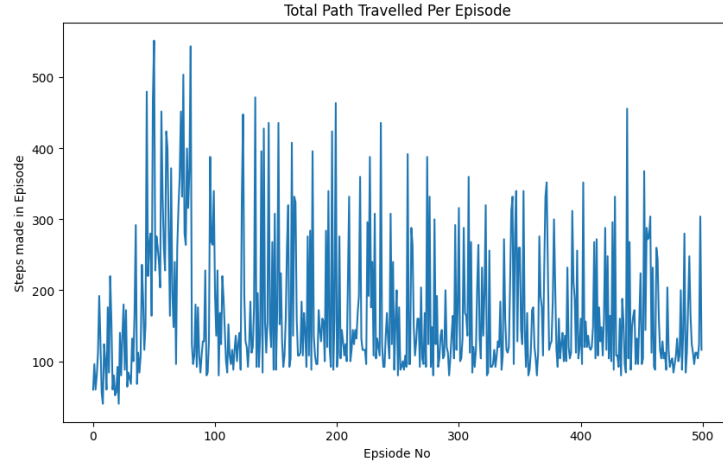


Figure 14: Cartpole - Episode Steps made

- Discount Factor = 0.95
- Epsilon(initial) = 1
- Epsilon Decay Rate = 0.001
- Replay Memory Size = 2000
- Mini Batch Size = 500000
- Model Fit Freq = 10
- Weights Sync Freq = 5

Following are our Results Figures 17-19 (Descriptions of plots explain themselves)

**Training Analysis** After training for 300 episodes, from the plots, we observed that epsilon (Fig-17) has been decayed after just 100 iterations. the total reward per episode (Fig-18) goes as high as -180 but averages -3500 through out the training. overall, the rewards are not converged yet. it's nowhere near expected results.

**Evaluation** Fig-12, we can observe that the reward has been converged (-100) but not went to the expected maximum

the model needs to be made optimal and rewards need to converge expectedly, we'll do this by the deadline and will produce the expected results for the deadline.

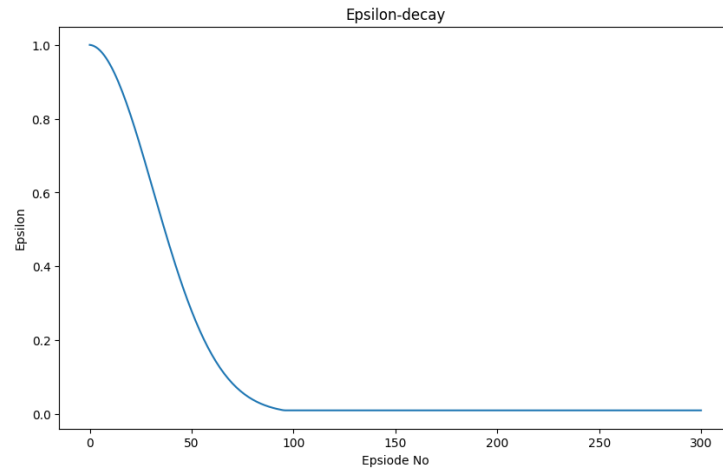


Figure 15: LunarLander - Epsilon Decay Plot

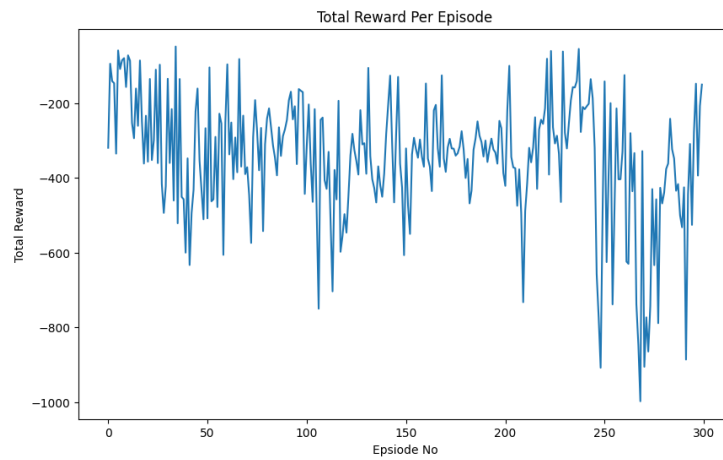


Figure 16: LunarLander - Episode Rewards Plot

1	test(model_300)
episode: 1/10, reward: -614.5165379187874, time_steps: 1	
episode: 2/10, reward: -100, time_steps: 1	
episode: 3/10, reward: -100, time_steps: 1	
episode: 4/10, reward: -100, time_steps: 1	
episode: 5/10, reward: -100, time_steps: 1	
episode: 6/10, reward: -100, time_steps: 1	
episode: 7/10, reward: -100, time_steps: 1	
episode: 8/10, reward: -100, time_steps: 1	
episode: 9/10, reward: -100, time_steps: 1	
episode: 10/10, reward: -100, time_steps: 1	

Figure 17: LunarLander - Evaluation Rewards for 10 episodes

## 5 Double DQN Results Evaluation & Visualizations

Finally, we have implemented the Double-DQN on the 3 RL environments and observed the following results(plots).

### 5.1 Grid Environment

Following are our parameters/hyper-parameters for all of our cases

- Total No of Episodes = 300
- Discount Factor = 0.99
- Epsilon(initial) = 1
- Epsilon Decay Rate = 0.001
- Max Time steps = 50
- Replay Memory Size = 3000
- Mini Batch Size = 128
- Model Fit Freq = 10
- Weights Sync Freq = 5

Following are our Results Figures 10-12 (Descriptions of plots explain themselves)

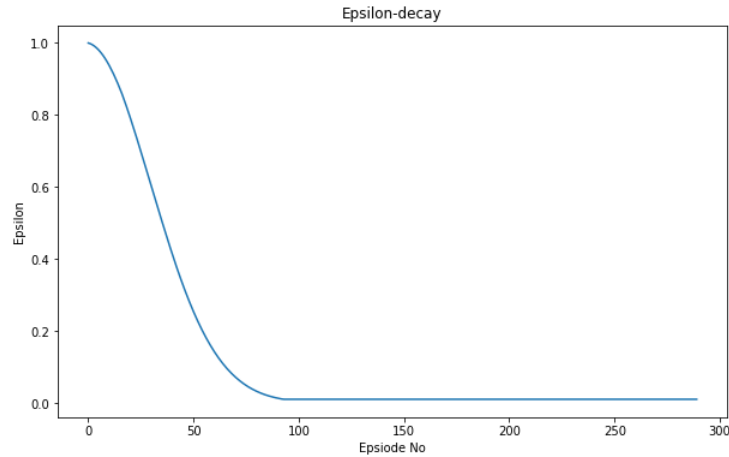


Figure 18: DDQN Grid - Epsilon Decay Plot

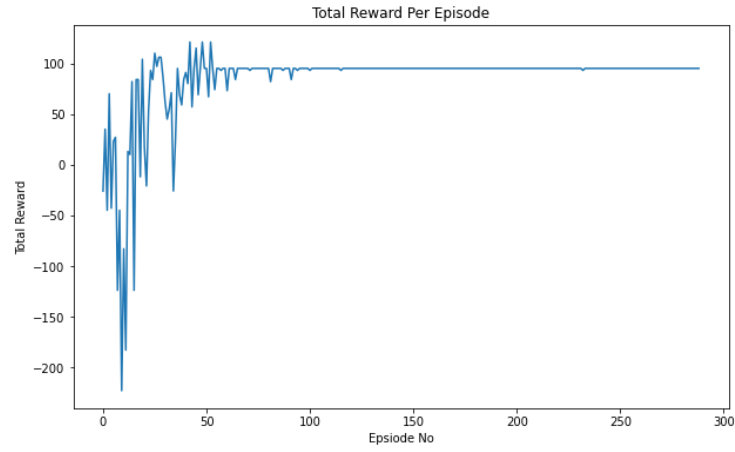


Figure 19: DDQN Grid - Episode Rewards Plot

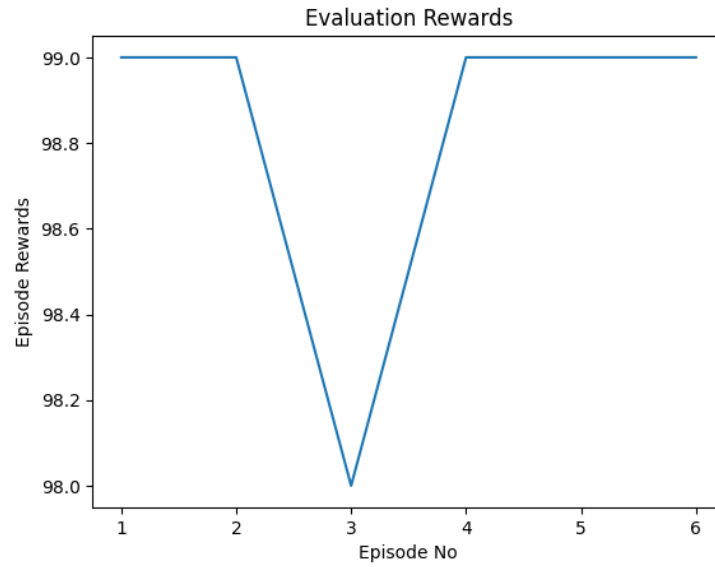


Figure 20: DDQN Grid - Evaluation Rewards for 6 episodes

**Training Analysis**from the plots, we observed that epsilon (Fig-9) has been decayed after just 100 iterations. the total reward per episode (Fig-11) has been converged after around 60 episodes and stabilised thereafter. the converged reward is around 98.

**Evaluation**Fig-12, the agent is ow only taking greedy actions and is getting the expected reward.

## 5.2 Cartpole-v1

Following are our parameters/hyper-parameters for all of our cases

- Total No of Episodes = 500
- Discount Factor = 0.99
- Epsilon(initial) = 1
- Epsilon Decay Rate = 0.001
- Replay Memory Size = 3000
- Mini Batch Size = 64
- Model Fit Freq = 10
- Weights Sync Freq = 5

Following are our Results Figures 13-16 (Descriptions of plots explain themselves)

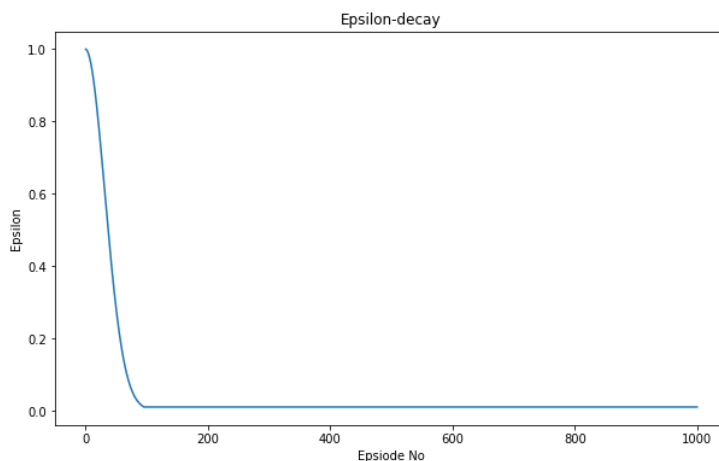


Figure 21: DDQN Cartpole - Epsilon Decay Plot

**Training Analysis** After training for 1000 episodes, from the plots, we observed that epsilon (Fig-13) has been decayed after just 100 iterations. the total reward per episode (Fig-14) goes as high as 60 but averages 10 through out the training. overall, the rewards are not converged yet and overall very bad.

**Evaluation** (Fig-16) we can observe that the max reward we got is only 16. we do not know why is this happening, we have tried tuning all the hyperparameters like no of layers(2,3,4,5), mini batch size(16,64,128), learning rate(0.001,0.005,0.05) etc.

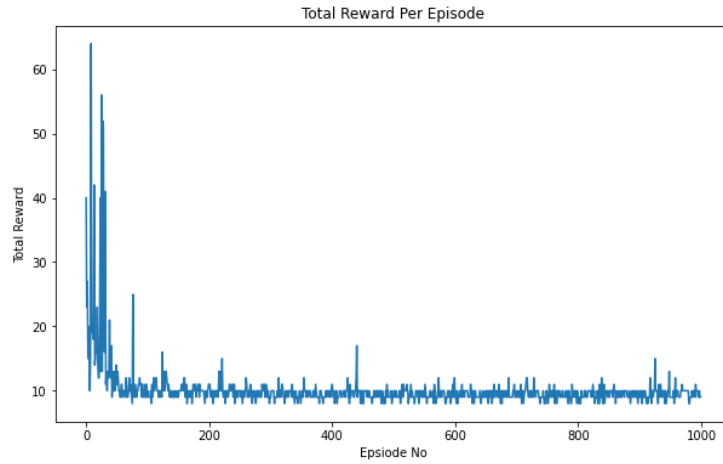


Figure 22: DDQN Cartpole - Episode Rewards Plot

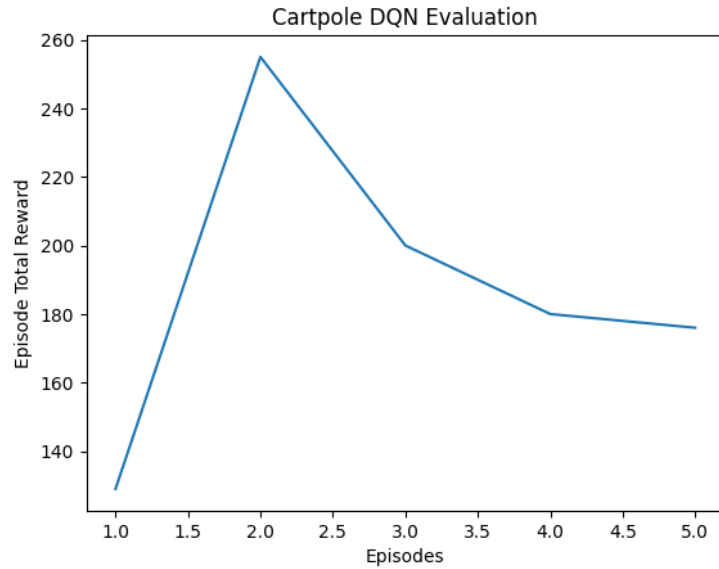


Figure 23: DDQN Cartpole - Evaluation Rewards for 10 episodes

### 5.3 LunarLander-v2

Following are our parameters/hyper-parameters for all of our cases

- Total No of Episodes = 300
- Discount Factor = 0.95



- Epsilon(initial) = 1
- Epsilon Decay Rate = 0.001
- Replay Memory Size = 2000
- Mini Batch Size = 128
- Model Fit Freq = 10
- Weights Sync Freq = 5

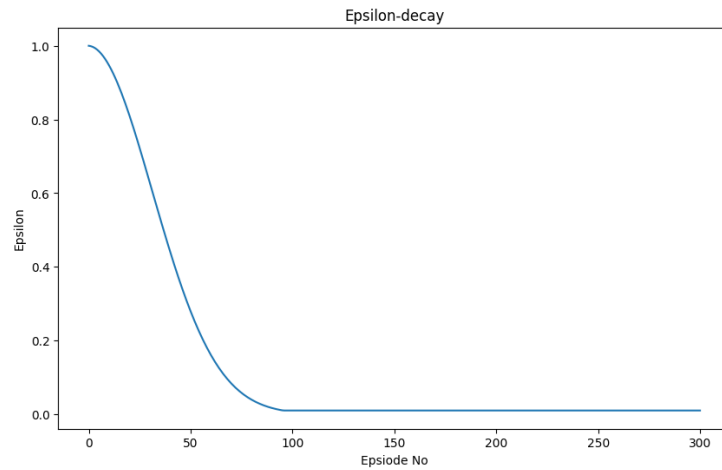


Figure 24: DDQN LunarLander - Epsilon Decay Plot

**Training Analysis** After training for 300 episodes, from the plots, we observed that epsilon (Fig-17) has been decayed after just 100 iterations. the total reward per episode (Fig-18) goes as high as -180 but averages -3500 throughout the training. overall, the rewards are not converged yet. it's nowhere near expected results.

**Evaluation** Fig-12, we can observe that the reward has been converged (-100) but not went to the expected maximum

the model needs to be made optimal and rewards need to converge expectedly, we'll do this by the deadline and will produce the expected results for the deadline.

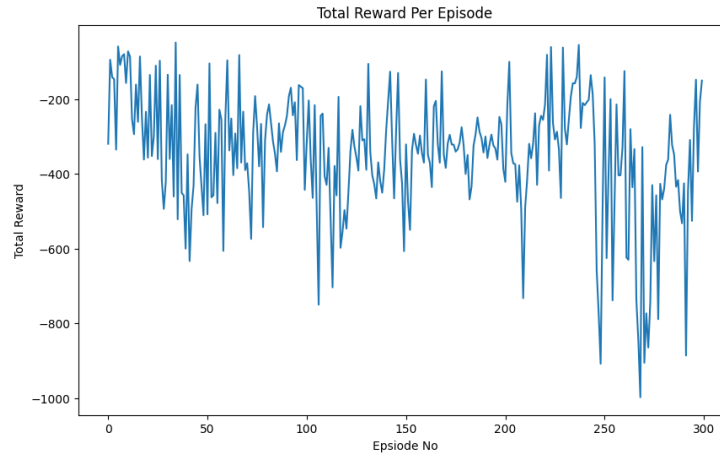


Figure 25: DQN LunarLander - Episode Rewards Plot

## 6 DQN vs Double-DQN)

### 6.1 Grid Env

from the reward graphs(Fig 10- DQN Grid and Fig 19 DDQN Grid) for our grid environment, we didn't find much improvements in Double DQN over DQN. the convergence started happening almost at the same no of the episodes(60) and giving the same average episode reward after converging.

the only significant difference observed is in terms of training time where Double DQN took only 25-30 minutes for 300 episodes while DQN took 35 minutes for the same no of episodes.

### 6.2 Cartpole-v1

both of our algorithms are not performing expectedly , though DQN have better random rewards than DDQN, DDQN has converged. both are not learning as expected.

but in general DDQN should perform better than DQN due to it's low maximisation bias and expected to converge faster than DQN

### 6.3 Lunar Lander

both of our algorithms are not performing expectedly , though DQN is random and DDQN has converged, both are not learning as expected.

but in general DDQN should perform better than DQN due to it's low maximisation bias and expected to converge faster than DQN

## 7 References

- NIPS Styles (docx, tex)
- Overleaf (LaTeX based online document generator) - a free tool for creating professional reports GYM environments

- GYM environments
- Open AI Gym (Documentation, etc)
- Lecture slides
- [DQN-Paper](#)
- Wikipedia
- Richard S. Sutton and Andrew G. Barto, "Reinforcement learning: An introduction" (pdf)
- google.com and internet in general(only for referencing not for copying)
- marvel comics