CSE 4/546: Reinforcement Learning

Ajit Mutalik(50416741)
G. Laxmi Devi (50400730)

PART 1 – Exploring OpenAI Gym Environments

1. CartPole-v1

The environment depicts "A cartpole agent where the cart is trying to balance the pole vertically, with a little shift of the angle. **The goal is to prevent it from falling over** (to keep the pole upright for as long as possible)."

1.1 Actions

A ϵ {0,1}
0 - left
1 - right

1.2 States

There are four values representing the state: cart position, cart-velocity, pole angle and pole velocity respectively.

	Min	Max
• Cart Position	-4.8	4.8
• Cart Velocity	-Inf	Inf
• Pole Angle	-0.418 rad (-24°)	0.418 rad (24°)
• Pole Angular Velocity	-Inf	Inf

The episode terminates if the cart position leaves the (-2.4, 2.4) range. The episode terminates if the pole angle is not in the range (-.2095, .2095) (or **±12°**).

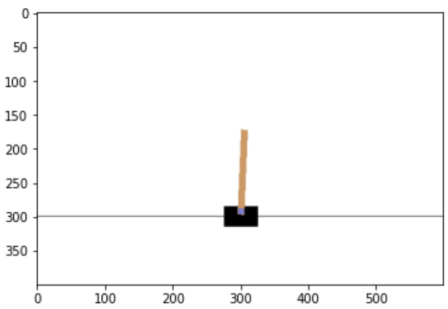
1.3 Rewards

There is **only one** unique reward in the environment.

R ϵ {+1}

The reward for every step taken is +1, including the termination step. The threshold for rewards is 475 for v1.

1.4 Visualization



2. Mountain Car

A car is on a track between two mountains. **The goal is to reach the mountain on the right**. This can be achieved by driving back and forth to build the momentum because the car's engine is not strong enough.

2.1 Actions

A ϵ {0,1,2}

0- Accelerate to the left
1- Do not Accelerate
2- Accelerate to the right

2.2 States

There are two values representing the state: Position of the car along x-axis and velocity of the car.

	Min	Max
• Car Position (0)	-Inf	Inf
• Car Velocity (1)	-Inf	Inf

The position is clipped to the range [-1.2, 0.6] and velocity is clipped to the range [-0.07, 0.07].

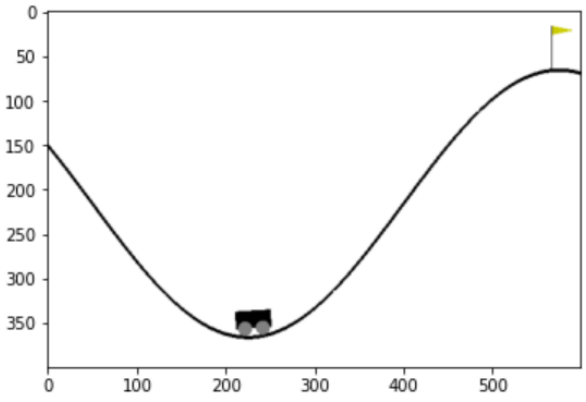
2.3 Rewards

There are **two** unique rewards in the environment.

R ϵ {-1,0}

The agent is penalized with a reward of -1 for each timestep it isn't at the goal and when it reaches the goal the reward is 0.

2.4 Visualization



PART 2- Implementing DQN & Solving Grid-world environment

1. Benefits

- **Using experience replay in DQN and how its size can influence the results.**
  - When using neural network to approximate the Q value of a state action pair, one main requirement for optimization of weight is to have a independent and identically distributed random data. While training an agent if the sequential observations which are highly correlated if used, the policy generated by the neural network could oscillate a lot, because of the distribution of the data.
  - This would also lead to unstable gradients being backpropagated in the network. To overcome this issue, we save the experiences of the agent in a replay buffer and break the correlation between the data by randomly selecting the data from the buffer. Since the data would be random it would also avoid large oscillations.
  - Having a small replay buffer would lead to agent being stuck in a local minimum, and the network not converging. If the replay buffer is too large, the agent is exploring more than exploitation and taking more penalties.

- **Introduction the target network.**
  - The major difference between the Q – Learning and Deep Q Network is that in Q-learning we are just updating a single Q value, but in DQN approach we are updating multiple Q values.
  - Since only one value is getting updated in Q-Learning there is stability while learning, but in DQN approach this leads to the problem of catastrophic forgetting.
  - To tackle this instability, we need to introduce the target network, whose weights are copied over from the policy network after certain time steps. Since this network is lagged over the policy network it helps to come up with a better target for calculating the loss. This gives the network more time to consider recently taken actions, rather than updating continuously.

- **Representing the Q Function as $q\hat{}$= (s,w)**
  - Q Learning is a tabular method, to get the Q value of state-action pair we just need two parameters (state and action). But in Deep Q Network, since the Q values of the state are generated using the neural network, we need to parameterize the Q function with the weights of the network and the current state.

2. Grid Environment

The main objective of the environment is that agent must go around the environment collecting the small reward to fight off the skeleton monsters and to reach the princess and rescue her.

2.1 Actions

A ϵ {up, down, right, left}

2.2 States

The environment is a **4*4** grid that has **16** states where the agent starts at (0,0) and the goal is at (3,3).

2.3 Rewards

There are **four** unique rewards in the environment.

R ϵ {-2, 0, +1, +4}

- Goal (Princess) at [3,3] = +4
- Monsters at [3,0] and [1,1] = -2 each
- Bird at [0,3] and fire at [2,3] = +1 each
- Other Positions = 0

2.3 Visualization

The main character (agent) has the visualization shown below.

Figure 1: Main Character Visualization in the environment.

The enemy / monster has the visualization as shown below.

Figure 2: Enemy / Skeleton Visualization

The Visualization for the bird and fire powerups is as shown below,

Figure 3: Fire and Bird Power ups

The Princess Visualization in the environment is as shown below,

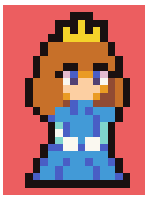


Figure 4: Princess (Goal in the environment)

When the agent collects the bird and fire power ups it is shown as below,

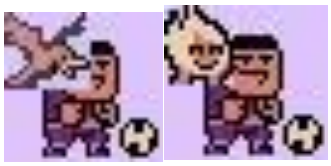


Figure 5: Main Character Fire and Bird Power Ups.

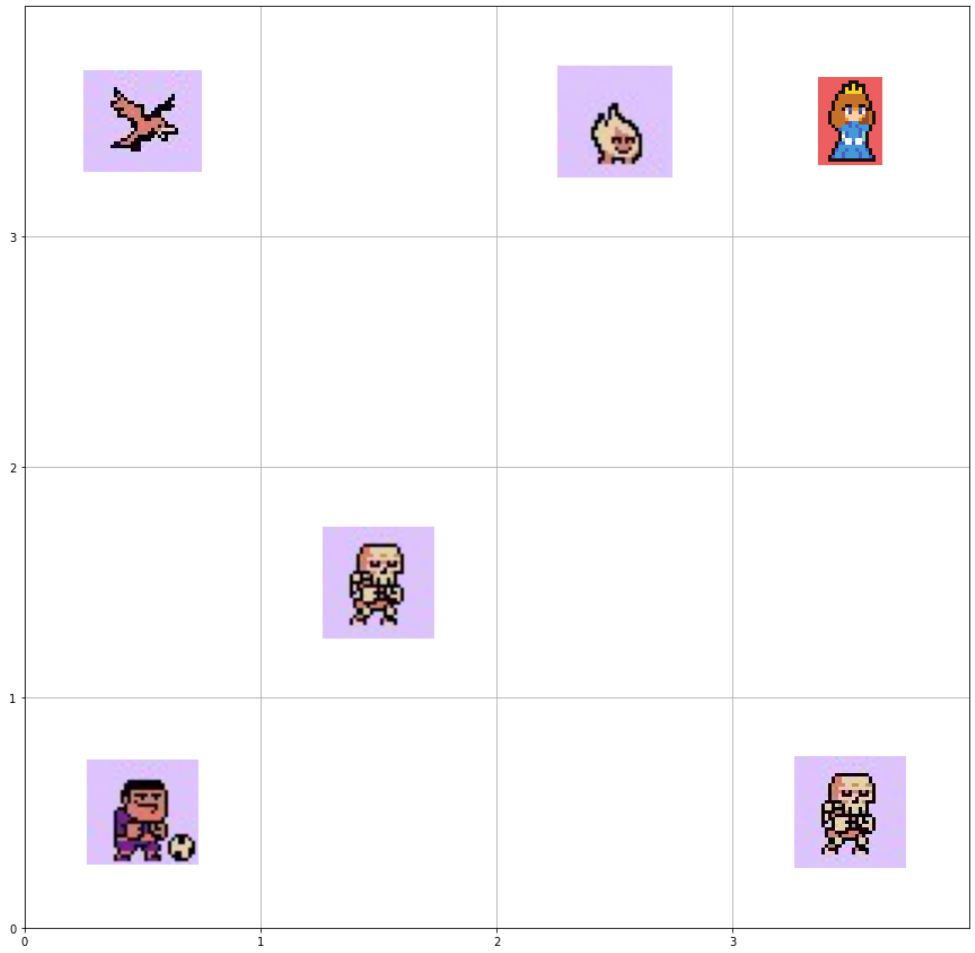
When the agent kills the skeleton, the visualization is shown as,

Figure 6: Main Character Killing the Skeleton.

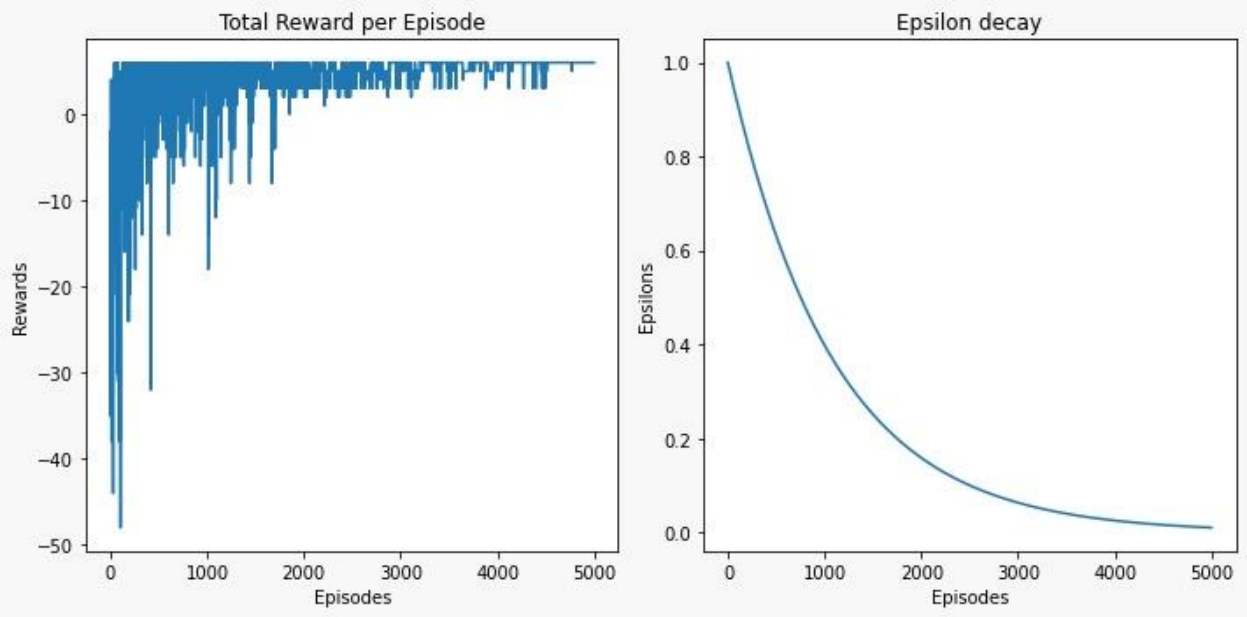
When the agent rescues the princess, the visualization is shown as,

Figure 7: Main Character rescuing the princess.

Generated environment is as shown in the figure,



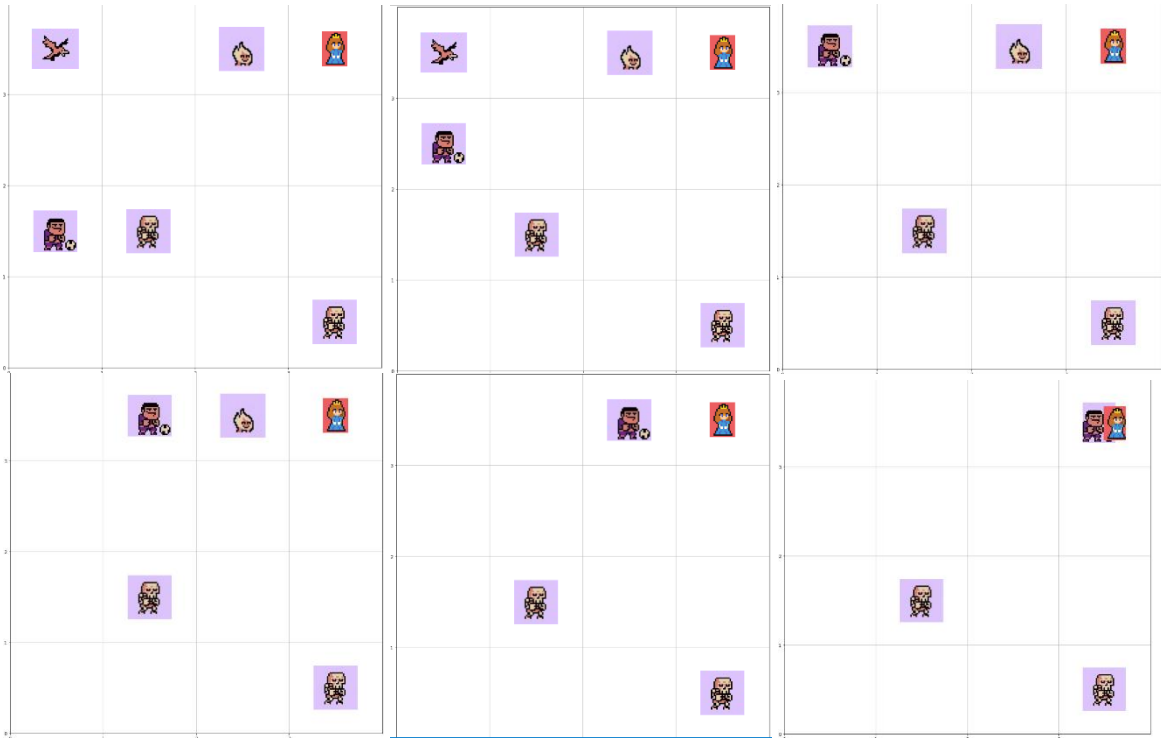
3. Results – Epsilon Decay and Rewards per Episode



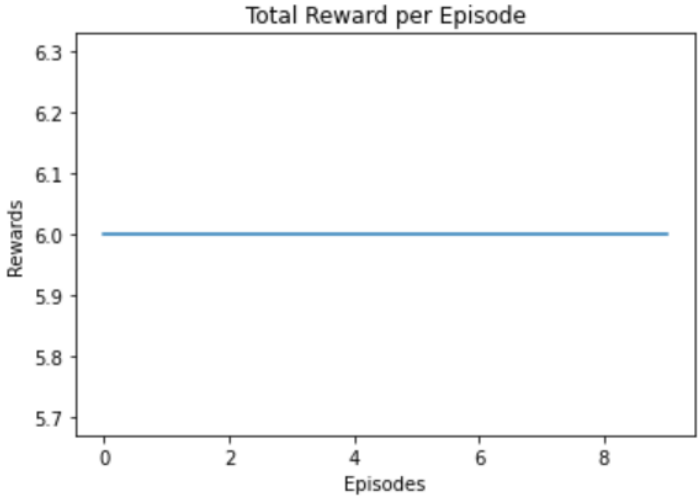
- Epsilon (parameter used for the epsilon-greedy action selection) with a decay factor (0.01) keeps exploring the environment till it reaches the minimum value (0.01). Having a high decay factor will not allow the agent to explore the environment and cannot find the optimal path.
- In this environment the agent explores for around 400 episodes as shown in the graph above.
- The Total Rewards graph shows that during the exploration phase the rewards are very low. The agent starts collecting rewards and avoiding the penalties and gradually the cumulative reward per episode increases (collecting the rewards).

4. Evaluation

Visualization for 1 Episode



From the above visualization we can say that agent has chosen only greedy action from the learnt policy and reaches the goal in 6 steps.



Since it is a deterministic environment the cumulative reward per episode will be constant, using the optimal policy that the agent has learned.

PART 3- Improving DQN & Solving OpenAI Gym Environments

3.1 Double DQN

Double DQN has two estimators unlike DQN, one estimator is used to obtain the best action and the second estimator uses the action from the first estimator to evaluate the Q value.

$$Q^*(s_t, a_t) = r_t + \gamma Q_{\theta'}\left(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a')\right)$$

where , $Q_\theta is\ the\ primary\ network\ and\ Q_{\theta'}$ is the target network

Loss is calculated between the calculated Q values and the primary network parameters. The same model is DQN is reused in DDQN the only difference is instead of evaluating the Q values using the primary network we use a second network which is the target network in this case to evaluate the Q values.

3.2 Improvement Over Vanilla DQN

In Vanilla DQN there is a chance of overestimating the same action which can make the learning unstable whereas in Double DQN having two estimators reduces the chances the overestimating the same action resulting in more stable and reliable learning.

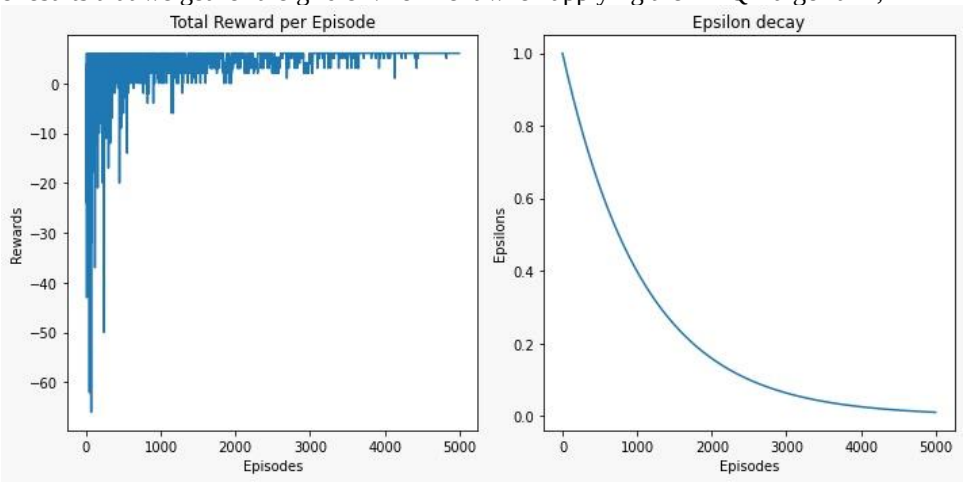
We implemented the DQN and DDQN algorithms on our own grid environment, Cartpole-V1 and for Mountain car environment. Here are the following results for that,

For our grid environment when applying DDQN to the grid environment, we used these parameters,

Parameters

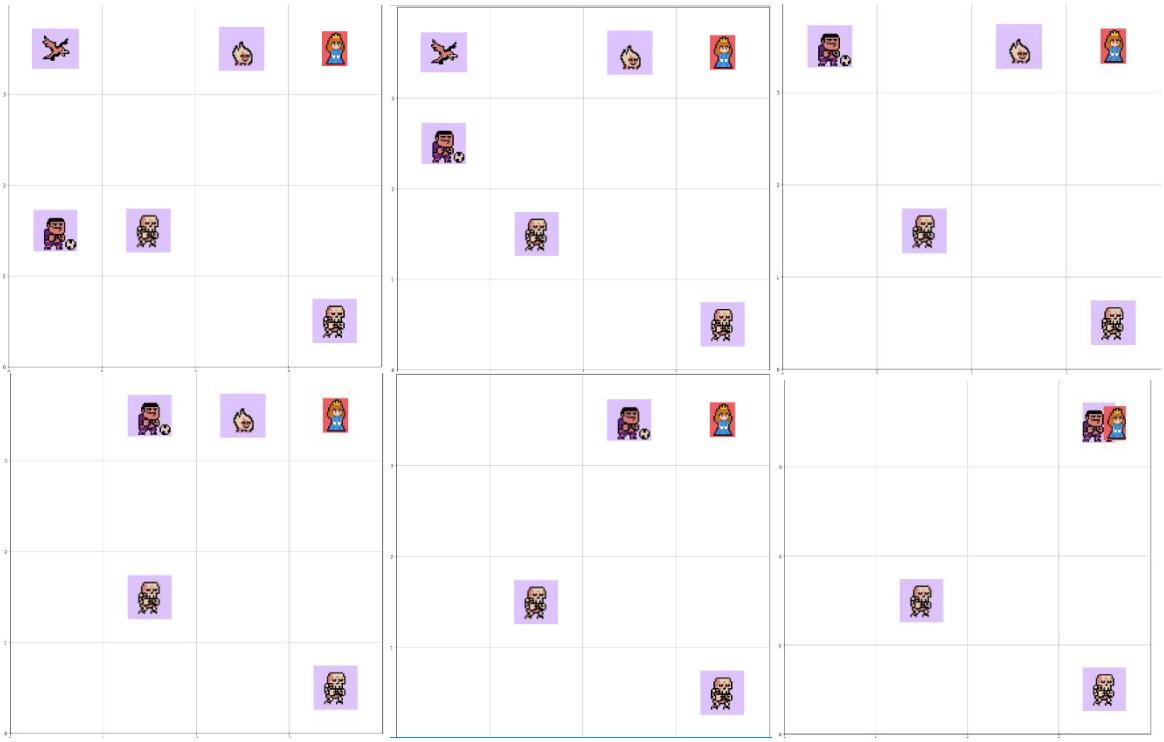
- Initial Epsilon – 1
- Learning Rate – 0.001
- Hidden Nodes - 100
- Hidden Layers – 2
- Replay Size – 400
- Batch Size – 300
- Episodes – 5000
- Minimum Epsilon – 0.01
- Discount Factor – 0.9

Follow are the results that we get for the grid environment when applying the DDQN algorithm,

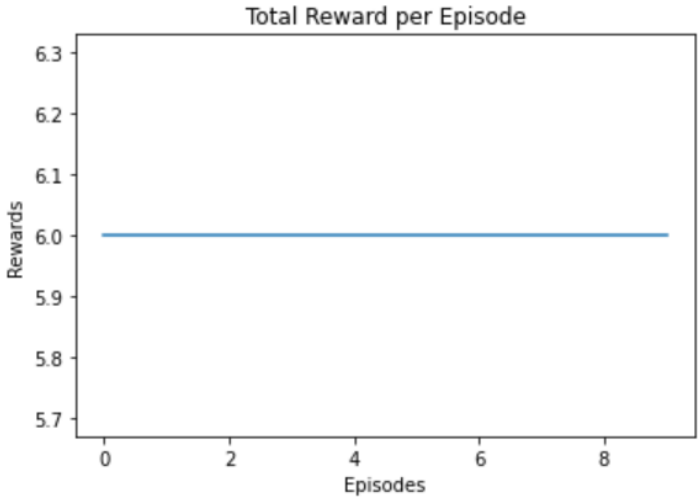


When compared with the DQN algorithm, DDQN converges earlier than that of the DQN algorithm.

Visualization for 1 Episode



From the above visualization we can say that agent has chosen only greedy action from the learnt policy and reaches the goal in 6 steps.



Since it is a deterministic environment the cumulative reward per episode will be constant, using the optimal policy that the agent has learned.

3.3 Show and discuss your results after applying the two algorithms implementation on the environment. Plots should include epsilon decay and the reward per episode.

We have implemented the DQN and DDQN algorithm on two other environments cartpole and mountain car.

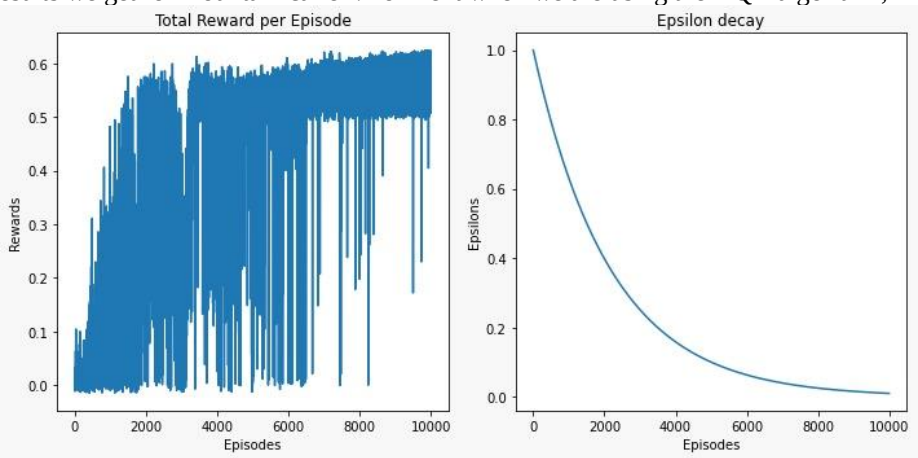
Mountain Car

For mountain car DQN implementation we used parameters as follows,

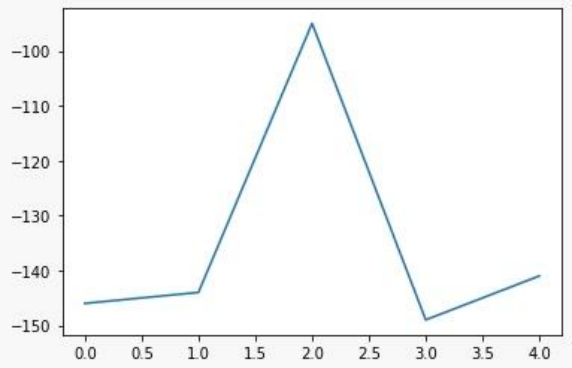
Parameters

- Initial Epsilon – 1
- Learning Rate – 0.01
- Hidden Nodes - 256
- Hidden Layers – 3
- Replay Size – 500
- Batch Size – 256
- Episodes – 10000
- Minimum Epsilon – 0.01
- Discount Factor – 0.99

Below are the results we get for mountain car environment when we are using the DQN algorithm,



The total reward we get after evaluating the model for 5 episodes are as follows,

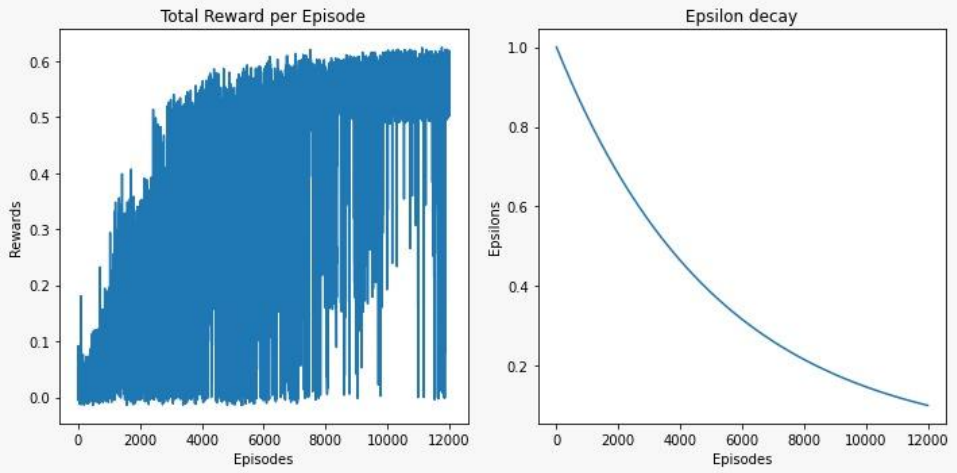


For mountain car DDQN implementation we used parameters as follows,

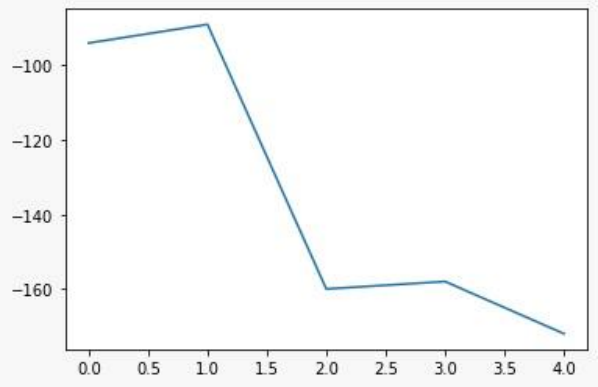
Parameters

- Initial Epsilon – 1
- Learning Rate – 0.01
- Hidden Nodes - 256
- Hidden Layers – 2
- Replay Size – 1000
- Batch Size – 850
- Episodes – 12000
- Minimum Epsilon – 0.01
- Discount Factor – 0.8

Below are the results we get for mountain car environment when we are using the DDQN algorithm,

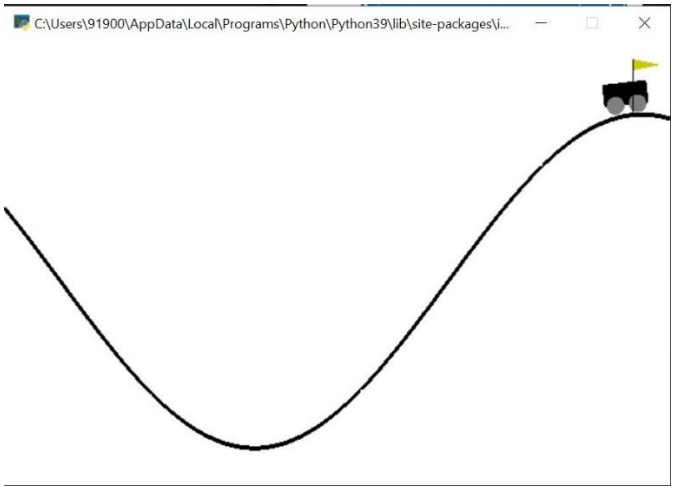


The total reward we get after evaluating the model for 5 episodes are as follows,



In both the cases the car reaches to the goal position. We observed that DDQN algorithm converges faster than that of the DQN algorithm.

Visualization



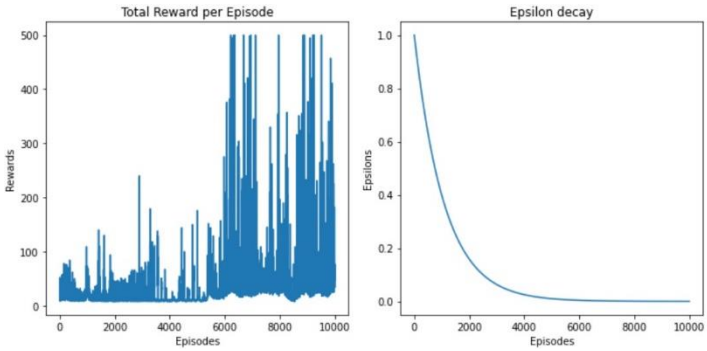
Cartpole Environment

For the cartpole environment when applying DQN algorithm, we used these parameters,

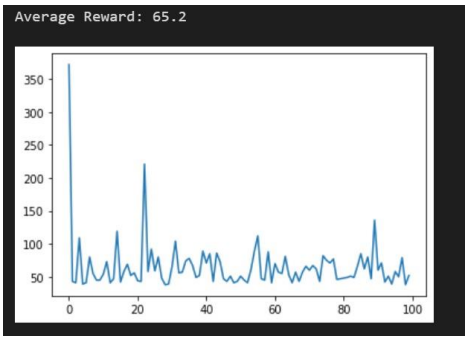
Parameters

- Initial Epsilon – 1
- Learning Rate – 0.00023
- Hidden Nodes - 120
- Hidden Layers – 2
- Replay Size – 500
- Batch Size – 256
- Episodes – 10000
- Minimum Epsilon – 0.0001
- Discount Factor – 1

Below are the results that we get for the cartpole environment when applying the DQN algorithm,



When evaluating the model for 100 iterations we get these rewards,

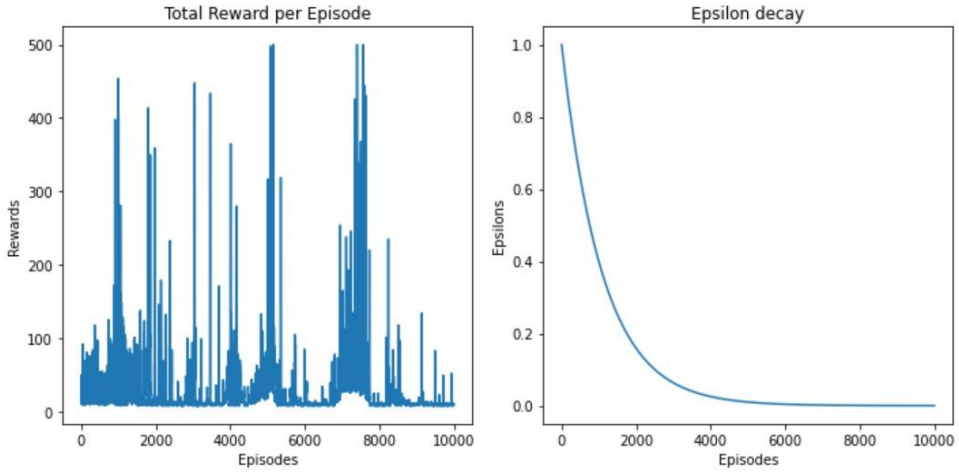


For the cartpole environment when applying DDQN algorithm, we used these parameters,

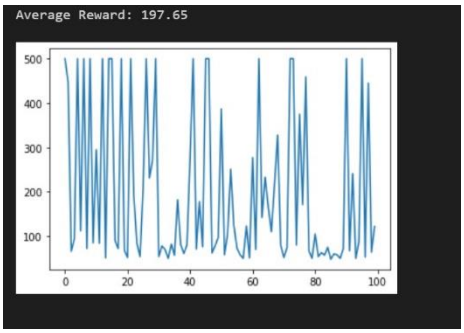
Parameters

- Initial Epsilon – 1
- Learning Rate – 0.00023
- Hidden Nodes - 120
- Hidden Layers – 2
- Replay Size – 500
- Batch Size – 256
- Episodes – 10000
- Minimum Epsilon – 0.0001
- Discount Factor – 1

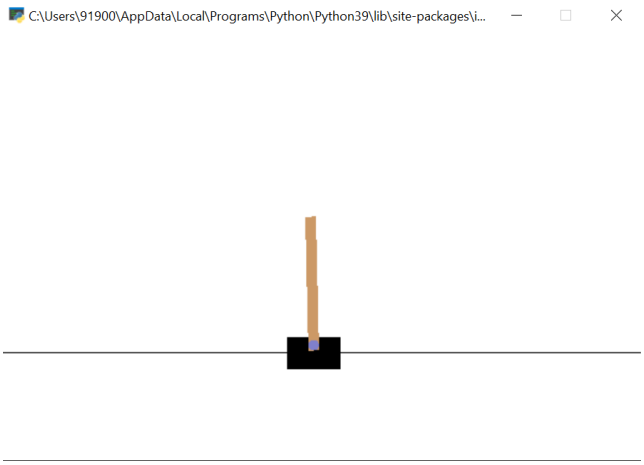
Below are the results that we get for the cartpole environment when applying the DDQN algorithm,



When evaluating the model for 100 iterations we get these rewards,



Visualization



For cartpole environment the DDQN algorithm is working better than that of DQN algorithm.