CSE 546 Reinforcement Learning (Spring 2024)
Assignment 2 – Final Report

The following explains the benefits of using different constructs in a DQN:

1. Using experience replay in DQN: It allows the agent to use past experiences during training and since we randomly sample the episodes, we break the correlations between successive samples of an episode. Finally, experience replay encourages exploration by allowing the agent to revisit past experiences during training.
2. Introducing target network: The purpose of introducing a second neural network is to eliminate the problem of moving targets which facilitates learning of the Q-network and allows the agent to learn optimal policy.
3. Representing the Q-values using an approximation function: This is very useful when the action space/ observation space is huge and maintaining them in a standard Q-table to represent every state-action value becomes in-efficient. Additionally, the approximation gives us the flexibility to use neural networks and other such functions to train the agent.

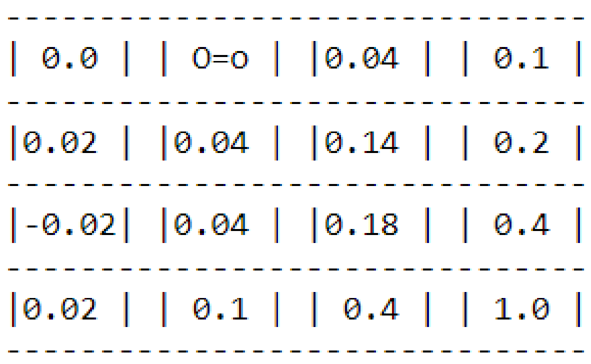
Part 2.1 Implementing DQN & Solving grid-world Environment:

Grid-World Environment

Description: The environment is a deterministic grid world based on Door dash delivery agent who must deliver food to the customer. On his way the agent is met with obstacles and rewards. The objective is to deliver the food to the customer in the least possible time while maximizing rewards.

Actions available: The agent can move in four directions; Up, Down, Left and Right

Rewards available: There are several types of rewards/ penalties. The below picture gives an idea of the reward function used in the environment.



```
---------------------------------
| 0.0 | | O=o | |0.04 | | 0.1 |
---------------------------------
|0.02 | |0.04 | |0.14 | | 0.2 |
---------------------------------
|-0.02| |0.04 | |0.18 | | 0.4 |
---------------------------------
|0.02 | | 0.1 | | 0.4 | | 1.0 |
---------------------------------
```

visualization of the grid environment

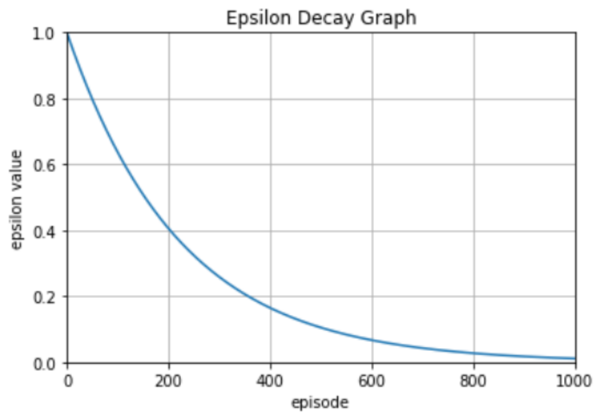
The above image is a simple visualization of an agent in the grid environment. Here the agent is represented by a car symbol(“O=o”). The rewards/ penalties are represented by constant values. The goal state is at the bottom right corner of the grid.

It is worth noting that the environment is modelled in a way that rewards stay alive at their respective positions even if the agent attains the reward at some point during the episode.

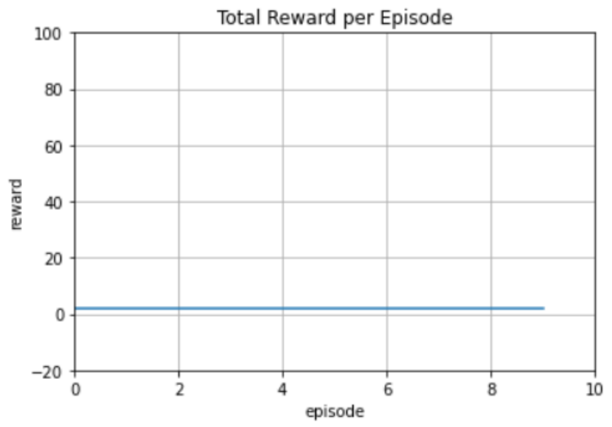
States available: There are 16 states in which the agent can be in.

Termination: The episode is terminated if the agent reaches the goal state, or it exceeds the maximum allowed time steps.

Results: The agent quickly learns the optimal path and reaches the destination while collecting maximum rewards along the way.



Epsilon decay graph

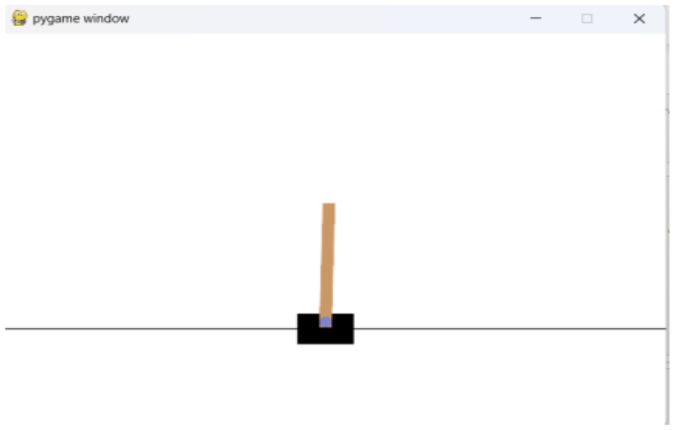


Rewards obtained during evaluation

Part 2.2 Applying DQN to solve various RL problems:

CartPole-V1 Environment

Description: The Cart Pole environment simulates a pole balanced on top of a cart that can move horizontally along a frictionless track. The pole is attached by an un-actuated joint to a cart.



Cart pole environment

Action Space: The agent can take two discrete actions, to keep the pole upright. The following table explains the action space of the agent in cartpole environment.

Action	Representation
Move cart to the left	0
Move cart to the right	1

The velocity that is reduced or increased by the applied force is not fixed and it depends on the angle the pole is pointing.

Observation Space: The state of the environment is represented by a 4-dimensional vector: cart position, cart velocity, pole angle, pole angular velocity.

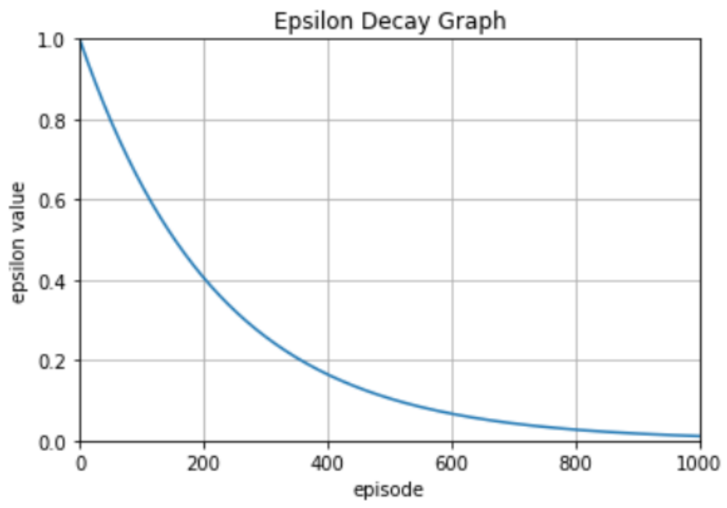
The cart x-position can take values between (-4.8, 4.8). The episode terminates if the cart leaves the (-2.4, 2.4) range.

The pole angle can be observed between (-.418, .418) radians (or ±24°). The episode terminates if the pole angle is not in the range (-.2095, .2095) radians (or ±12°)

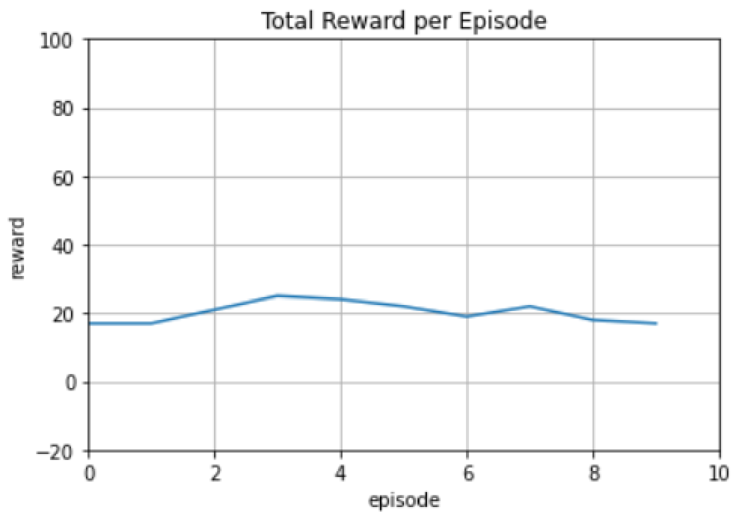
Goal state: The objective is to keep the pole balanced upright by moving the cart to the left or right direction.

Reward structure: The agent receives a reward of +1 for each time step in which the pole remains upright.

Results: Post training, the agent learns keep the pole in an upright position during evaluation, thus we can say that the agent has learnt the optimal policy



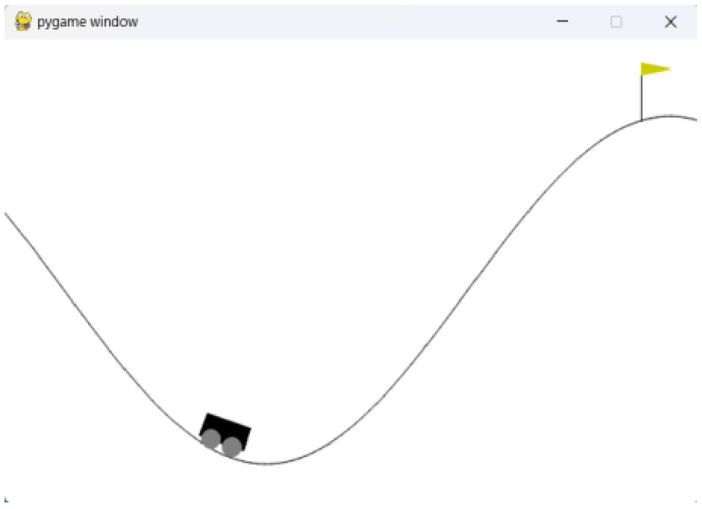
Epsilon decay graph



Rewards obtained during evaluation.

Mountain Car Environment:

Description: In the Mountain Car environment, a car is situated at the bottom of two hills. The car cannot directly drive-up hill so it must learn to use momentum by driving back and forth to build up enough speed to reach the goal.



Discrete mountain car environment

Action space: The action space in this environment is a deterministic discrete one with three actions.

Action	Representation
Accelerate to the left	0
Don’t accelerate	1
Accelerate to the right	2

Observation space: The state of the environment is represented by a 2-dimnesional vector. They individual components of the vector are:

1. Position of the car along x-axis

2. Velocity of the car.

The initial position of the car is assigned a uniform random value in [-0.6, -0.4]. The starting velocity of the car is always assigned to zero.

The episode ends if either of the following happens:

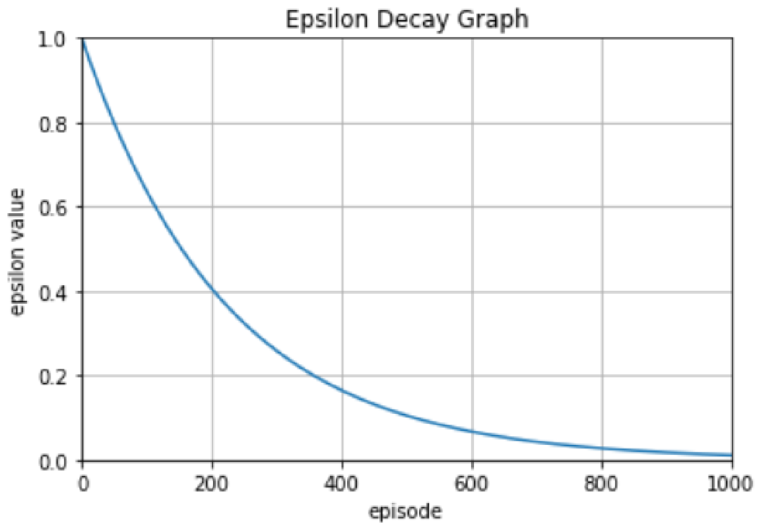
1. Termination: The position of the car is greater than or equal to 0.5

2. Truncation: The length of the episode is 200.

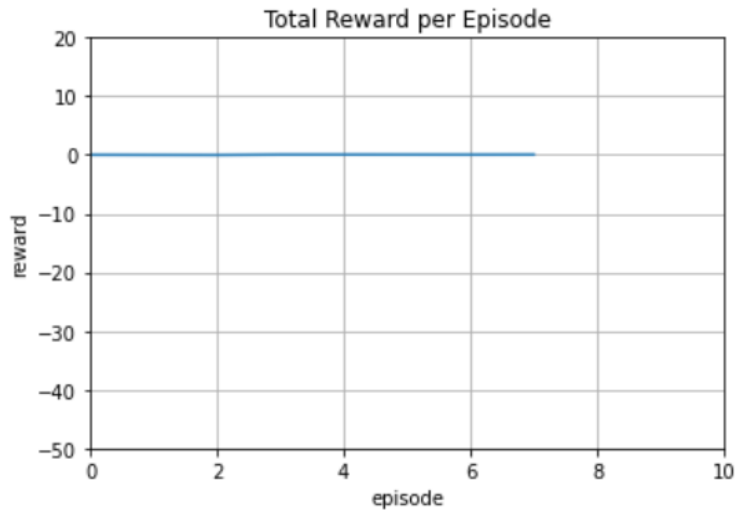
Goal state: The goal of the agent is to strategically accelerate the car to reach the flag post on top of the right hill.

Reward structure: The agent gets a penalty of -1 as a reward until it reaches the top of the right hill.

Results: The agent seems to stay on constant rewards and does not yet learn to take the best path.



Epsilon decay graph



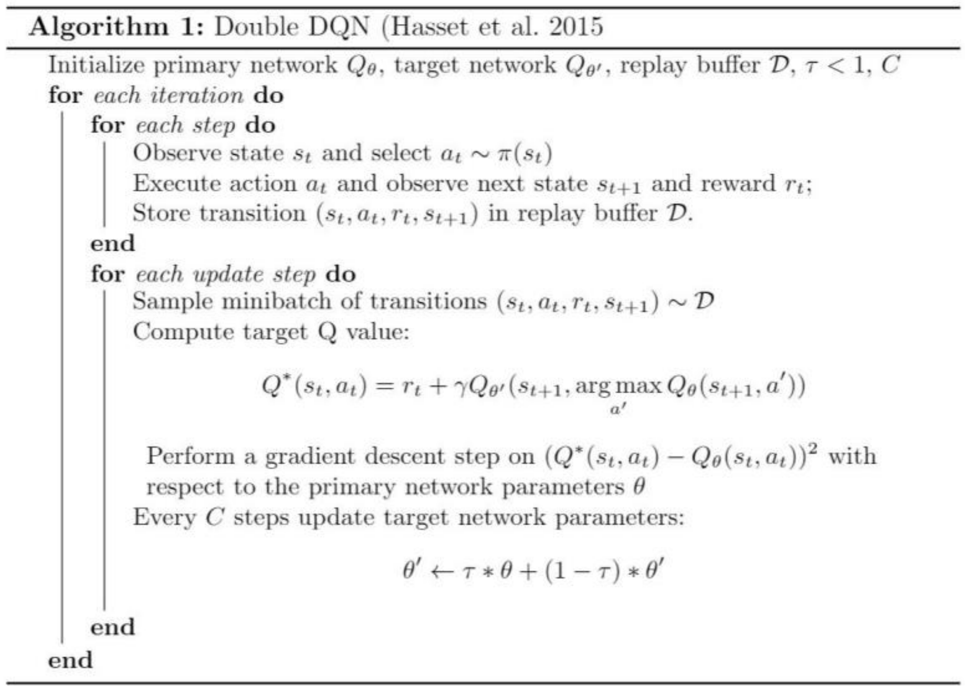
Rewards obtained during evaluation.

Part 3.1 Improving Vanilla version of DQN.

The Double Deep Q-Network (Double DQN) algorithm is an extension of the Deep Q-Network (DQN) algorithm. It addresses the issue of overestimation bias in Q-learning by decoupling action selection from value estimation.

- Like DQN, Double DQN also uses two neural networks: the policy Q-network and the target network.
- Both networks have the same architecture and maintain separate sets of weights.
- During action selection, Double DQN decouples the action selection process from value estimation.
- Instead of directly using the policy Q-network to select the action with the maximum Q-value, it uses the policy Q-network to select the action with the maximum Q-value. Then, it uses the target network to estimate the value of that action.

Double DQN uses two separate neural networks to estimate the Q-values, one network used for action selection and the other for action evaluation. By doing so, Double DQN mitigates the overestimation of Q-values due to the presence of a single network, leading to more stable and accurate Q-value estimates. This is the improvement of double DQN over vanilla DQN.

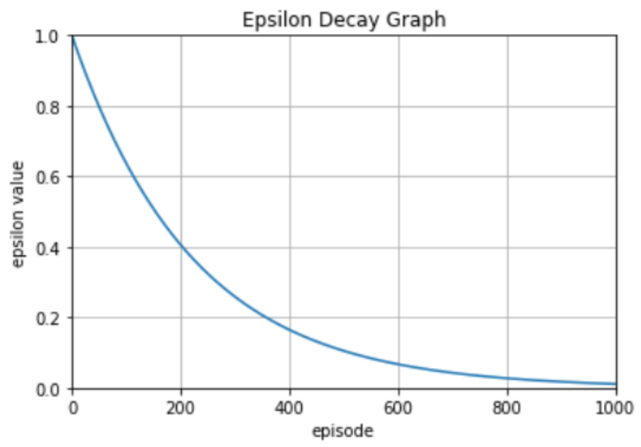


```
Algorithm 1: Double DQN (Hasset et al. 2015
Initialize primary network $Q_\theta$, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau < 1$, $C$
for each iteration do
    for each step do
        Observe state $s_t$ and select $a_t \sim \pi(s_t)$
        Execute action $a_t$ and observe next state $s_{t+1}$ and reward $r_t$;
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$.
    end
    for each update step do
        Sample minibatch of transitions $(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
        Compute target Q value:
            $Q^*(s_t, a_t) = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$
        Perform a gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$ with
        respect to the primary network parameters $\theta$
        Every $C$ steps update target network parameters:
            $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$
    end
end
```

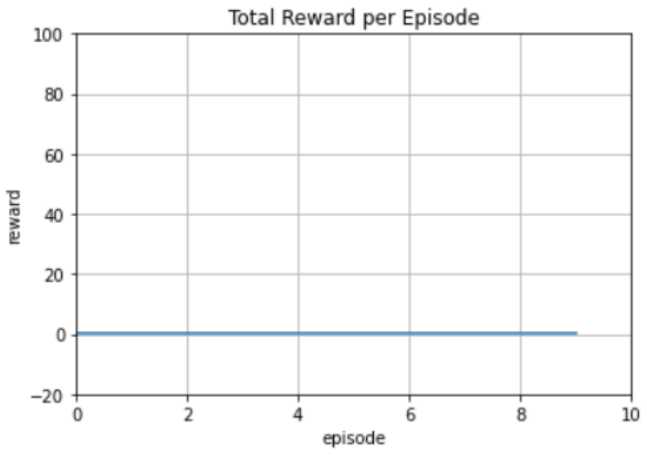
Double DQN algorithm

Grid World Environment

Double DQN is implemented on the grid-world for about 100 episodes with a learning rate of 0.10 and discount factor 0.99

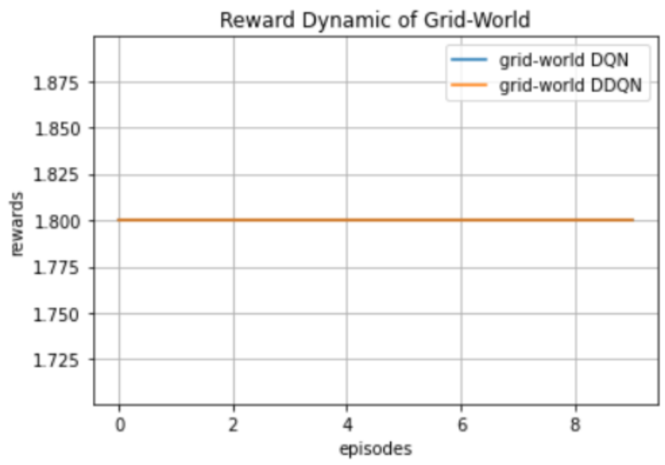


epsilon decay graph



Total reward per episode

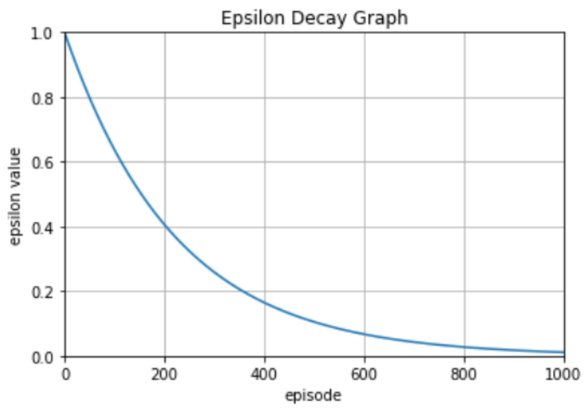
Results: Post training the Double DQN agent, it learns the optimal path while obtaining the most rewards possible. In the reward dynamics, both the regards seem to overlap indicating quite similar performance using both the algorithms



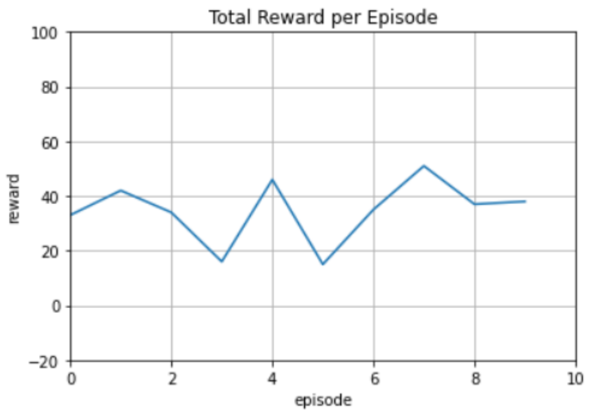
Part 3.2 Applying Double DQN algorithm to Solve Various Problems

Cart Pole Environment

To train the DQN agent in Cartpole Environment, we run it for 1000 episodes and a max timesteps of 200.

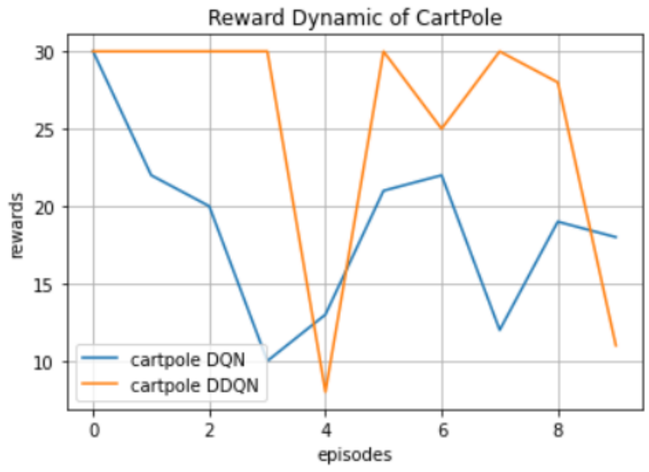


epsilon decay graph



Total rewards per episode

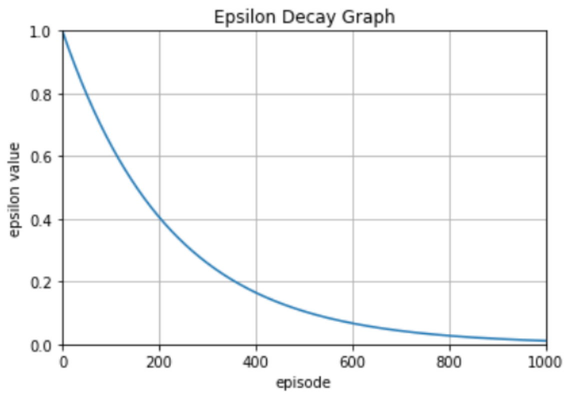
Result: After training the cartpole agent for 1000 episodes, the agent seems to learn to stay upright by choosing appropriate actions. The same is seen based on the reward structure. During evaluation the rewards seem to stay stable without much variance. The reward dynamic however has increased variance in rewards by both the algorithms.



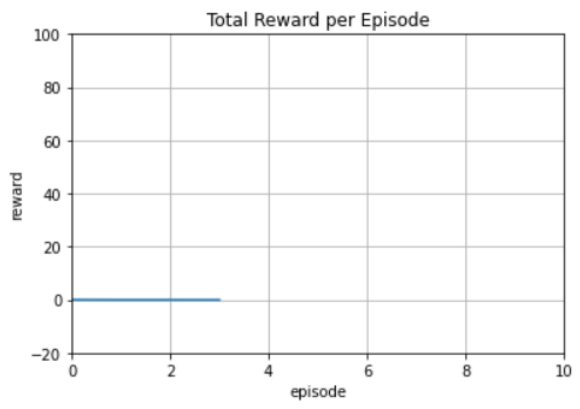
Reward structure of cartpole environment – total reward per episode

Mountain Car Environment

The Agent is trained in the discrete Mountain car environment for about 1000 episodes and 50 timesteps. Below are results of evaluation and relevant graphs.

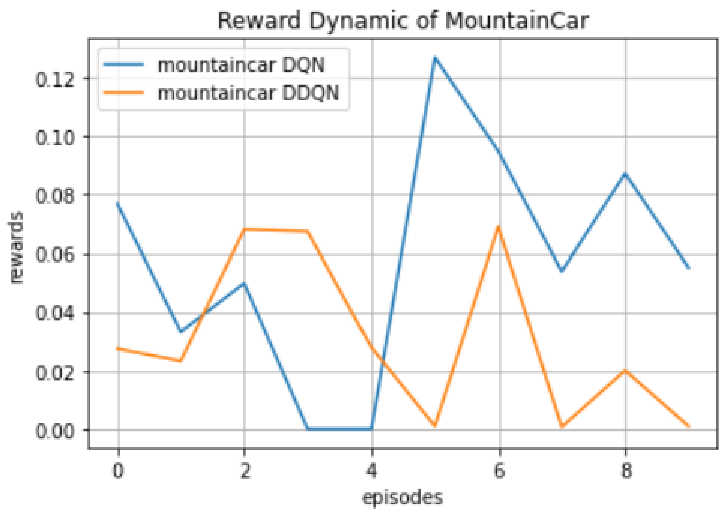


epsilon decay graph



Total rewards per episode during evaluation

Results: After training the agent for about 100 episodes each with 200 timesteps, the agent seems to learn to go top the hill by moving to the left (obtaining less rewards at the beginning). However, it seems that the DQN algorithm performs better in the long run than the Double DQN. The same is clearly shown in the reward dynamic.



Reward dynamic of mountain car

References:

1. https://gymnasium.farama.org/environments/classic_control/(Gymnasium environments)
2. https://gymnasium.farama.org/api/env/ (Gymnasium standard API)
3. https://www.tensorflow.org/api_docs/python/tf (TensorFlow documentation)
4. https://numpy.org/doc/stable/reference/index.html#reference(NumPy Reference)
5. CSE 546 Lecture Slides on UB Learns