Assignment 2 – Deep Q-Networks (Checkpoint)

Part 1: Exploring OpenAI gym Environments

1. CartPole-v1

In this environment, a pole is attached to a cart and the pole moves across the frictionless surface. Here, the goal is to balance the cartpole i.e., prevent it from falling by sliding the cart left or right.

1.1. Observation:

Number	Observation	Minimum	Maximum
0	Cart Position	-4.8	4.8
1	Cart Velocity	-∞	∞
2	Pole angle	-0.418 rad (-24°)	0.418 rad (24°)
3	Pole angular velocity	-∞	∞

1.2. Action space:

For cartpole-v1 environments, the cart can only take two actions to balance the pole above the cart.

$$A = \{Left, Right\}$$

1.3. Rewards:

The reward will be +1 for each step, including the termination step. The maximum reward cartpole can reach is 0.

$$R = \{+1\}$$

1.4. Starting state:

All observations are assigned a uniform random value in $[-0.05 \ldots 0.05]$

1.5. Episode termination:

1. Pole angle > ±12°
2. Cart position > ± 2.4 (center of the cart reaches the edge of the display)
3. Episode length > 200
4. Solved requirements:
   4.1. Average return >= 195.0 over 100 consecutive trials.

2. MountainCar-v0

In Mountain car, a car is situated at the bottom of the valley between two mountains and the car must drive up the hill to reach the top of the mountain. Car must learn to utilize the energy by driving up the hill which is in the opposite direction of the goal to make it to the goal at the top of the mountain near flag.

2.1. Observation:

Number	Observation	Minimum	Maximum
0	Car Position	-1.2	0.6
1	Car Velocity	-0.07	0.07

2.2. Action space:

For mountain car, agent (car) can take up to 3 actions:

$$A = \{Left, Right, Noop\}$$

- Left: Accelerate the car to the left.
- Right: Accelerate the car to the right.
- Noop: Don't accelerate the car.

2.3. Reward:

$$R = \{-1, 0\}$$

Reward will be -1 for each step till the agent reaches the goal(position < 0.5).
Reward will be 0 if the agent reaches the goal i.e at the top of mountain where there is a flag (position = 0.5).

2.4. Starting state:
Initial velocity of the car will be 0. The position of the car is initialized with the uniform random value in [-0.6, -0.4].

2.5. Termination Conditions:
There are two terminating conditions:

1. Episode length > 200
2. Car position > 0.5 (if the car/agent reached the goal)

Part 2: Implementing DQN & Solving grid-world environment

1. Implementation

The project was implemented following OpenAI Gym standards.

1.1. Environment

The environment is a 5*5 Grid World that consists of 25 states. The environment is fully observable, Single-agent, Episodic, and Discrete.

s21	s22	s23	s24	s25
s16	s17	s18	s19	s20
s11	s12	s13	s14	s15
s6	s7	s8	s9	s10
s1	s2	s3	s4	s5

State set: It is the set of all possible states present in the environment.

$$S = \{s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15, s16, s17, s18, s19, s20, s21, s22, s23, s24, s25\}$$

State s1 is the starting state and state s25 is the goal state. States s4, s13 contains reward of +5 and +10 respectively. State s16 contains a reward of -3. States s9, s12 are dead states i.e., when an agent reaches that state the environment resets and the agent receives a reward of -50 in each state. State s25 contains a reward of +50. All the other states have a reward of -1. An episode ends if the agent reached the goal state or any one of the dead states.

The environment is deterministic:

- **Deterministic**: In the deterministic environment the probability of an action *a* in state *s* is certain i.e. there is no randomness or uncertainty. Example: If there are four actions (up, right, left, down) if the agent chooses to go up, the probability of that action in state *s* is 1 and all other actions are 0.

$$P(s', r|s, a) = \{0, 1\}$$

1.2. Action set

It is the set of actions that an agent can take in the environment. For the grid world environment, the action set is given below:

$$A = \{Up, Down, Left, Right\}$$

1.3. Rewards

Reward set: It is the set of all possible rewards an agent can receive from the environment after taking an action. For the grid world environment, the reward set is as follows:

$$R = \{-1, -3, 5, 10, -50, 50\}$$

2. Visualization of the environment

Below are some screenshots of the environment taken at different timesteps:

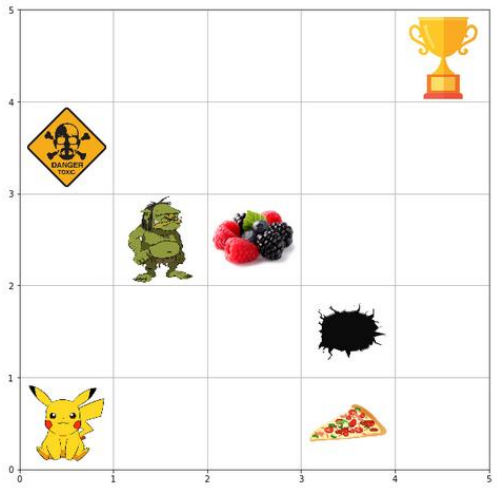

Figure 1: Initial State

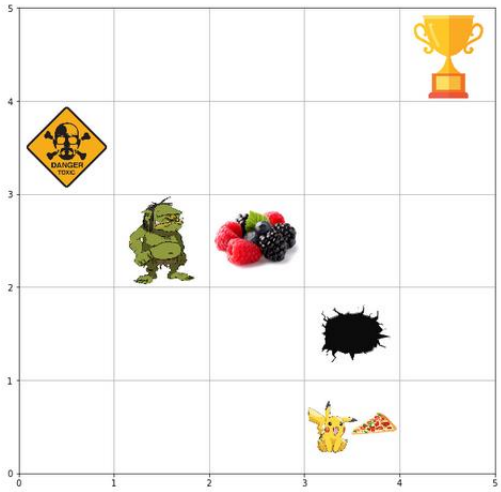

Figure 2: Agent received reward

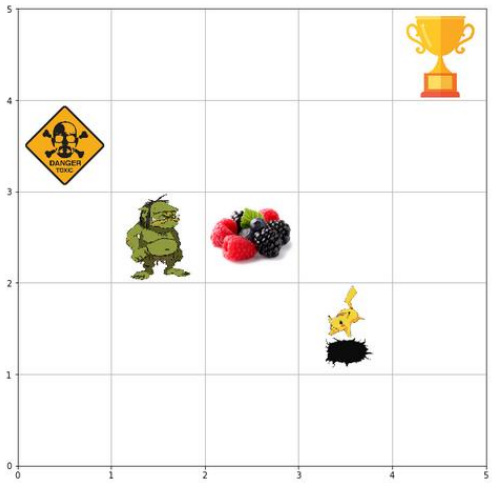

Figure 3: Agent fell in the pit

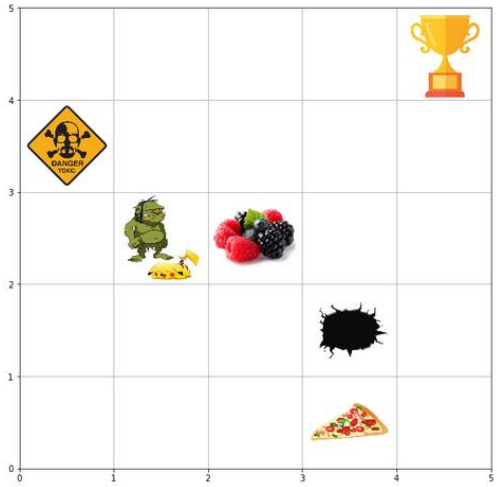

Figure 4: Agent killed by the monster

Pikachu represents the agent in the starting state (bottom left corner) and the trophy represents the goal state. Pizza and Berries represent rewards of +5 and +10 respectively. The danger sign represents a reward of -3. Monster and Pit are dead states i.e. when an agent reaches that state the environment resets and the agent receives a reward of -50 in each state. The trophy represents is our goal state and contains a reward of +50.

3. Deep Q-Network

In a problem statement where the number of states and actions is small, it was easy for us to solve the problem using any of the tabular methods. These tabular methods may not be that convenient when the number of states and actions is huge. In such situations, we want a function approximator that can approximate the value function and be scalable. We all know that neural networks are universal functional approximators. This gave rise to a set of techniques where we combine our previously known knowledge of tabular methods with the use of neural networks to solve the reinforcement learning problem. In Deep Q-network the input is the state and output is the Q-value for each action in that state.

In Deep Q-Network (DQN) we represent the value function by deep Q-network with weights w.

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

The objective function is represented by the following equation:

$$\mathcal{L}(w) = \mathbb{E}[\left(r + \gamma \, max_{a'} Q(s', a', w) - Q(s, a, w)\right)^2]$$

But using the naive q-learning has its own sets of problems with neural nets:

1. A neural network assumes the data to be Independent and Identically Distributed (IID). But in reinforcement learning, as the successive sample are correlated i.e., non-iid.
2. Policy changes rapidly with slight changes in Q-values. As the weights update, it causes the target to change as well which causes the problem of moving the target.
3. The scale of reward and Q-values are unknown due to which it can be unstable when backpropagated.

To overcome these issues there are various techniques developed:

1. Use of experience replay:
    - In this technique, we save the tuple (s, a, r, s') in memory D after each step following the epsilon-greedy policy
    - After which we sample random samples from the memory and use them to minimize the loss function
    - This breaks the correlation and brings us back to the iid setting
2. Freeze the target Q-network:
    - We create two separate neural networks that represent the target network $w^-$and policy network w
    - We freeze the target network and update the weights of the policy network during the backpropagation step
    - After certain episodes/timesteps, we synchronize the weights $w^- \leftarrow w$
    - This helps avoid the oscillations and break the correlations between Q-network and the target

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s'\sim D}[(r + \gamma \, max_{a'} Q(s', a', w^-) - Q(s, a, w))^2]$$

3. Clip the rewards or normalize the network adaptive to the sensible range.

4. Double Deep Q-Network:

In Double Deep Q-Network, we use two estimators. One estimator will estimate the best action and another estimator will be used to evaluate the Q value for the action which was passed by another estimator. We will use two networks i.e. policy network and target network as two different estimators.

Update function for Double DQN:

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

4.1. Improvement over vanilla DQN :

Only difference between DQN and Double DQN is how we estimate the target values. After every C steps, we synchronize the target and policy network.

Since in Deep Q-Network, the optimal policy is a greedy policy based on the current action values, which is defined based on the maximum operation. Consider if the right action is taken by the agent and by using max operator we estimate the value to be positive but if the reward is drawn from a normal distribution with a mean of -1 and variance 0.5. Thus, the expected return for taking a right from that state is a mistake. But still, the agent will take action to go right. This is call maximization bias. To overcome this issue we maintain two Q-Networks so that the chances of overestimation for the same action are reduced.

5. Results

Below are the graphs for different replay memory sizes:

5.1. DQN on Grid World Environment:

5.1.1. Replay Memory size: 128

Below we can observe that the epsilon is decaying over the number of episodes. Total reward per episode is fluctuating in the beginning as the agent is exploring the environment and then stabilizes as the agent takes greedy actions. Agents received the maximum possible reward while taking greedy actions. This can be validated in the average reward per 50 episodes graph, in gradually decreases as the agent explores and learns more about the environment. As the agent takes greedy actions the agent reaches the goal in the optimal number of timesteps. In the cumulative reward overall episodes, it's increasing.

Since the replay memory is small the memory stores more and more recent experiences as the older experiences get removed. As the agents start to take greedy actions it exploits the environment to obtain the maximum reward which gets saved in the memory. After some episodes, the memory can be full of experiences that are optimal based on greedy actions. Hence, in the total reward per episode graph we can see the after 410 episodes the graph is almost constant, and the agent receives the maximum reward.

But having a small replay memory can have adverse effects as well and the agent can overfit the data/experiences which may not help to generalize. Since the grid world is a small and easy environment we didn't face this issue. But this may not be in the case of a complex environment.

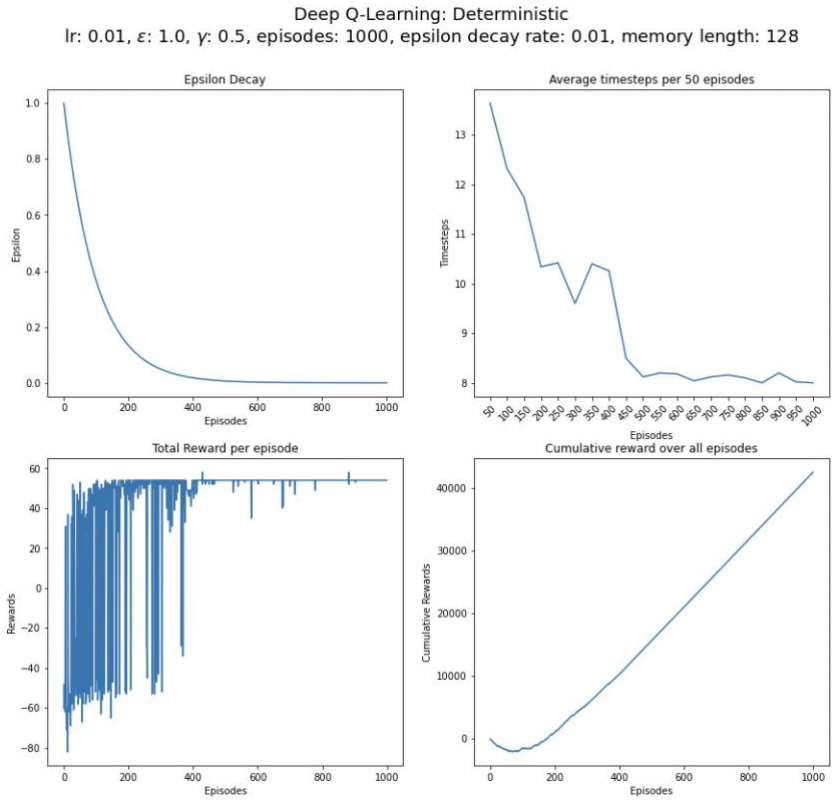

Figure 5: Training results for replay memory size 128

After training, evaluating the agent:

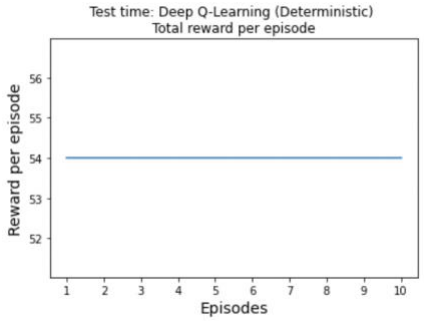

Figure 6: Agent test results for replay memory size: 128

The above graph (Figure 6) showcases the total reward per episode for 10 episodes. The agent learns the optimal policy and follows the path to obtain the maximum reward.

5.1.2. Replay Memory size: 500

The total reward per episode graph converges as the number of episodes increases also due to the agent taking more greedy actions than random actions based on the epsilon decay. The average timesteps per 50 episodes show a gradual decline as the agent learns the optimal policy. Similarly, cumulative reward per episode can be seen increasing as the agent learns the optimal policy.

Compared to the previous trial, here we have increased the replay memory size to 500. This enables to store more experiences in the memory. As the memory size is large it is less likely to sample the correlated experiences and makes the neural network more stable. But it also may be the case that when the agent is taking greedy actions and the neural network gets trained on the very old experiences. If the experiences are not good, it can cause the weights to change in the wrong direction. Hence, sometimes even though the number of episodes is increasing, and the agent is taking greedy action, bad experiences can cause the episodes to end.

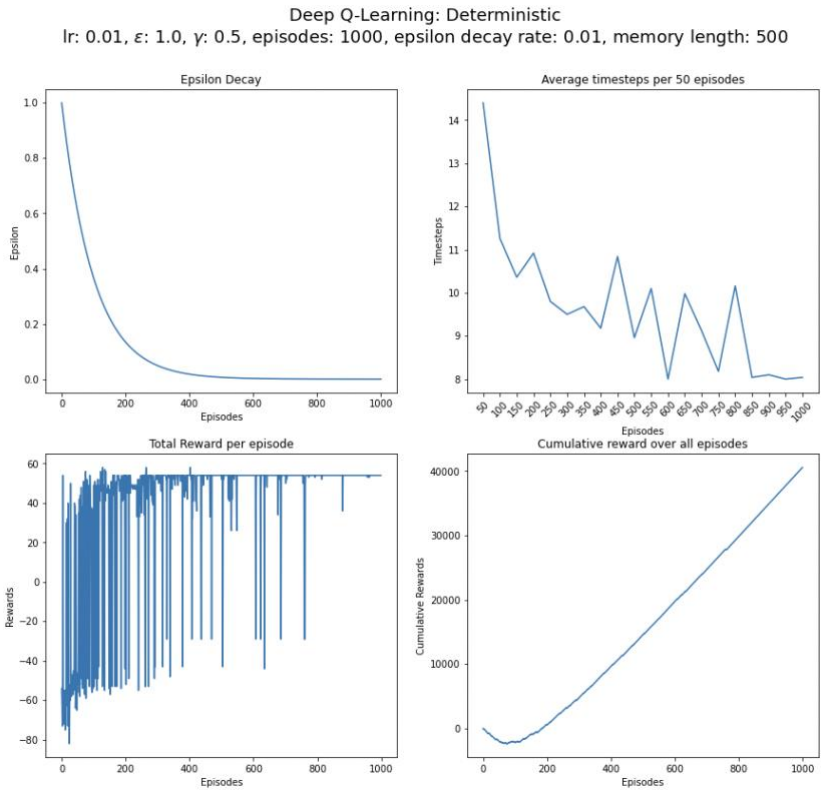

Figure 7: Training results for replay memory size 500

After training, evaluating the agent:

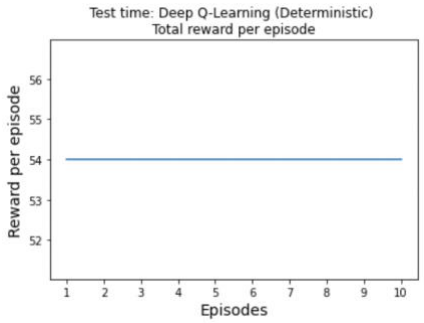

Figure 8: Agent test results for replay memory size: 500

The above graph (Figure 8) showcases the total reward per episode for 10 episodes. The agent learns the optimal policy and follows the path to obtain the maximum reward.

5.1.3. Replay Memory size: 1000

The total reward per episode graph converges because the agent takes more greedy actions than random actions based on the epsilon decay as the number of episodes increases. The average timesteps per 50 episodes show a gradual decline as the agent learns the optimal policy. Similarly, cumulative reward per episode can be seen increasing as the agent learns the optimal policy.

Like the above trial, we have further increased the memory size to 1000. Here we store even more experiences. Similar results but with more variation can be seen in the total reward per episode graph due to the problem we discussed in section 4.2.

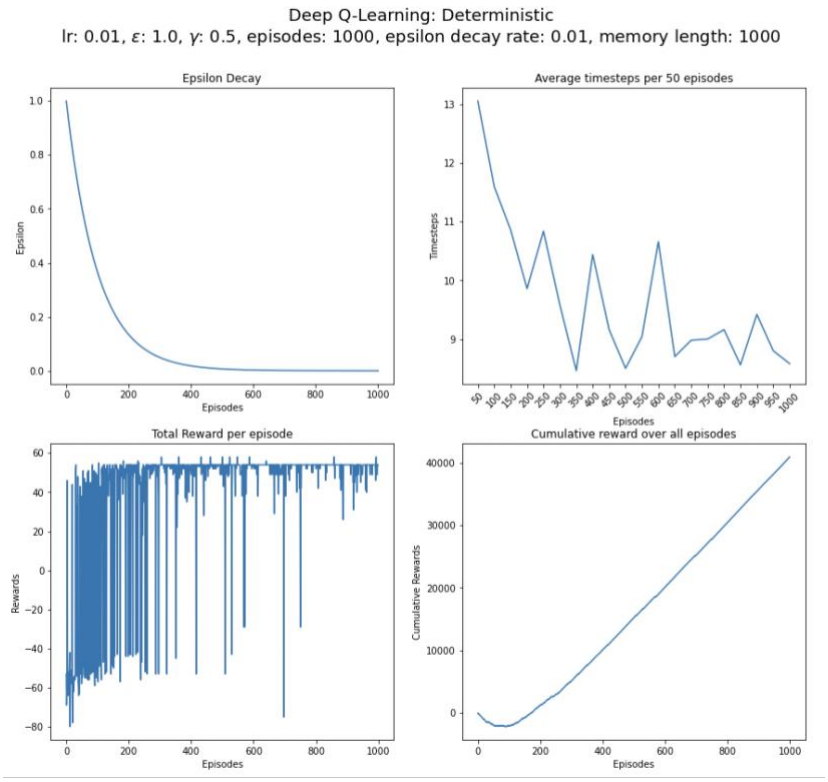

Figure 9: Training results for replay memory size 1000

After training, evaluating the agent:

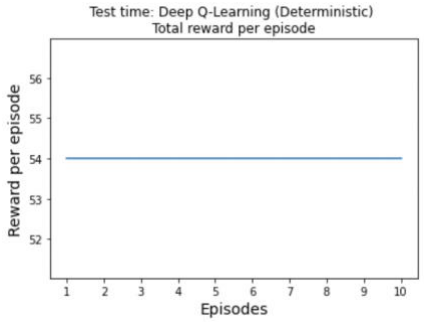

Figure 10: Agent test results for replay memory size: 1000

The above graph (Figure10) showcases the total reward per episode for 10 episodes. The agent learns the optimal policy and follows the path to obtain the maximum reward.

5.2. Double DQN on Grid World Environment:

For the same memory length i.e. 1000 we can see some differences in the results when using DQN and Double DQN for the gird world environment. In DQN the graph (Figure 9) for total reward per episode was varing till the end of 1000 episodes but compared to Double DQN we can see a straight line after 600 odd episodes. This means that the using the double DQN the agent converged comparatively fasters and the results are more stable.

To support the same, in the figure 11 we can see that the average timesteps per 50 episodes graph is decreasing slowly and the cumulative reward had a small dip in the beginning but started increasing as the agent started to learning more.

After training, evaluating the agent:

The below graph (Figure 12) showcases the total reward per episode for 10 episodes. The agent learns the optimal policy and follows the path to obtain the maximum reward.

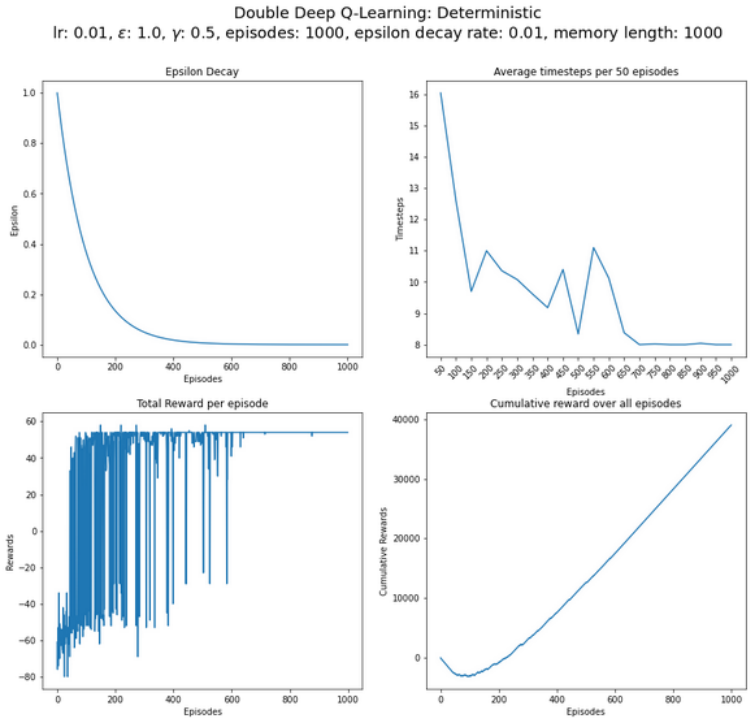

Figure 11: Double DQN results on Grid World Env

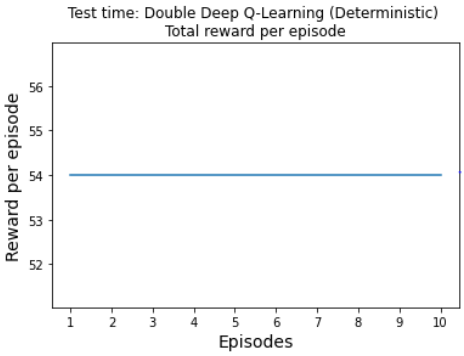

Figure 12: Test results of Double DQN on Grid World Env

5.3. DQN on CartPole-v1:

Since CartPole is a challenging environment compared to grid world we have incorporated some techniques to improve the performance and make the agent more stable. CartPole is challenging because the agent receives a reward of +1 for each step and also then the agent falls i.e. the episode terminate the agents receives a +1 reward. The environment is challenging because the agent does not receive a negative reward to indicate that a bad action was performed.

Here we terminate the training when the average reward for the last 10 episodes is greater than 475 then we stop the training. We do this to avoid the agent learning some bad actions in the later stages. Also, rather than training the Q-network everytimestep we train the network every 10 timesteps.

Similar functionality with some change in values are applied to DQN and Double DQN algorithms for CartPole and Moutain Car.

In the graph below we can see that the total reward per episode is increasing and converging to its optimal values after 550 odd episodes. The variations in the graph can be seen due to the random actions performed by the agent while exploring and slowly converges to the optimal value as the agent takes more greedy actions.

Likewise we can see in the average timesteps per 50 episodes that the timesteps are increasing as the agent is able to balance the pole for longer time and receive more rewards. Here, we have used the default learning rate for Adam optimizer.

The episode decay graph is short because the training was terminated before the value set for total episodes as 1000. Epsilon decay graph can be seen in the figure 14

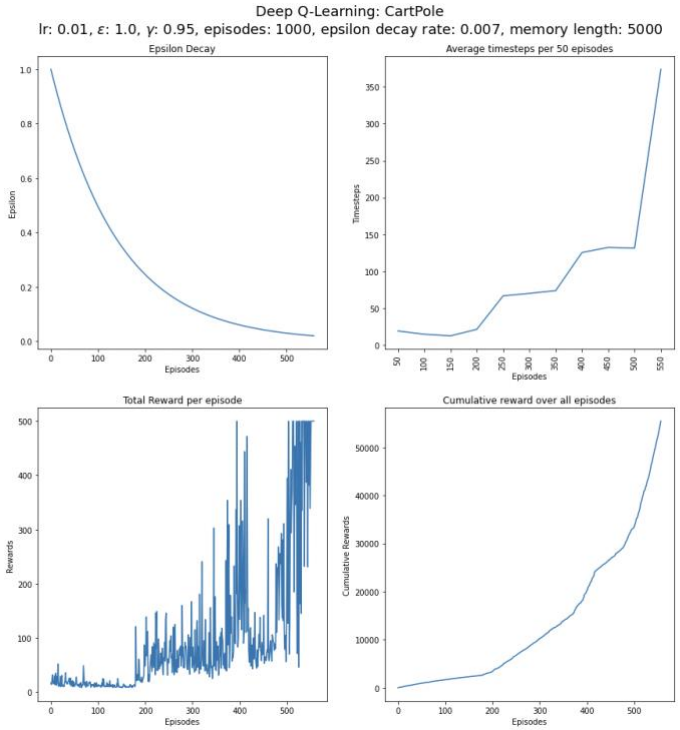

Figure 13: DQN results on Cartpole-v1

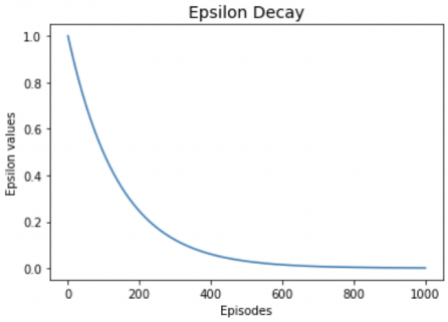

Figure 14: Epsilon decay for DQN Cartpole

After training, evaluating the agent:

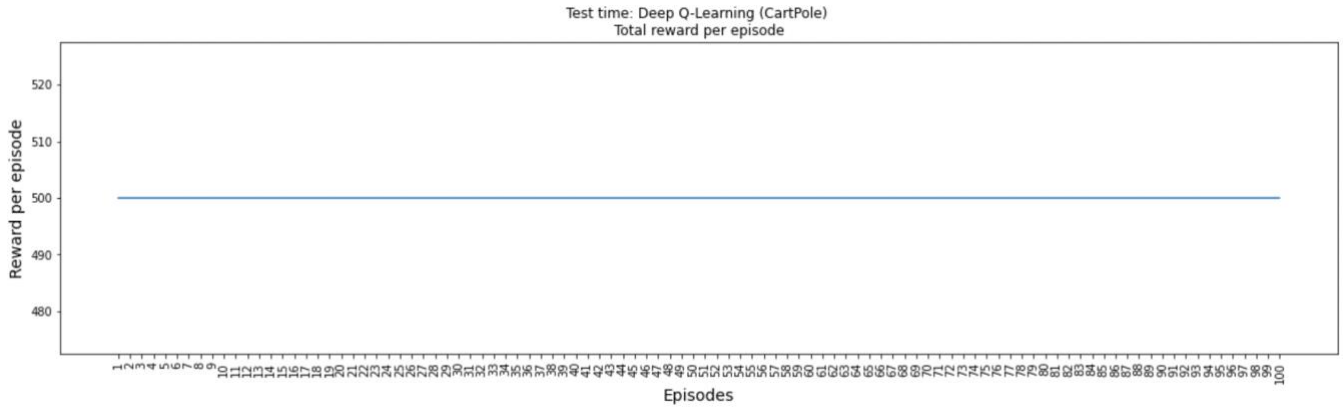

Figure 15: Test results of DQN on Cartpole

The above graph (Figure 15) showcases the total reward per episode for 100 episodes. The agent learns the optimal policy and follows the path to obtain the maximum reward.

5.4. Double DQN on CartPole-v1:

In this setting we terminate the training when the average reward for the last 10 episodes is greater than 475 and we train the agent after every 10 timesteps. Here we can see in the total rewards per episodes that compared to DQN the agent has not overestimated the q-values since the graph is increasing slowly and reaching its optimal value. Similar behavior can be seen in the average timesteps per 50 episodes graph, the graph is increasing as the agent learns how to better manage the pole and receive more rewards. Likewise cumulative rewards are also increasing. Here, we have used the default learning rate for Adam optimizer.
The episode decay graph is short because the training was terminated before the value set for total episodes as 1000. Epsilon decay graph can be seen in the figure 17.

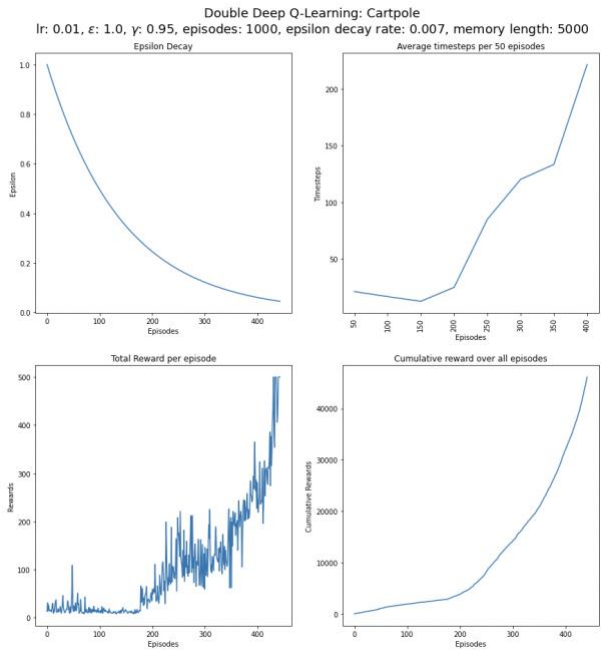

Figure 16: Results of Double DQN on Cartpole

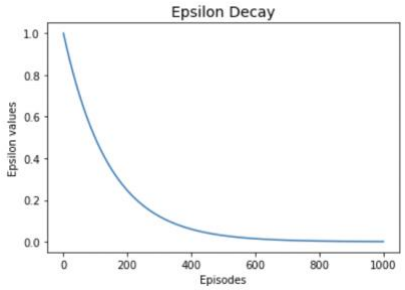

Figure 17: Epsilon Decay graph for Double DQN on Cartpole

After training, evaluating the agent:

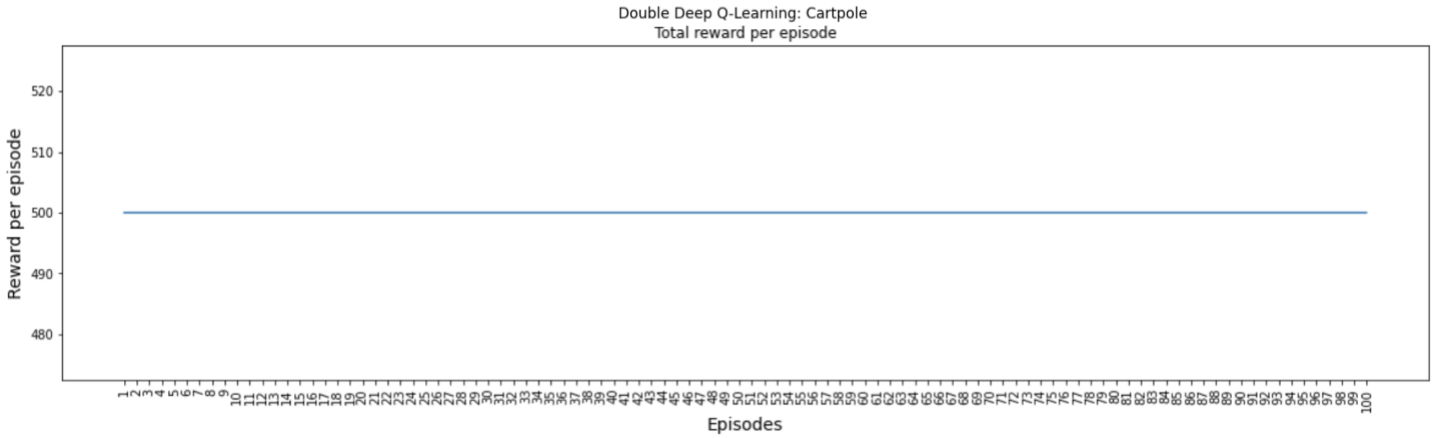

Figure 18: Test results of Double DQN on Cartpole

The above graph (Figure 18) showcases the total reward per episode for 100 episodes. The agent learns the optimal policy and follows the path to obtain the maximum reward.

5.5. DQN on MountainCar-v0:

This environment is also challenging because for every action the agent is getting -1 reward and only after reaching the goal it receives reward 0. In MountainCar we terminate the training when the average for the last 10 episodes are greater than -100 and we update the model after every 5 timesteps.

In the total reward per episode graph we can see that it has converged after 1200 odd episodes. Initial the graph is unable because the agent is performing random actions as the agent is exploring and then the graph can be seen converging a bit in the middle but agent the graph varies. This can be because the weight update might have happened in the wrong direction. As the agent learns more the agent converges to the optimal values.

Similar results can be seen in the average timesteps per 50 episodes are decreasing till episode 950 but after that the number of timesteps is increasing suddenly then agent decreases as the agent learns and follows the optimal path.

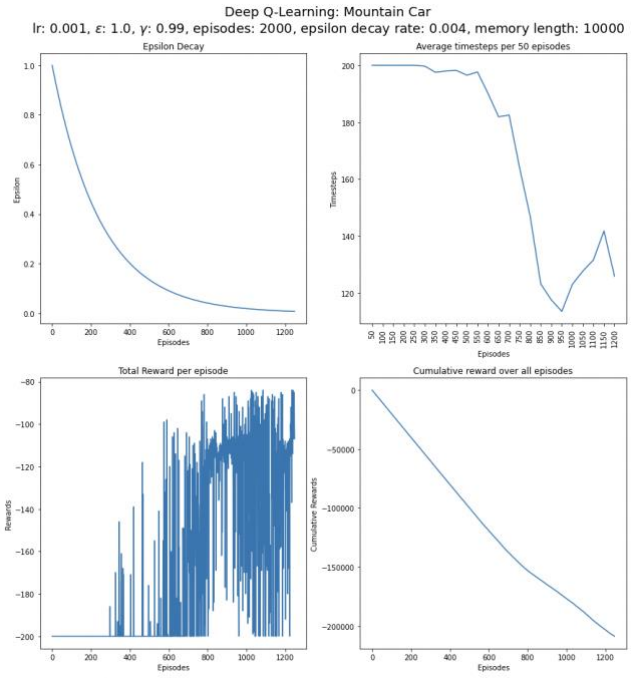

Figure 19: Results of DQN on Mountain Car

The episode decay graph is short because the training was terminated before the value set for total episodes as 1000. Epsilon decay graph can be seen in the figure 20.

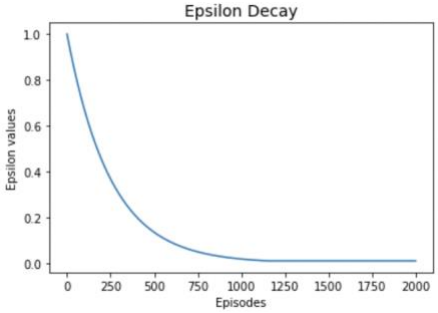

Figure 20: Epsilon Decay graph for DQN on Mountain Car

After training, evaluating the agent:

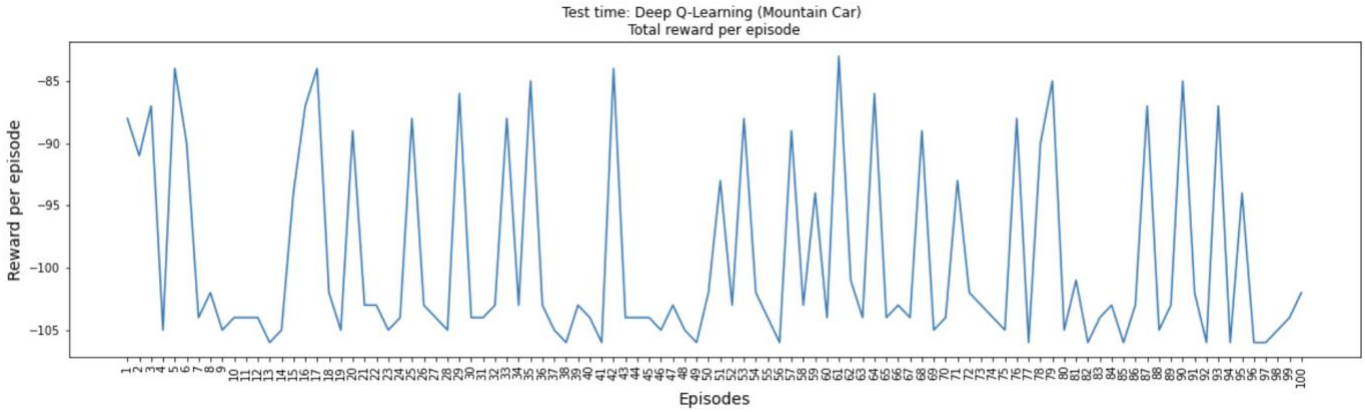

Figure 21: Test results of DQN on Mountain Car

The above graph (Figure 21) showcases the total reward per episode for 100 episodes. The agent learns the optimal policy and follows the path to reach the goal.

5.6. Double DQN on MountainCar-v0:

In double DQN we have set the termination condition to be average rewards for the last 10 episodes be greater than -90 and we train the model after every 5 timesteps. Here we can see similar variation compared to that of DQN but they are lesser. This is because the double DQN does not overestimate like DQN. Over multiple trials we have observed that double DQN provides more stable results compared to DQN algorithm. Here, we have used the default learning rate for Adam optimizer.

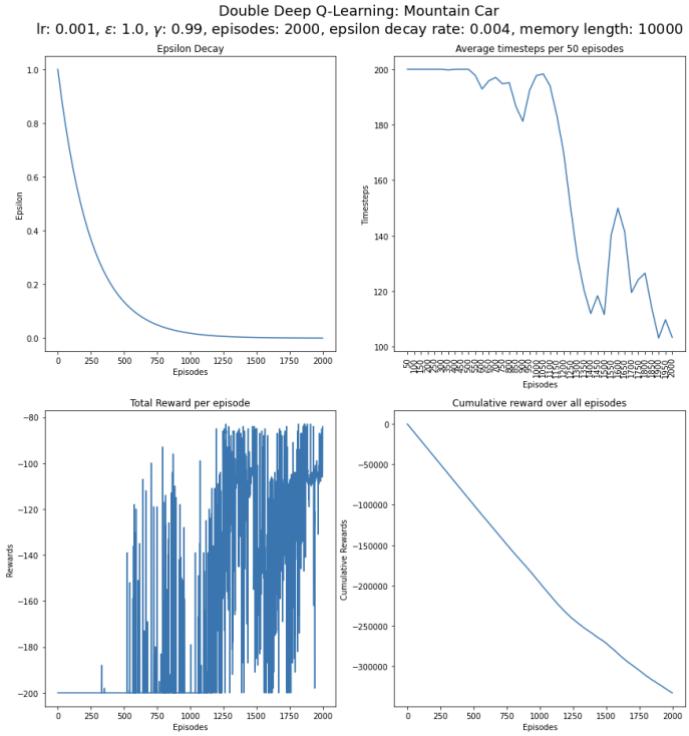

Figure 22: Results of Double DQN on Mountain Car

After training, evaluating the agent:

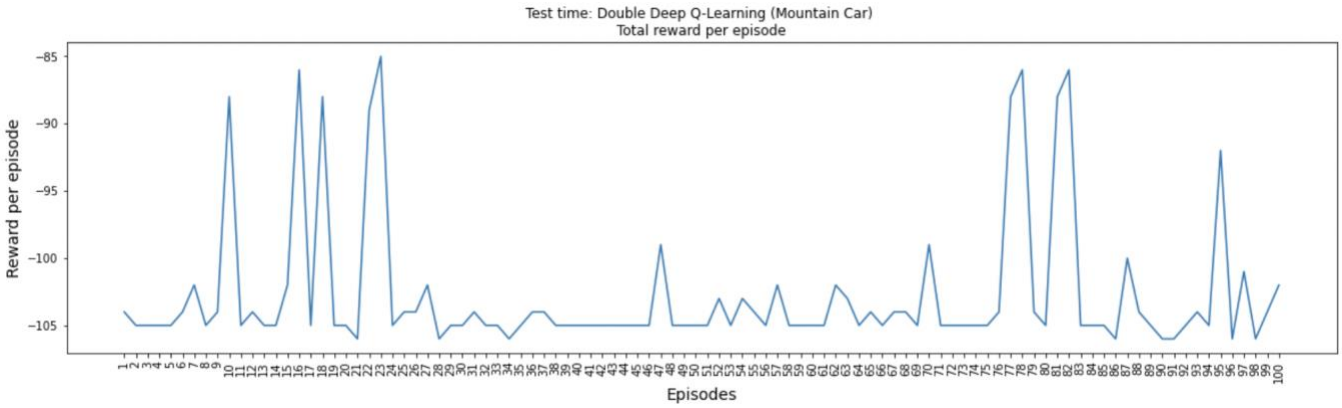

Figure 23: Test results of Double DQN on Mountain Car

The above graph (Figure 23) showcases the total reward per episode for 100 episodes. The agent learns the optimal policy and follows the path to reach the goal.

6. Conclusion

The size of the replay memory does affect how the neural network gets trained that was observed when we applied DQN on the grid world environment. It is necessary to experiment with different replay memory lengths for any problem statement so that the network can learn in a better way. Both the environment performed differs on the complexity of the environment. Since the grid world was easy both learnt compararively faster. As the environment gets complex the algorithm struggles and tuning of hyperparamters because more crucial. On the same environment, when we compare the performance of the two algorithms and see that the Double DQN produces stables results in the long run compared to DQN.

7. Contribution Summary

Team Member	Assignment part	Contribution (%)
Sagar Jitendra Thacker	Part 1,2	50%
Anuja Raghunath Katkar	Part 1,2	50%