DQN and DDQN Agents

Kolla Rohith Reddy | rkolla@buffalo.edu

PART – 1

1.

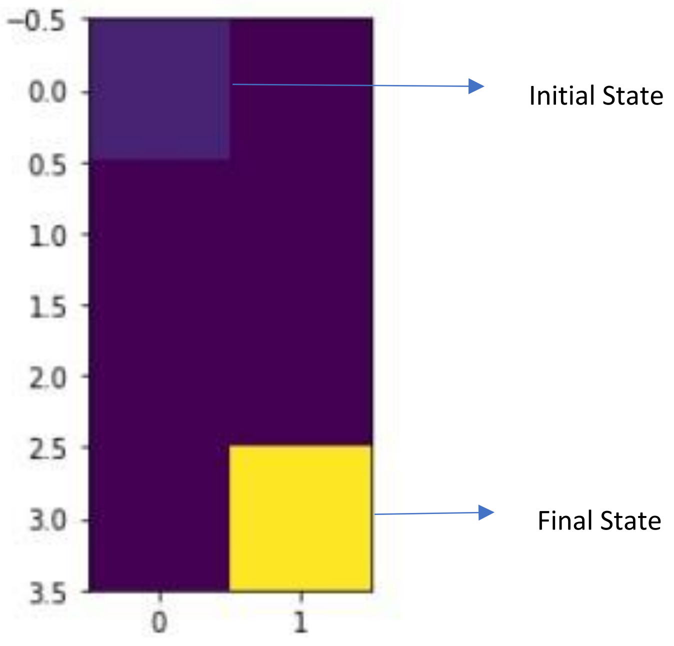
- Experience Replay in DQN involves storing experiences in a replay buffer which are later sampled to train the network. This helps decorrelate the data and is especially useful in situations where gaining experience is costly. It also helps in better convergence of the function approximator. Generally, larger sizes for experience replay provide better results since the sample taken from it will have lesser correlation.
- The target network is a key part of DQN which helps break correlation between the Q-network and the target. It also helps avoid oscillations due to incremental updates.
- Creating and updating q tables is very memory intensive and is impractical in most situations with large data. Using function approximator to represent the Q function makes the process a lot more computationally efficient.

2. The grid world environment used is similar to the deterministic environment used in the previous assignment which is defined with 8 states in a 4 x 2 grid where the initial state is at [1,1] and the goal is to reach the final state at [4,2].

4 actions (down, up, left, right) are possible with any attempts to go out of bounds resulting in staying in the same state.

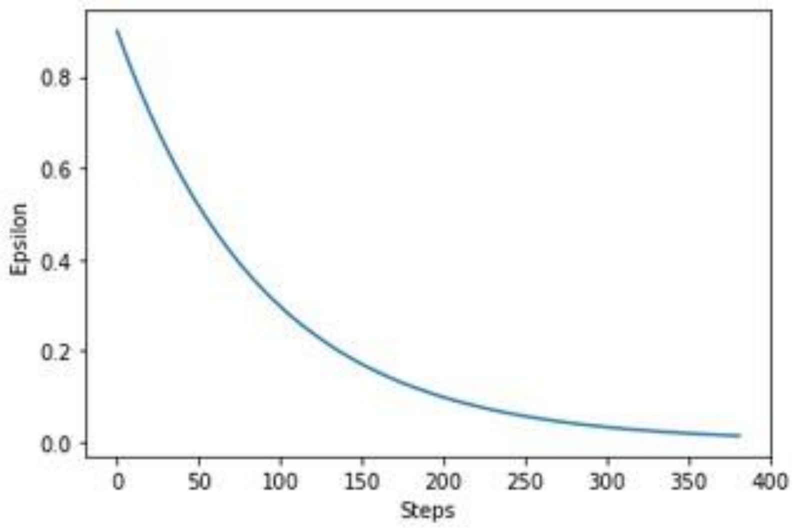
Unlike the environment in the previous assignment which has 3 reward states, the rewards are based on whether the transition bring the agent closer to the final state. +1 is it comes closer, -1 if it goes farther, 0 if it stays in the same place.

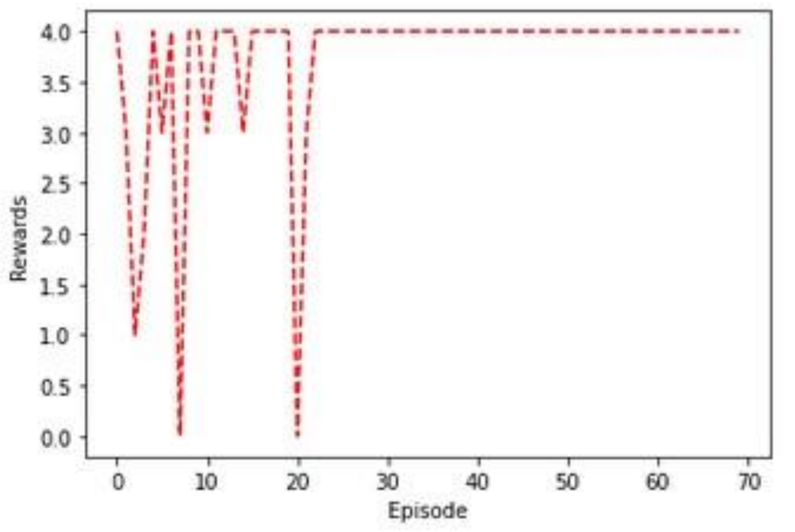
The agent implements DQN with a network having 2 hidden layers (70 and 30 nodes) and ReLu activation. The target network weights are synchronized every 4 episodes.



3. The agent is consistently able to finish the episodes with the most possible rewards after training for about 30 episodes in most random seeds. The result shown below is the case for a particular random seed where it manages

to get the most possible reward after about 20 episodes. Changing the max timesteps or the target net update frequency should result in different results. The epsilon follows an exponential pattern as shown below.





PART – 2

1. The improved algorithm being implemented is Double DQN. It is very similar to DQN with the only difference being that in DQN the Q-value is calculated using the reward and the maximum Q-value of the target network whereas in DDQN the Q-value is calculated using the reward and the Q-value of the main network for the best action from the target network.

2. The main improvement of Double DQN over the vanilla DQN is that the overestimation of Q-values is significantly reduced resulting in better training and results.

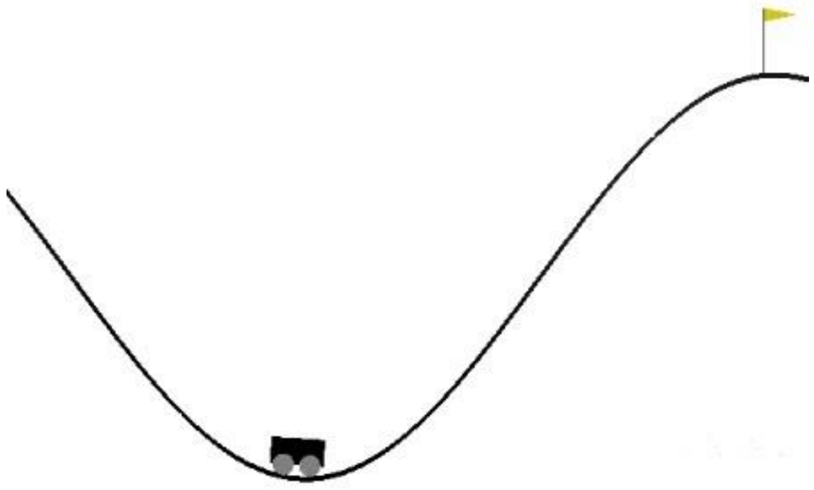
3. The two complex environments used are the OpenAI Gym environments MountainCar-v0 and LunarLander-v2.

MountainCar-v0:

The States are of the type Box(2) containing the car position ranging from -1.2 to 0.6 and car velocity ranging from -0.07 to 0.07. In the starting state the car has a random position between -0.6 and -0.4 and no velocity. The goal is to climb the mountain.

The agent has 3 possible actions which are push left, push right and no push.

The agent receives a reward of -1 for each timestep until the goal position of 0.5 is reached. The max timestep for each episode is 200. The environment is considered to be solved upon receiving a reward of -110 or better.



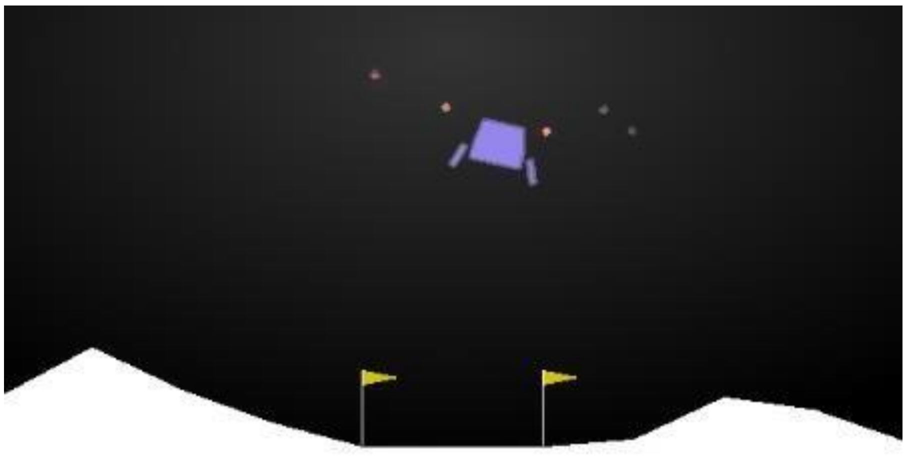
Visualization of the MountainCar-v0 environment

LunarLander-v2:

The Lunar Lander environment is relatively more complex with States of the type Box(8) where the first two numbers are the coordinates of the lander. The landing pad is located at the coordinates (0,0) and the goal is to land safely on it.

The agent has 4 possible actions which are firing main engine, firing left engine, firing right engine and not firing any engine.

The agent loses reward if the lander moves away from the landing pad. The episode finishes upon crashing or coming to rest receiving a reward of -100 and +100 respectively. Each leg in contact with the ground provides a reward of +10. Firing the main engine is -0.3 points and firing either of the side engines is -0.03 points. The environment is considered to be solved upon receiving a reward of +200.

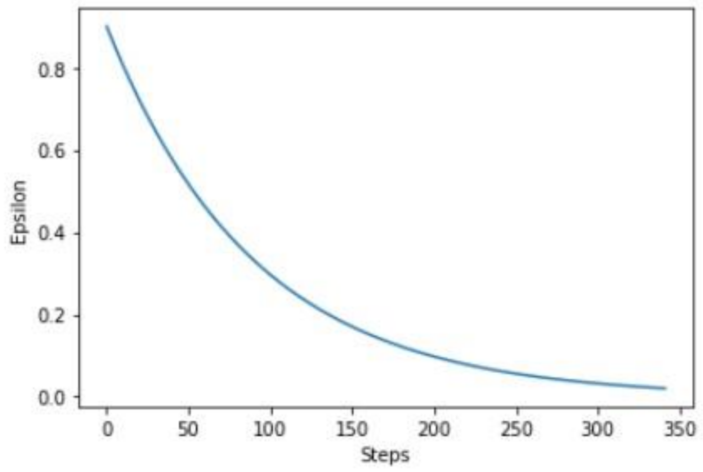


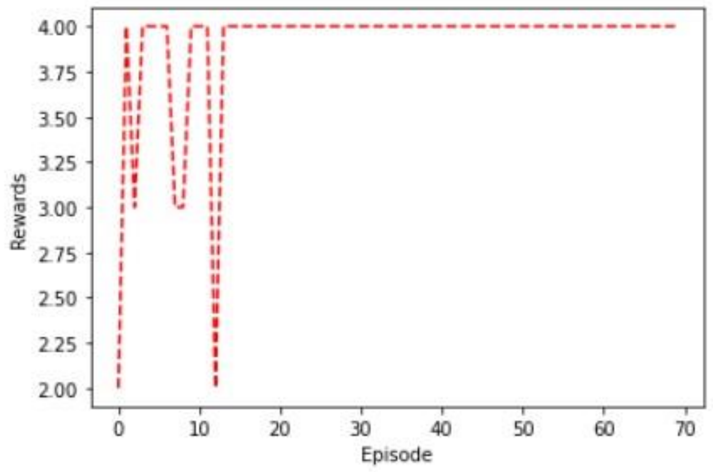
Visualization of the LunarLander-v2 environment

4.

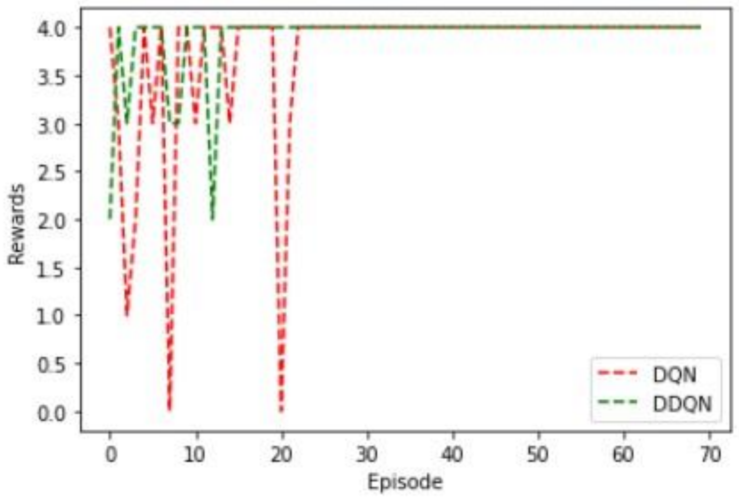
- Gridworld Environment –

The agent implementing Double DQN on the grid world environment is consistently able to finish the episodes with the most rewards possible after considerably lesser training than the agent implementing DQN in almost all cases (Only 11 episodes for this example). The epsilon decay and reward dynamics are shown below.





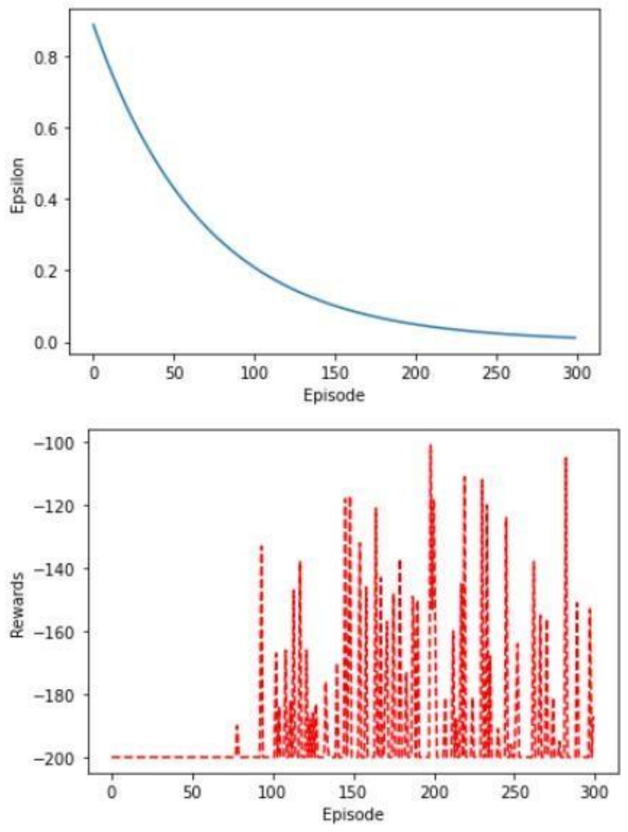
Comparing the rewards for the episodes of both the agents in the graph below gives a clear indication that Double DQN is able to train faster than DQN for this environment and also get more cumulative rewards over the 70 episodes.



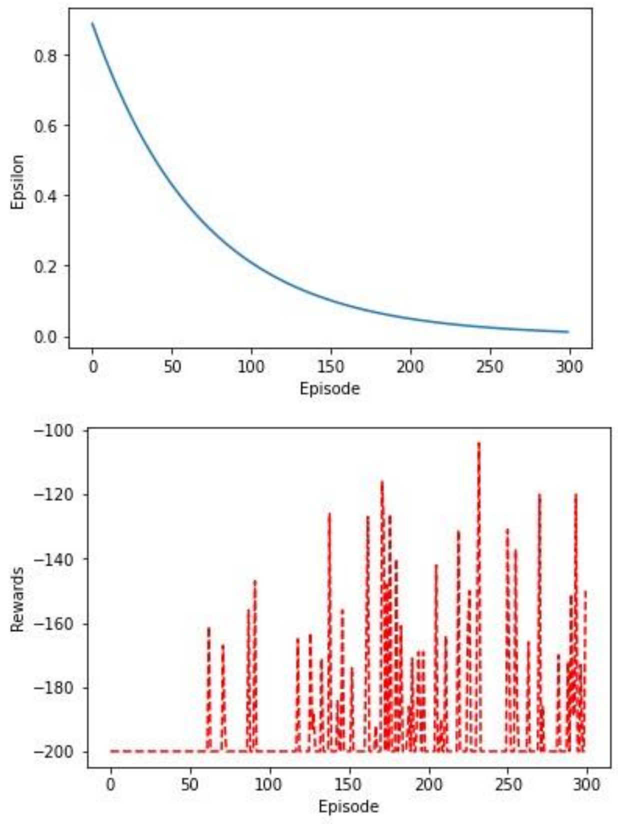
- Mountain Car Environment –

The agent implementing DQN uses a neural network with 2 hidden layers (22 and 64 nodes, ReLu activation) and is set to update in every 3 steps with the target network's weights being synchronized after every 5 episodes. The agent implementing Double DQN has the same parameters as the DQN agent and is also set to update the neural network every 3 steps and the target network’s weights are synchronized every 5 episodes so the comparison between the two algorithms can be more relevant.

The epsilon decay and total rewards for each episode are shown in the graph below.



DQN Agent

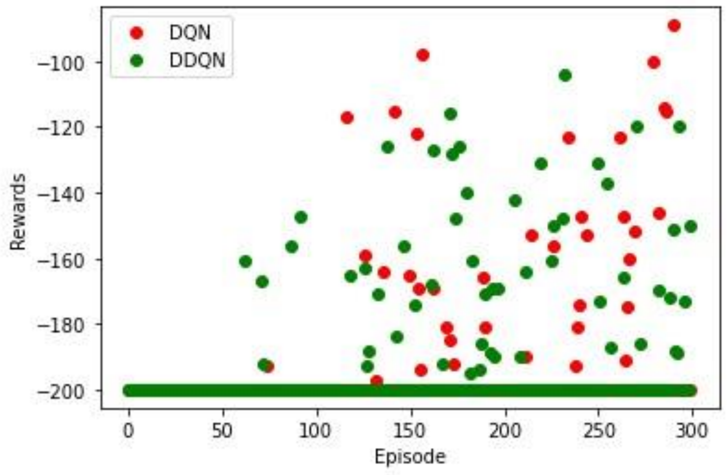


DDQN Agent

DQN Agent –
The first episode with a reward greater than the minimum is obtained around the 80th episode with significant improvements in rewards following it. From around the 150th episode, the agent is able to get a reward greater than -120 and manages to solve the environment first around the 200th episode by getting a reward greater than -110. This implies that the agent is able to climb the top of the hill in less than 110 timesteps. It is important to note that this particular environment provides very sparse rewards making the problem of solving this environment relatively hard.

DDQN Agent –

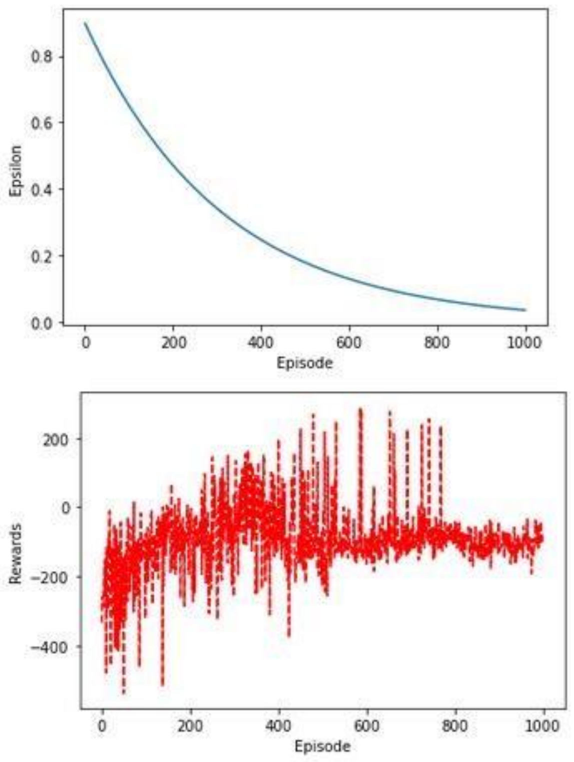
The first reward greater than the minimum reward possible is reached at around the 60th episode with the agent being able to gain a reward greater than -120 after the 130th episode. However, the agent takes slightly longer than the DQN agent to solve the environment for the first time at around the 240th episode.



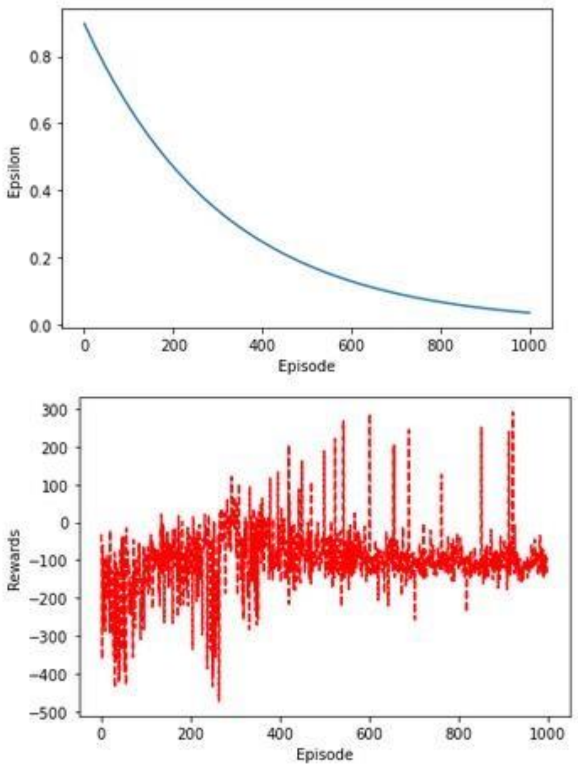
Looking at the rewards dynamics of both the agents in this particular example shows that DDQN is able to get the non-minimal rewards earlier than DQN but is not able to get the highest possible score. It is also not as consistent in solving the environment. This could be attributed to the inherent randomness in training a neural network and changing the parameters would improve DDQN's performance (especially the frequency of syncing the training net). It is important to note that even though DDQN is considered to be an improvement over DQN, the performance also depends significantly on the type of environment and its characteristics.

- Lunar Lander Environment –

Both the agents on the lunar lander environment use a neural network with 2 hidden layers (70 and 30 nodes, ReLu activation) and are set to update the neural network after every 4 steps with the training net weights being synced after every 4 episodes. The agents use the Adam optimizer and mse_loss as the loss function. The epsilon decay and reward dynamics are shown in the graphs below.



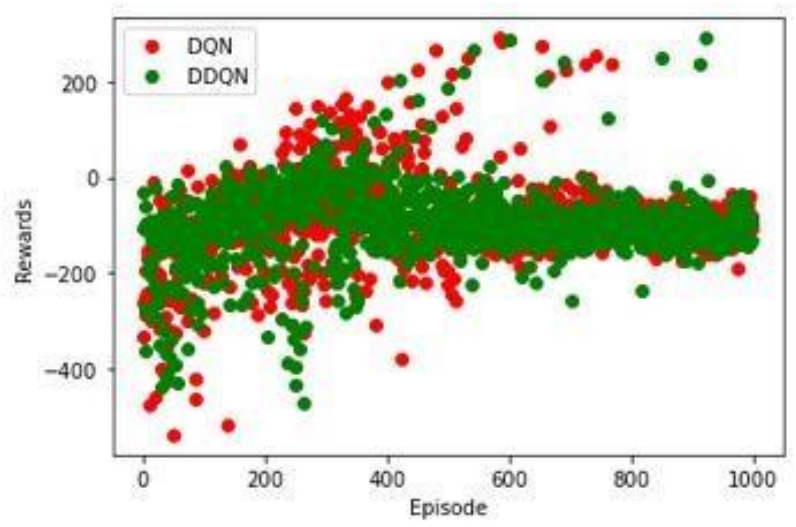
DQN Agent



DDQN Agent

DQN Agent -
During the training over 1000 episodes in this particular example, the agent is able to solve the environment numerous times with the earliest being around the 500th episode. The reward dynamics show a pattern of steady increase till the 400th episode after which the rewards become more inconsistent with the agent being unable to get a positive reward after the 800th episode just before which is the last episode where it manages to solve the environment by getting a reward of 200.

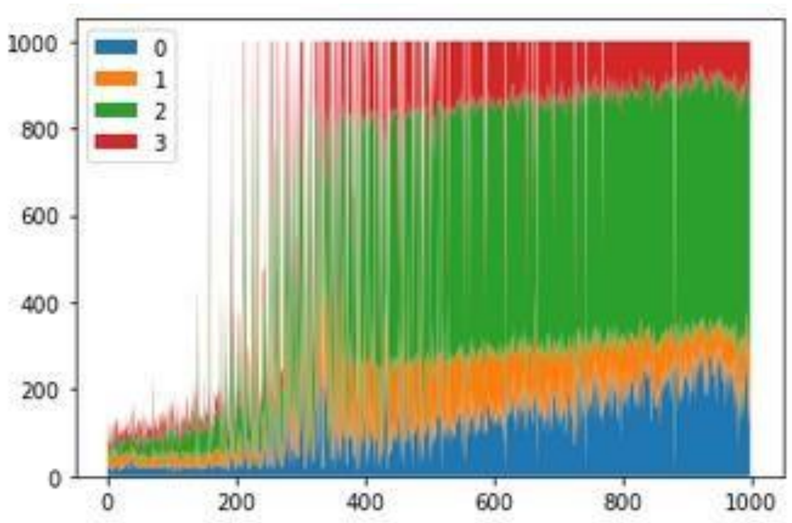
DDQN Agent –
As can be seen from the reward dynamics, the agent is able to solve the environment on numerous episodes but is quite inconsistent in the later episodes of the training, regularly getting a reward less than 0 similar to the DQN agent. Furthermore, it takes the agent around 450 episodes to solve the environment for the first time which is comparable to the DQN agent but manages to get better rewards following the 800th episode.



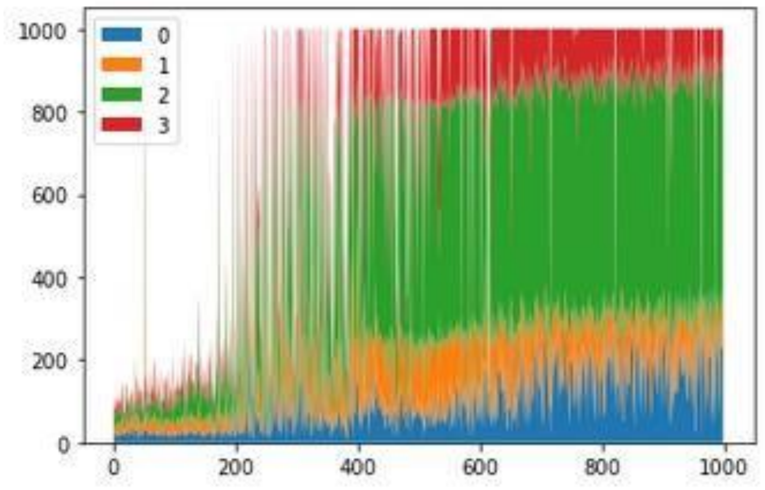
The above graph which give a direct comparison of the reward dynamics shows that the agents perform quite similarly in this particular example. The fact that both the agents are able to solve the environment on multiple occasions but become inconsistent in the later episodes indicates that with some fine tuning of the parameters, the agents could be trained to consistently solve the environment on almost every episode. It would be interesting to see the results of gradually changing the target network's weights to those of the main network over each timestep rather than synchronizing the weights after a certain number of episodes along with using a different optimizer and loss function.

5. One of the most fascinating realizations while assessing the results is the incredible versatility of DQN. The fact that the agents in both the grid world and lunar lander use the exact same neural network on vastly different environments and manage to solve them both is quite incredible. The two complex environments used – lunar lander and mountain car require policies that are completely unrelated to each other but the same algorithm is able to behave fundamentally differently in order to obtain the most possible rewards despite having different observations (box(2) and box(8)) and different number of possible actions (3 and 4)

However, the complex environments chosen and the parameters used do not show considerably different performance. More insight can be gathered by comparing the number of times an action is taken in one of the complex environments. Below, the Lunar Lander environment is considered (since it has a bigger max timesteps and significantly larger number of episodes) to see the similarities and differences between the two agents by using the stacked area plot.



DQN Agent – Lunar Lander



DDQN Agent – Lunar Lander

From the plots, it can be seen that the frequency at which both the agents perform the 4 different actions is quite similar. In the starting 200 episodes, the agents are not able to perform more than 200 actions since they crash quite early on but once they learn to survive for all 1000 timesteps, the similarity is evident. Although this particular plot shows only a partial picture of the true difference between the agents, there are some conclusion that can be drawn from it. Most obvious and important being that in this particular example, the DQN agent performs the 0 action a lot more frequently than the DDQN agent especially in the later episodes possibly explaining why DDQN agent is performing slightly better after the initial 800 episodes. The above plots are a good indication that the two algorithms learn differently (albeit slightly) from the same environment while using the exact same parameters.