CSE 546
Reinforcement Learning

Assignment 2

Team 16

Author: Sarveshwar Singhal (50418642), Sungjoon Park (50419347)

Submission Date: April 11, 2022

Part 1

1. The main details of ‘CartPole-v1’
   a. Objectives
      i. to achieve the average return greater than or equal to 195.0 over 100 consecutive trials by controlling the cart's velocity
      ii. while preventing a pole which is attached to a cart from falling over an episode whose length is greater than 200
   b. State Space (defined by 4 continuous elements)

	Position (x)	**Velocity** ($\partial x/\partial t$)	**Angle** (θ)	**Angular Velocity** ($\partial\theta/\partial t$)
Domain	$[-4.8,\ 4.8]$	$(-\infty,\ \infty)$	$[-0.418\,rad,\ 0.418\,rad]$	$(-\infty,\ \infty)$

   c. Action Space
      i. 0: Push a cart to the *left*.
      ii. 1: Push a cart to the *right*.
   d. Rewards
      i. 0: if the game is over or mission is completed
      ii. 1: if the game is not over
   e. The termination conditions (game over conditions)
      i. if the pole's angle is more than 12 degrees or
      ii. a cart's position is more than 2.4 (center of the cart reaches the edge of the display)

2. The main details of ‘FrozenLake-v1’
   a. Objectives
      i. to get to the goal (G) in the bottom right corner where a frisbee is
      ii. while avoiding the holes (H) on the way to the goal
   b. State Space

S	**F**	**F**	**F**
F	**H**	**F**	**H**
F	**F**	**F**	**H**

H	F	F	G

i. S: Starting point (safe)
ii. F: Frozen surface (safe)
iii. H: Hole, the game will be over once you step in
iv. G: Goal, where the frisbee is located

c. Action Space
   i. 0: Go *left*.
   ii. 1: Go *down*.
   iii. 2: Go *right*.
   iv. 3: Go *up*.

d. Rewards
   i. 1: if you reach the goal (G)
   ii. 0: otherwise

e. The termination conditions (game over conditions)
   - The game is over if your agent reaches H states.

Part 2

1. Discuss the benefits of the following.
   a. Using experience replay in DQN and how its size can influence the results:
   Sampling data from a replay buffer in order to learn is called an experience relay.
   Basic use of experience replay is to store past experience and sample a random subset of experience to update Q-network, rather than picking the latest experience.
   Larger experience replay provides more stable training of neural network but it takes more memory and mostly it's slow as well.
   It depends on our priority, if stability is our priority then use larger experience replay network, else if training time is our priority use small experience replay.

   b. Introducing the target network:
   Target network is used to make neural network training stable. It is used to avoid chasing problems. We create two networks, one main network and the other is the target network.
   The reason why this measure is necessary is to have more accurate loss function values. For every iteration in an algorithm, the estimates for a model's parameters continue to change. As such, if we get $w1 = argmax\hat{q}(s = 1, w)$

and $a = \hat{q}(s = 1, w = w_1)$ in the nth iteration, as soon as we try to do the next iteration for $s = 1$ after a few iterations, the target value for loss function computation becomes $b = \hat{q}(s = 1, w = w_k)$ where k is not equal to 1. (Assume that it is the first iteration that $s = 1$ after the nth iteration.) The loss at this time is supposed to be $a - \hat{q}(s = 1, w = w_k)$ but it is $b - \hat{q}(s = 1, w = w_k)$. This is because the estimates for parameters continue to be updated.

To fix this problem, if we fix the estimates for the target network's parameters for a while, we might be able to obtain more accurate loss function values. Specifically, we train our neural network on the main network and synchronize its parameters periodically with those in the target network.

c. Representing the Q function as $\hat{q}(s, w)$

If we were to use Q-learning algorithm, we need to create a Q-table and update every cell in the table throughout the whole given data. It might be possible to implement this approach for an environment with a moderate number of states and actions. However, if the number of states or actions increases drastically, we can hardly implement the algorithm or it will take too much time even if it is feasible to complete the implementation.

Deep Q-learning for the Q function approximation ($\hat{q}(s, w)$) help us reduce the time required. Although it sacrifices the precision of estimating Q values, it practically enables us to train our models within a limited time.

2. Briefly describe the grid-world environment that you used (e.g. possible actions, states, agent, goal, rewards, etc). You can reuse related parts from your Assignment 1 report.
    a. main features
        i. The environment is deterministic. That is, $p(s', r|s, a) = 1$ or 0 for $\forall a \in Action\ Space,\ \forall s,\ s' \in Observation\ Space,$ $and\ \forall r \in rewards\ set.$
        ii. set of actions: $\{up, down, left, right\}$
        iii. the number of states: 16 (4 by 4 Grid World)
        iv. rewards: $\{0, 0.2, 0.3, 0.7, 5\}$
        v. main objective: to arrive at the terminal state within 20 steps
        vi. visualization

0	1	2	3

(starting)			(0.7)
4	5	6 (0.2)	7
8	9	10	11
12 (0.3)	13	14	15 (terminal, 5.0)

b. What has changed from the Assignment 1?

   i. rewards: $\{0, \ -0.2,\ 0.3, -\ 0.7,\ 1\} \rightarrow \{0,\ 0.2,\ 0.3,\ 0.7,\ 5\}$

   ii. time limit: 15 → 20

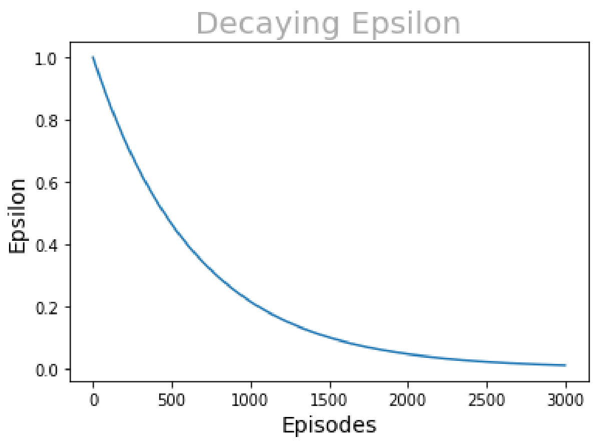
c. Safety in AI

   i. When an agent is forced to leave the edge of the grid, it is forced to maintain the edge of the grid so that the simulation can only be repeated within a given environment.

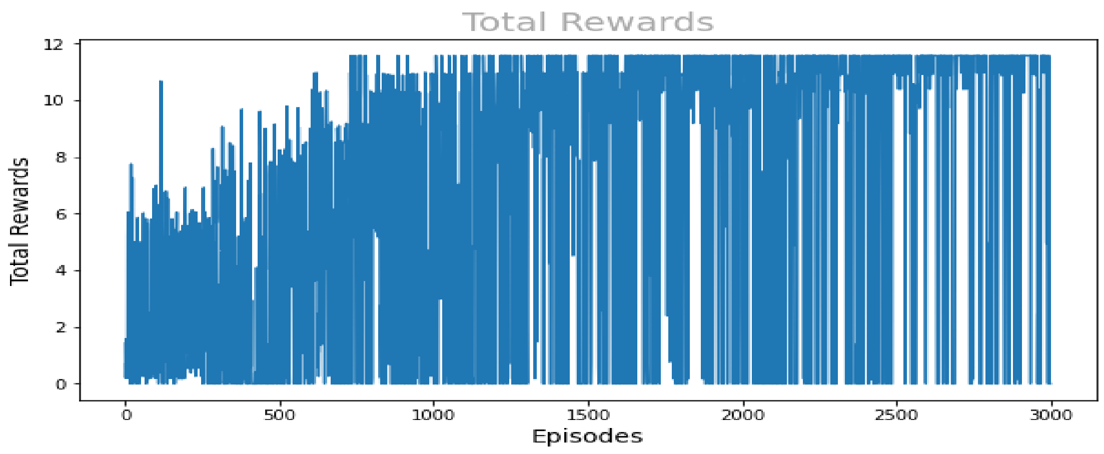
   ii. It is implemented through NumPy's clip function.

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

   a. Plot: epsilon decay

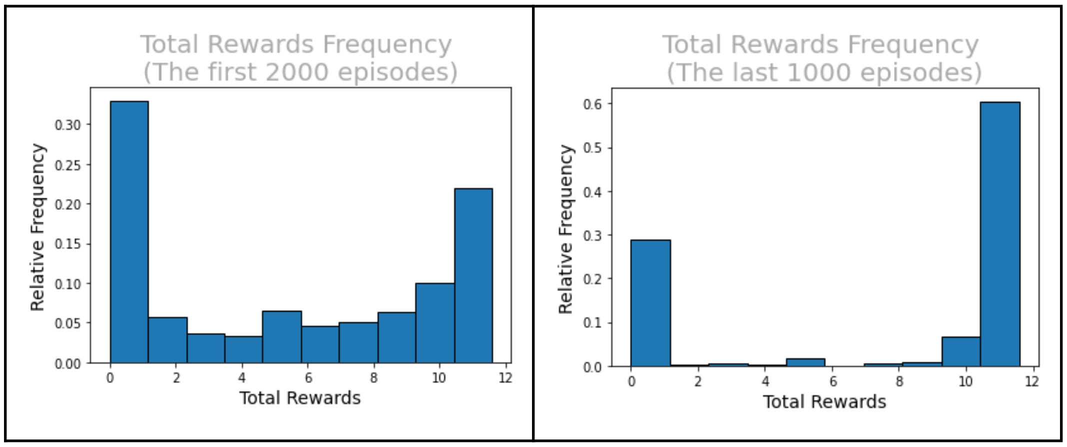


   b. Plot: total reward per episode



The performance seems to improve as episodes progress, but it is uncomfortable to see because the line densely fluctuates in the figure. Thus, we provide two relative frequency histograms to make it visually easier to determine whether our vanilla DQN improves the performance in the following.

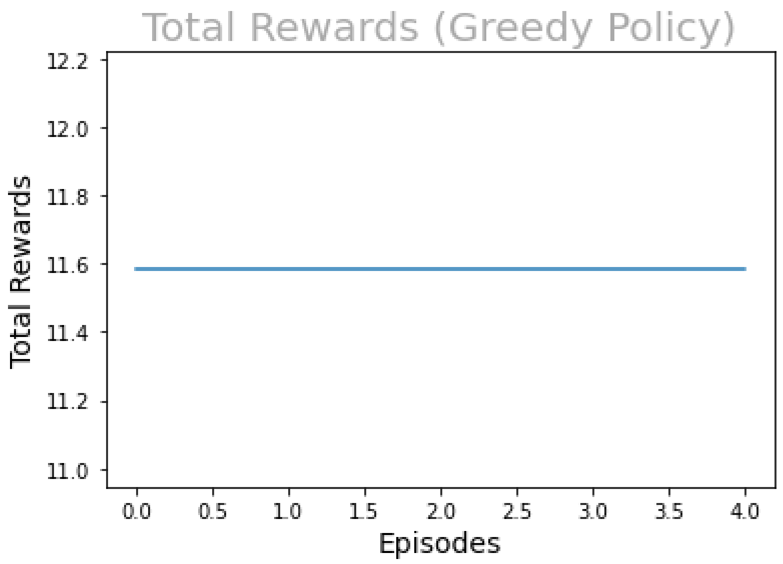
c. Plots: Improved Performance through the vanilla DQN training



At the beginning of the learning, the proportion of achieving a total reward more than 10 is about 20%, but after the 2000th episode, when the epsilon value becomes less than 0.05, the total reward is greater than 10 for about 60% episodes.

However, in episodes, which are still close to about 30%, the total reward value is close to zero despite of the vanilla DQN training and decaying epsilon. It appears that it is due to the existence of epsilon, which enables an agent to choose non-optimal actions.

4. Provide the evaluation results. Run your environment for at least 5 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.



When adopting the greedy policy, it always shows the greatest total reward that can be achieved with the epsilon-greedy policy. From this, it can be seen that the reason why the episodes, which still showed low performance even close to the last episode in the epsilon-greedy policy, existed was due to the presence of epsilon, not zero.

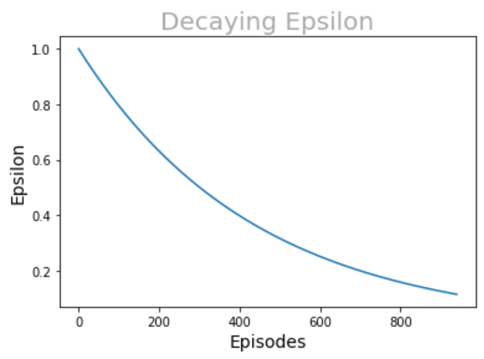
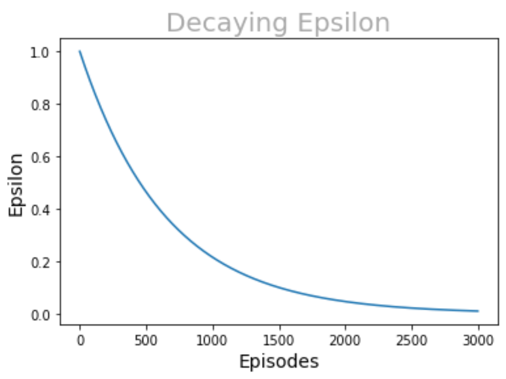
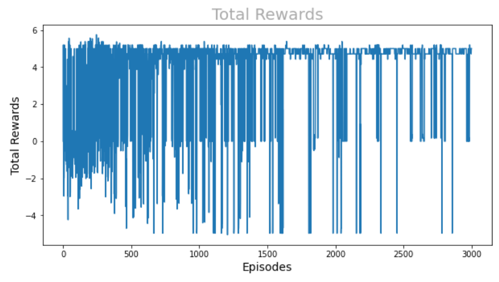
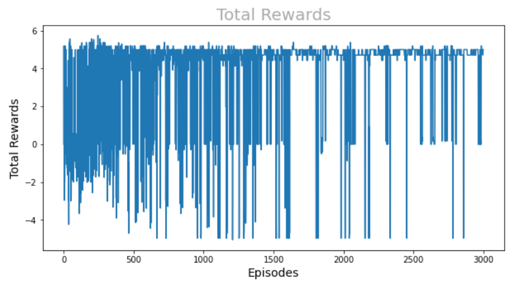
Part 3

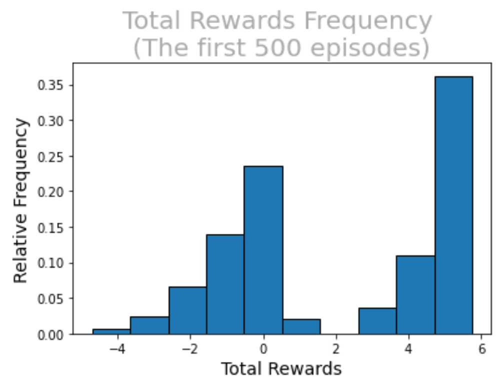
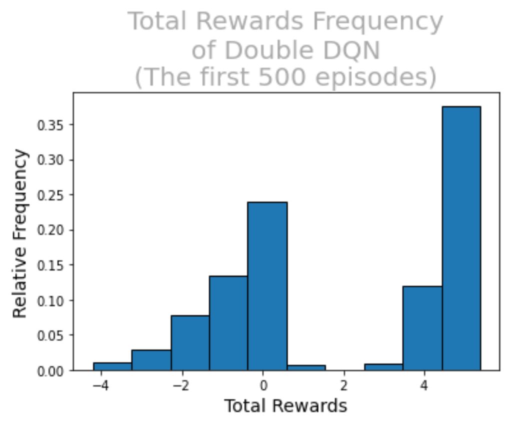
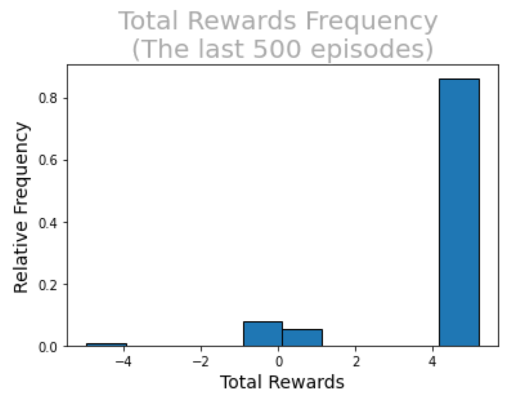
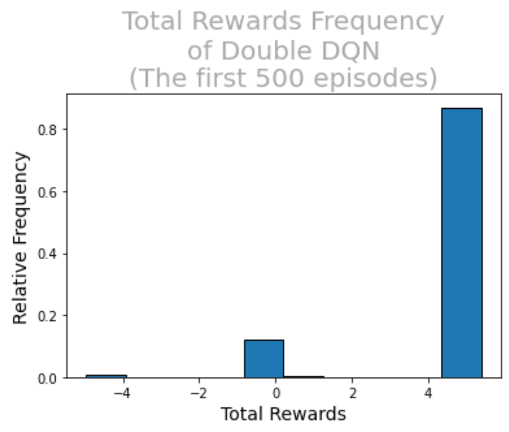
1. Discuss the algorithm you implemented.
    a. (Vanilla) DQN
    It approximates Q function (action value function) applying deep neural networks. By doing so, it can be used for the cases in which the number of states is too large to implement Q-table. The main idea is to use Q-learning algorithm and utilize neural networks to obtain estimates for $Q(s, a)$. Meanwhile, it outperforms DQN algorithms by adopting the concept of experience replay and introducing a target network. However, it still has some drawbacks, one of which is maximization bias. This problem aggravates the performance of vanilla DQN algorithms.
    b. Double DQN
    Double DQN appeared to solve maximization bias which is found in vanilla DQN algorithms. The main idea of this algorithm is the same is that of Double Q-learning. It creates two individual neural networks as Double Q-learning updates two individual Q-tables. So it computes Q-target by obtaining the action which maximizes Q-network outputs and the

corresponding state and substituting them into another Q-network's to get the Q-target.

2. What is the main improvement over the vanilla DQN?
   Double DQN successfully alleviates maximization bias from which vanilla DQN algorithms suffer. In the following examples, we show that Double DQN algorithms outperform vanilla DQN algorithms in the same environments.

3. Show and discuss your results after applying your two algorithms implementation on the environment. Plots should include epsilon decay and the reward per episode.
   a. GridWorld Environment (3.1)

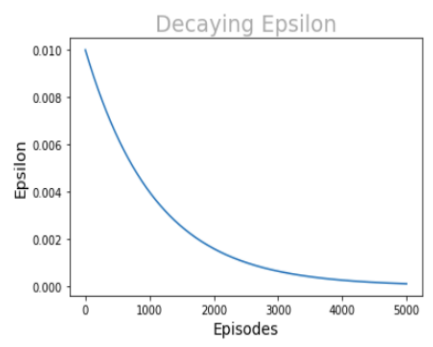
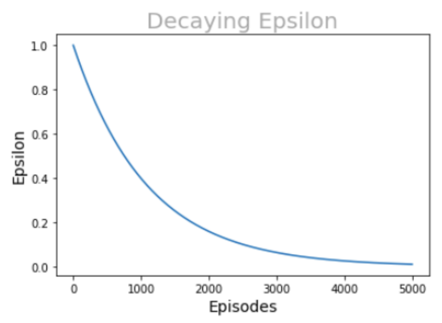
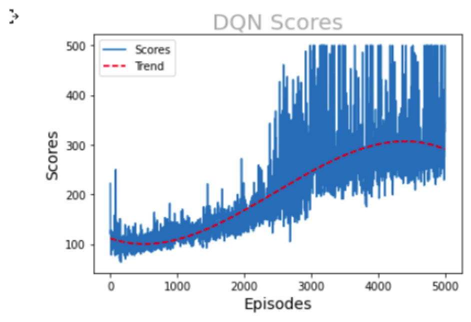
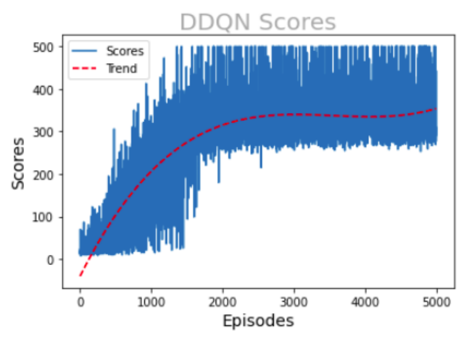
Epsilon Decay	
	
DQN	Double DQN
Reward per Episode	
	
DQN	Double DQN
Total Rewards in the first 500 episodes	

	
DQN	Double DQN
Total Rewards in the last 500 episodes	
	
DQN	Double DQN

It does not appear that there are significant differences between the performance of vanilla DQN algorithm and that of Double DQN algorithm. The latter only shows slightly higher performance. The reason seems to be due to the simplicity of GridWorld environment. The environment is deterministic and consists of 4 by 4 states.

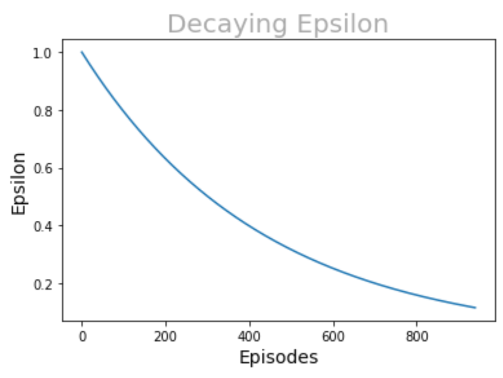
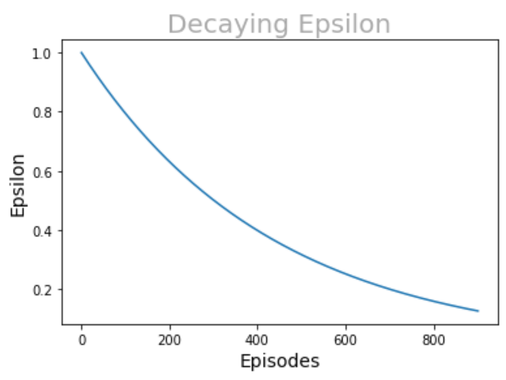
b. CartPole-v1 (3.2)

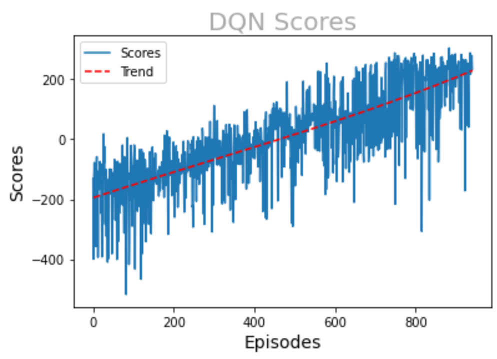
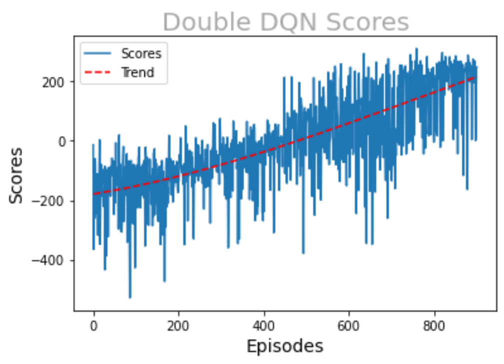
Epsilon Decay

	
DQN	Double DQN
Reward per Episode	
	
DQN	Double DQN

i.

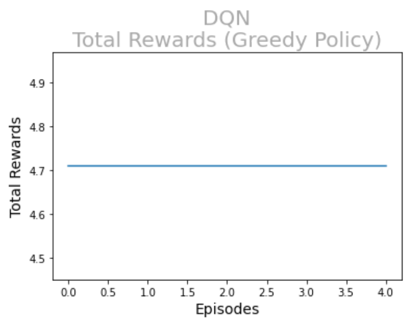
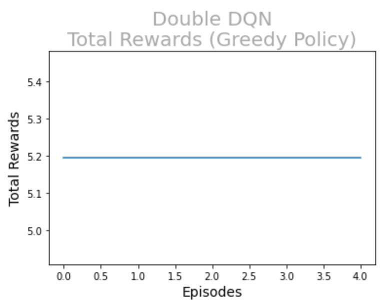
c. LunarLander-v2 (3.2)

Epsilon Decay	
	
DQN	Double DQN
Reward per Episode	

	
DQN	Double DQN

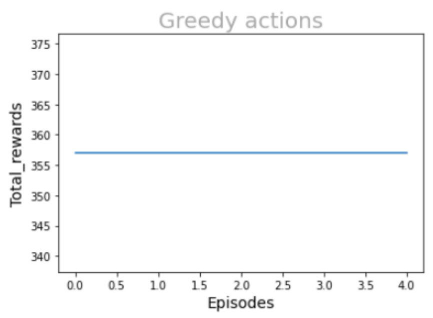
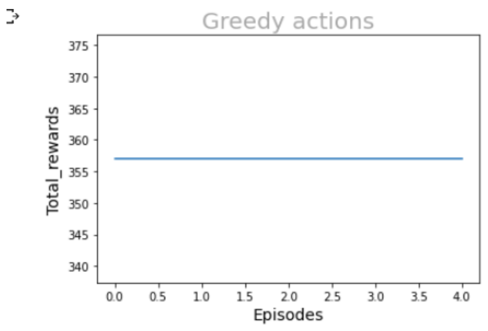
In LunarLander-v2 environment, the DQN algorithm completed training for about 665 seconds during 941 episodes. On the other hand, in the same environment, the Double DQN algorithm made it for about 578 seconds during 902 episodes. That is, the latter was faster than the former by over 10%. Note that the objective of this environment is to obtain more than 200 scores in an episode. We revised the objective a little bit and considered the average score of the latest 100 which is over 200 as successful. It appears that the reason why Double DQN clearly outperforms vanilla DQN in LunarLander-v2 whereas it does not in a simple GridWorld environment is LunarLander-v2 is stochastic and much more complicated than our GridWorld.

4. Provide the evaluation results. Run your environment for at least 5 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.
    a. GridWorld Environment (3.1)

	
DQN	Double DQN

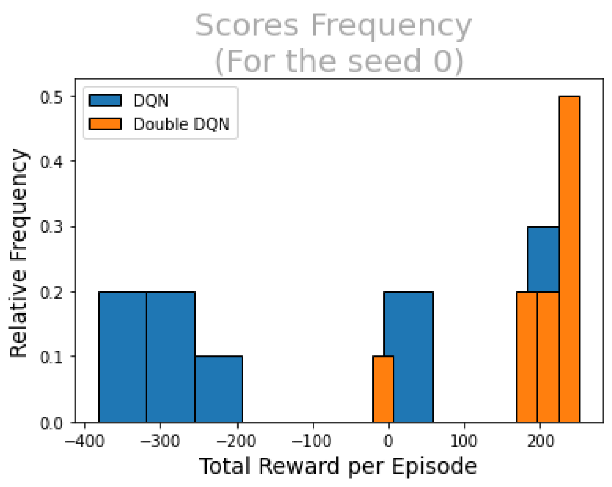
Both algorithms show stable performance when it comes to greedy actions from the learnt policy. However, the Double DQN obtains larger total rewards than the vanilla DQN.

b. CartPole-v1 (3.2)

	
DQN	Double DQN

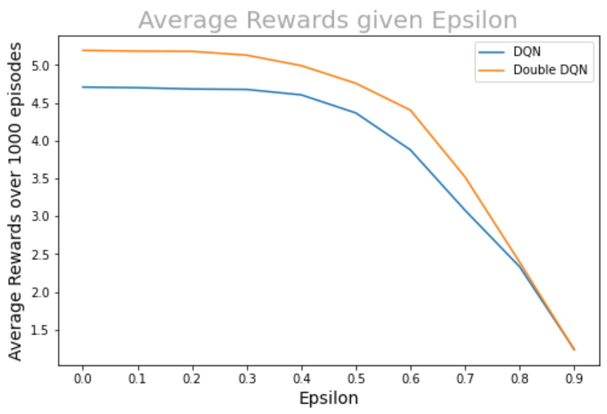
i.

c. LunarLander-v2 (3.2)



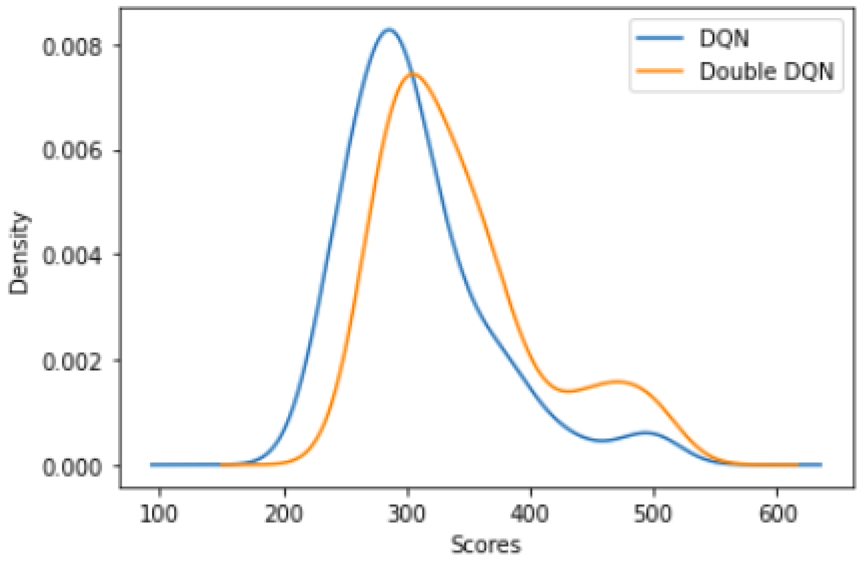
This is a fairly interesting result because our vanilla DQN and Double DQN succeeded in completing the mission. However, since the environment is stochastic, DQN algorithm excessively reacts to noisy trajectories, which consequently results in worse performance than Double DQN. Our DQN did show some successful cases just like in the right side of the figure (a blue bar in the right side) but it failed several times presumably in some noisy episodes. On the other hand, Double DQN showed stable and better performance. The difference is mainly due to the existence and size of maximization bias.

5. Compare the performance of both algorithms (DQN & Improved version of DQN) on the same environments (e.g. show one graph with two reward dynamics) and provide your interpretation of the results. Overall three rewards dynamics plots with results from two algorithms applied on:
   a. GridWorld Environment (3.1)



   To see reward dynamics, we tracked and stored the rewards along the various epsilon values. We found that there exist consistent performance differences between two algorithms. That is, the Double DQN algorithm strongly dominates the DQN algorithm, even if the environment's setting changes.

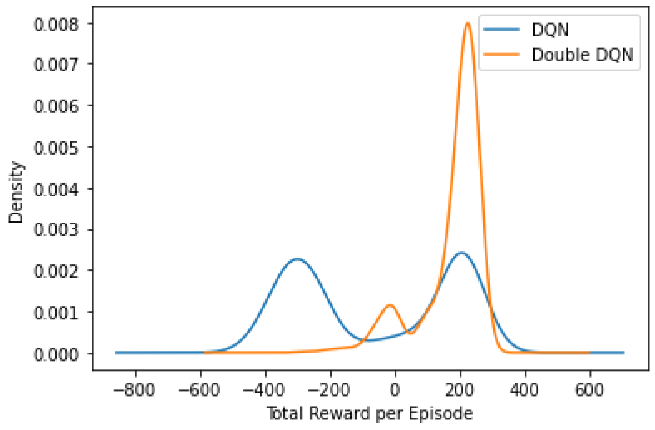
   b. CartPole-v1



      i. We can see that the double DQN total reward per episode is more as compared to DQN. The majority of scores of DQN lies between

200-300 whereas Double DQN is between 300-400. On further increasing the no. of episodes and tuning hyperparameters we can achieve stable results.

c. LunarLander-v2



We have repeated simulation for 1000 random seeds which have not been applied to the training process. The above figure shows the results of 1000 trials. The DQN algorithm sometimes succeeds in completing the mission but it shows bi-modal distribution, which means it receives scores less than 0 frequently. So if there exists some randomness which the algorithm has not experienced, it excessively reacts to those noisy patterns so that it fails too many times. On the other hand, the Double DQN which is designed to reduce maximization bias show stable performance around the score of 200. It indicates that the Double DQN is less sensitive to noisy patterns than the DQN algorithm.

6. Provide your interpretation of the results. E.g. how the same algorithm behaves on different environments, or how various algorithms behave on the same environment.

   Based on our understanding so far we felt that on same environment Double DQN performs betters in terms of convergence and giving better rewards as compared to DQN. Moreover we felt that, with the same algorithm which worked in one environment doesn’t necessarily work in another environment with that much efficiency.

Contribution:

Team Member	Assignment Part	Contribution
Sungjoon Park	Part 1, 2, 3	50 %
Sarveshwar Singhal	Part 1, 2, 3	50 %

We coordinated amongst ourselves and choose the best working solution.