Arshabh Semwal
50419031
arshabhs@buffalo.edu

Naman Tejaswi
50435697
namantej@buffalo.edu

Assignment 2

"We certify that the code and data in this assignment were generated independently, using only the tools and resources defined in the course and that we did not receive any external help, coaching or contributions during the production of this work."

Part 1

1. Environment 1 'Cart Pole - v1' :
   It is a standard environment present in the gym library along with many others which can be directly accessed using the make() function of the gym library

   Environments features:
   - Actions: This environment has two actions left and right (represented by 0, 1)
   - Observation Spaces/States: The state contains all the positions in 2.4 units from the center (in both left and right direction) and in each position in a straight line the pole can be between +12 to -12 degrees from the normal. Also, cart velocity and pole angular momentum is observed.
   - Reward: +1 for each timestep the pole doesn't fall
   - Goal: To keep the pole from falling from the cart as long as possible.

2. Environment 2 'Mountain Car - v0':
   Just as the above Cart Pole environment it is also a standard environment present in the gym library and can be accessed in the same manner.

   Environments features:
   - Actions: This environment has three actions accelerate to left, don't accelerate, and accelerate to right (represented by 0, 1, 2)
   - Observation Spaces/States: position of the car along the x-axis and velocity of car.
   - Reward: -1 for each timestep the car doesn't reach the goal, and 0 when it reaches the goal.
   - Goal: To maneuver the car to reach the goal.

Part 2

1. Benefits of :

1. **Experience Replay:**

   Using experience relay helps the model to actually learn as it relays the information about the optimum steps from the past experience to the next episode, if we do not use experience relay there would be no learning that is it would be the same as if we had one single episode. Replay is required in order to transfer our experiences and learnings from one particular episode and improve our decision making and consequently rewards.

   Impact of Experience Replay Size

   Having too large or too small of an experience replay buffer will deteriorate the performance because if the experience replay buffer is too large the model will not learn with the latest experience. The learning from earlier experiences will contribute first as sequentially they will be first in the buffer and this might not be favorable.
   Also the training time increases as we increase the experience replay buffer size.

   On the other hand if replay buffer is too small the model might not learn anything of significance
   To mitigate this we can use combined experience replay where we use both the latest experience and the previous buffer. We can also consider using prioritized experience replay.

   Target Network:

   Input: Number of State Action pair of dimension 2
   Fully Connected Layer 1: Linear Activation Function
   Fully Connected Layer 2: Linear Activation Function
   Output Layer: Action dimension 4

   Q Function

   Q(s,a)t+1 = Q(s,a) + ɑ[r(s,a) + ɣmax Q\`(s\`,a\`) - Q(s,a)]

   Q(s, a, w) ≈ Q π (s, a)

   DQN essentially approximates the Bellman Equation

$$Q(s,a) = E_{(s,a \to s',r)\sim\mathcal{H}}\left(r_{s,a} + \gamma \max_{a'} Q(s',a')\right)$$

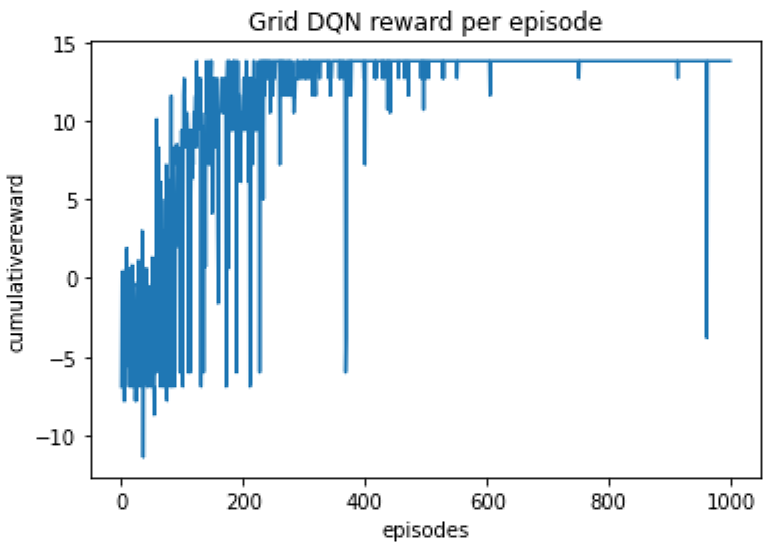
2. Specification of Environment:

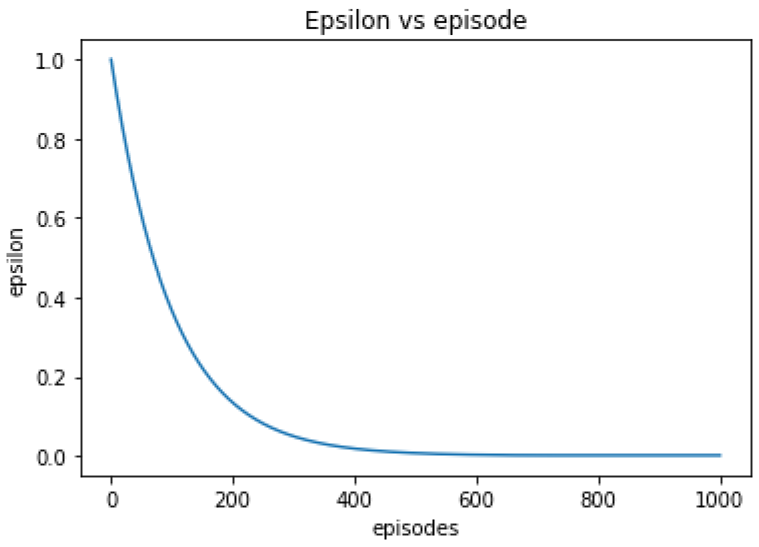
- Total states = 16 (a 4x4 grid)
- Total rewards = 4 (reward 2 corresponding to pos(3,3), reward 1 corresponding to pos(0,3), reward -1 corresponding to pos(1,1) and, reward 0 for all the other states)
- Number of actions: 4 (up, down, left, right)
- Agent start position is (0,0) and goal position is (3,3)

3. Results

The DQN is applied at the gridWorld

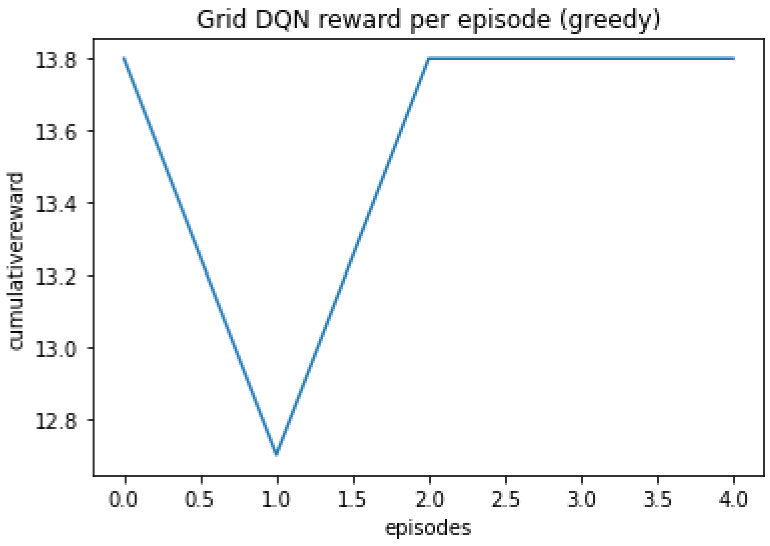
The DQN converges to an reward of approximately 15 at about 200 episodes, the convergence is faster as expected with a Deep Q learning network as opposed to simple Q learning from assignment 1 although the reward is similar, this as gridworld is a relatively simple environment and using double q learning is also effective as it converged to the same average reward. DQN improves it slightly but the main benefit is that the convergence is faster for the same learning rate and this is attributed to the deep neural network used to calculate the q values.

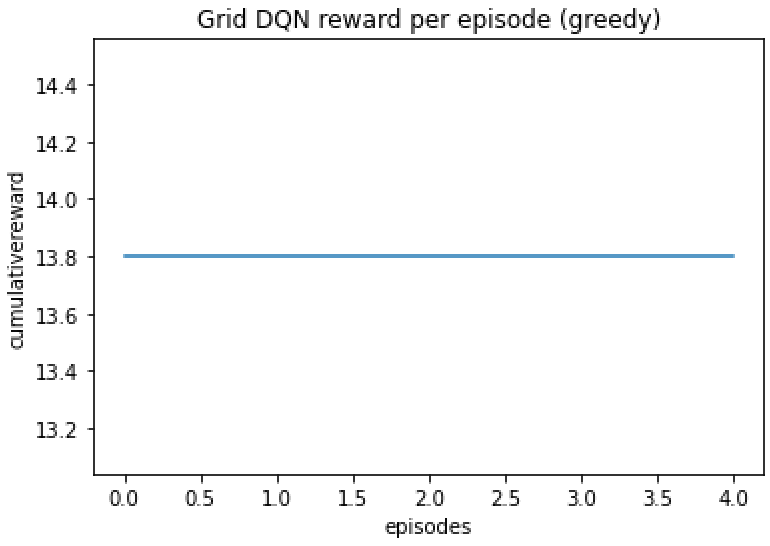




4. Evaluation

We run the model for 5 episodes where the agent simply chooses the greedy action that is the action with highest reward and in this case the reward converges to 13.8.



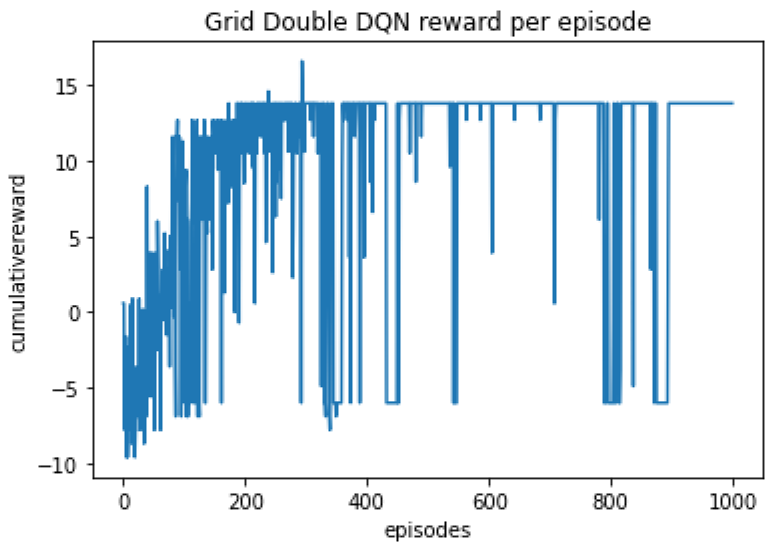


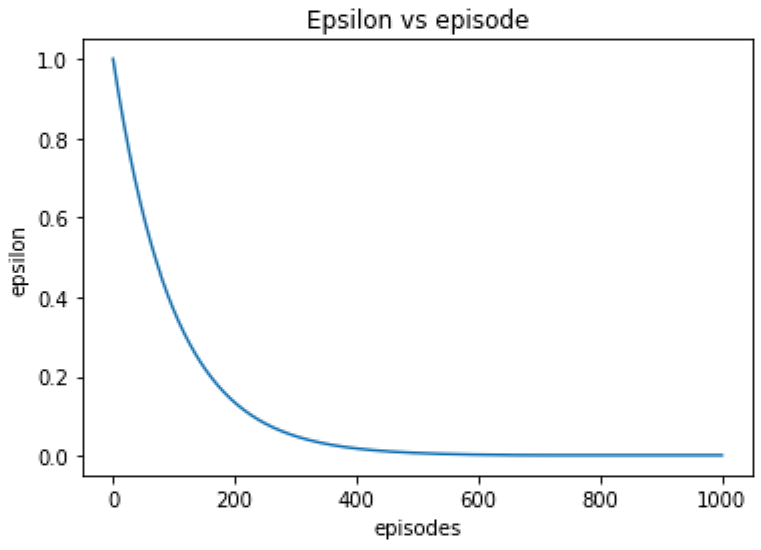
Part 3

Improving vanilla version of DQN and applying it to to grid world, cartpole and Lunar lander environments

Applying Double DQN on Grid Environment

Using Double DQN on our grid environment we observe that convergence is even faster and again it converges to the same value. Average reward per episode is similar after convergence.





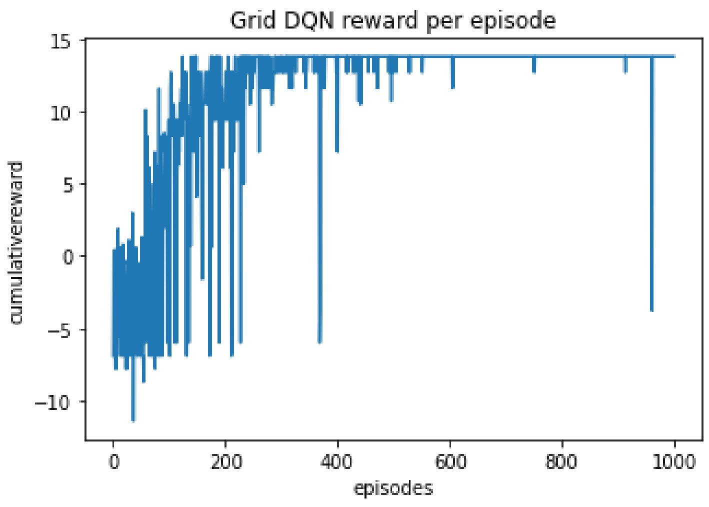
.

1. We have implemented double DQN where we have used 2 different deep neural networks, similar to when we used double q learning after q learning. We use the first DQN network to select or determine what is the best action to take for the next state and we use the second DQN target network to calculate the target Q value taking the action of the next state.
   We have applied Double DQN on the grid world environment, on the cartpole open ai environment and on Lunar Lander open ai environment.

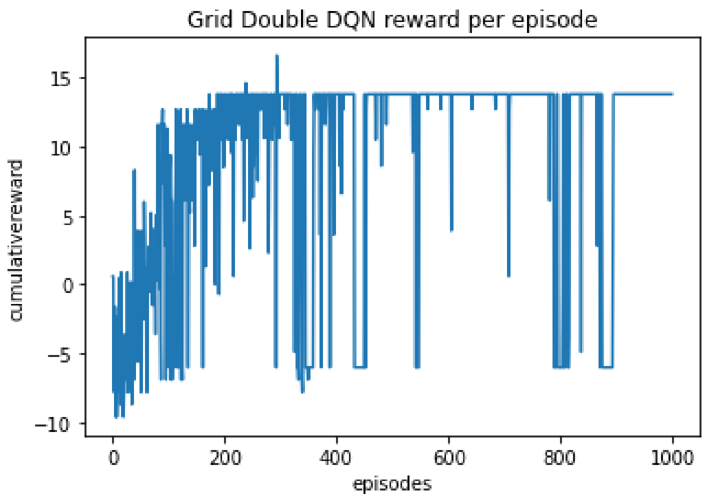
2. The main improvement from vanilla DQN is that we are using an additional target deep neural network which increases the rate of convergence similar to using double q learning over q learning. In DDQN one network is used to select the best action for the next state and another target network is used to calculate the target Q value of taking that action at the next state.

3. Comparison of results

   Results from Vanilla DQN

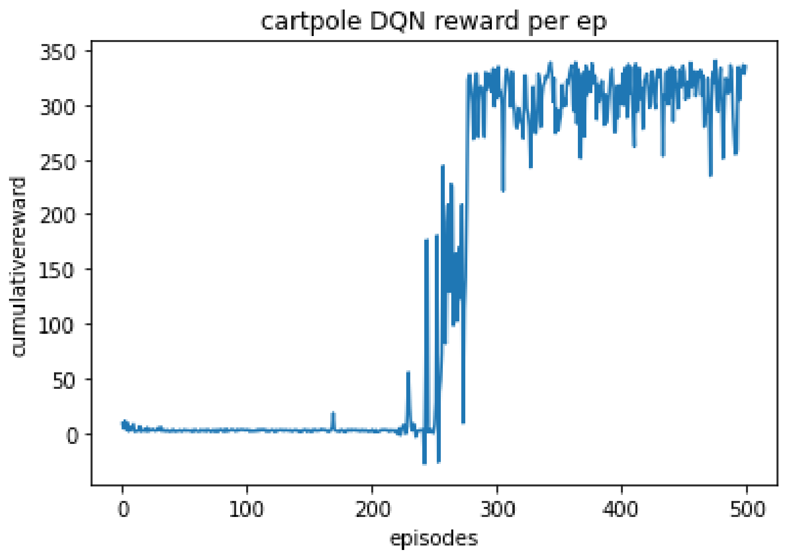


   Results from Double DQN

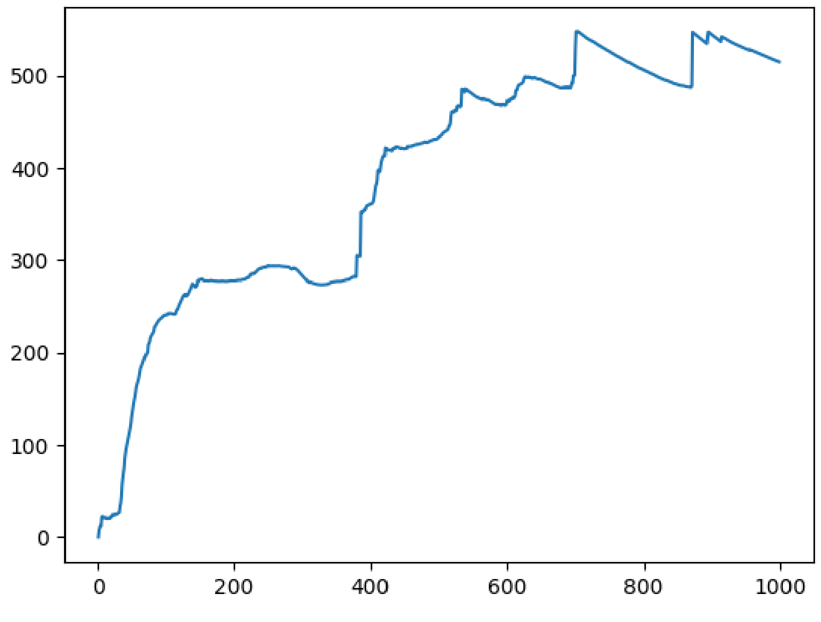


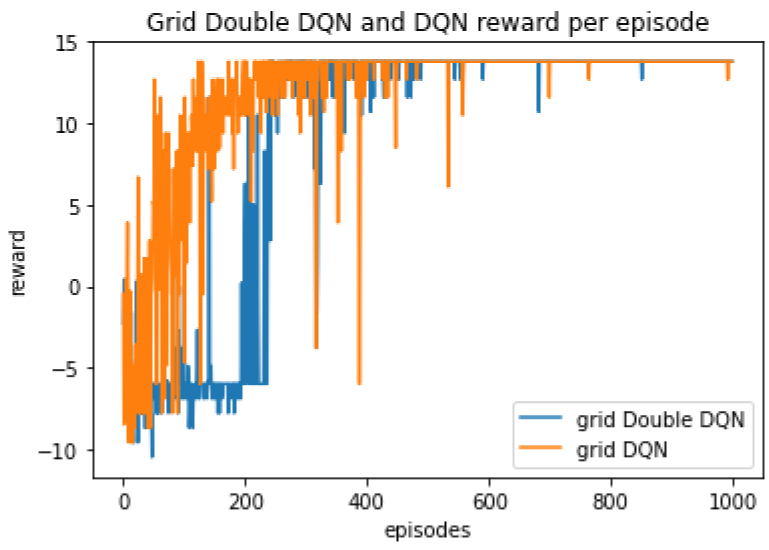
For cartpole Environment

Vanilla DQN



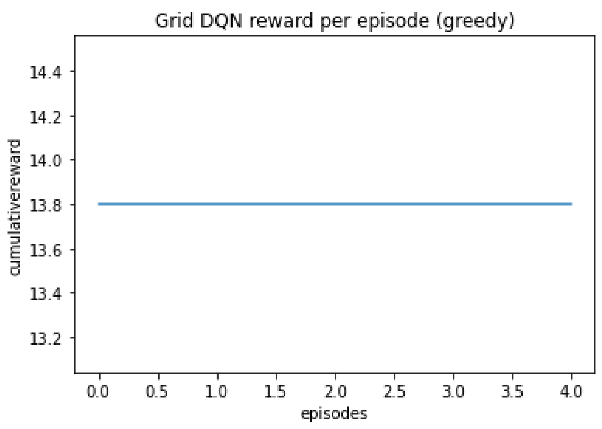
Double DQN

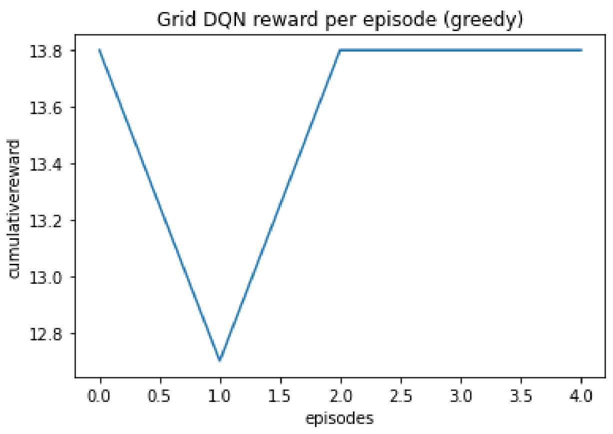




We see that the average reward converges to 500 with double DQN whereas it converges to only 350 with vanilla DQN. Also once again the rate of convergence is much higher than vanilla DQN due to using 2 different neural networks.

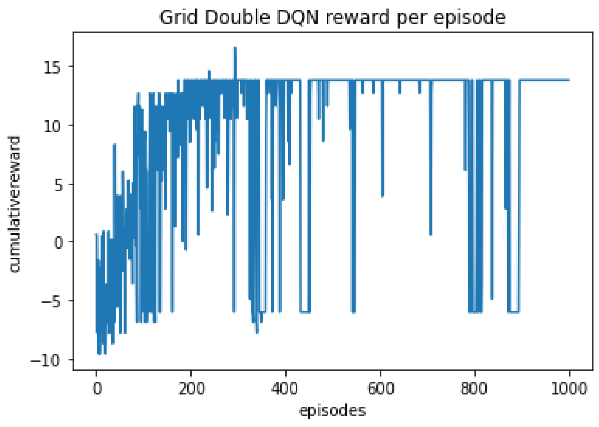
4..

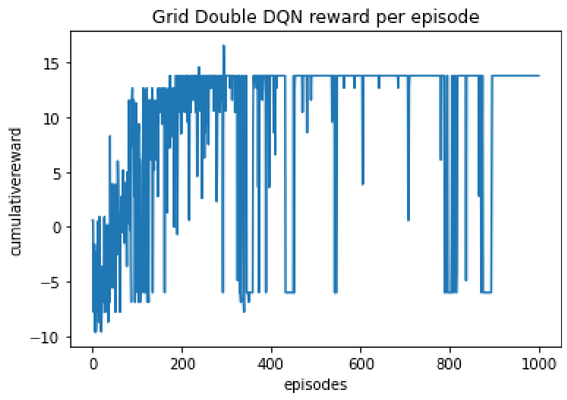


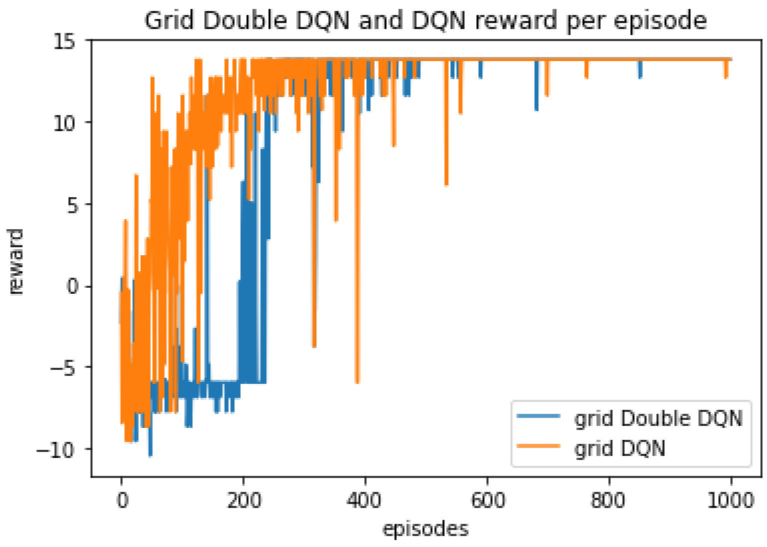


5. Comparing performance of both algorithms

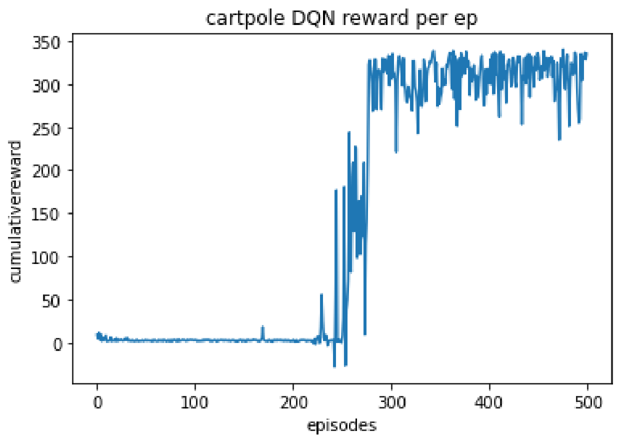
Grid World

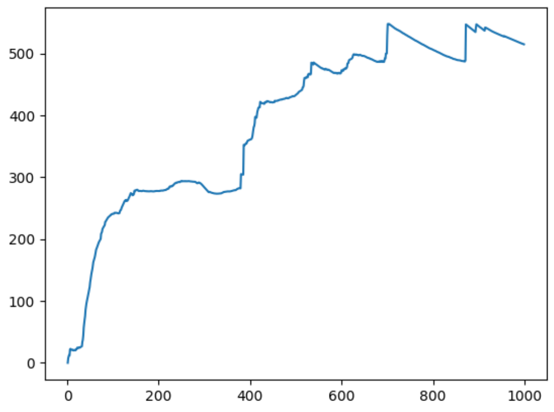






Cartpole

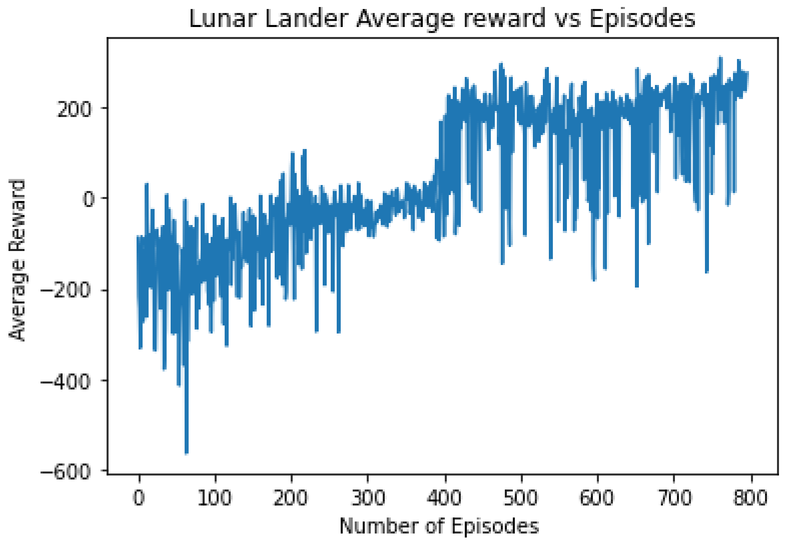




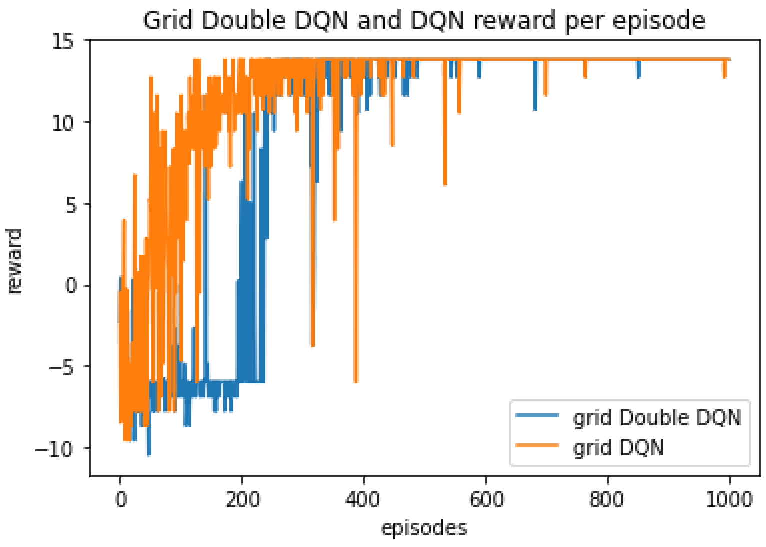
Vanilla DQN

DDQN

Lunar Lander



6. Interpretation of Results.



For simpler environments such as grid world vanilla DQN and double DQN both converged to the same value with similar speed however for complex environments such as cartpole vanilla DQN could only get to about 350 reward whereas double DQN converged to an average reward of 500 also the rate of convergence here was significantly faster. Thus we can see that using two different neural networks is extremely beneficial as the complexity of the environment increases.

BONUS

- Pytorch
- Jira https://namantejaswi.atlassian.net/jira/software/projects/RLP/boards/1

Contribution	Naman	Arshabh
Part 1	50%	50%
Part 2	50%	50%
Part 3	50%	50%