1. Discuss the benefits of:

- **Using experience replay in DQN and how its size can influence the results**

1) Experience Replay is a technique used in Deep Q-Networks (DQNs) to improve the efficiency of learning by storing and reusing experiences from past interactions with the environment. We store the current state, action, reward, and next state in a buffer, and then sample a batch of experiences randomly from the buffer to update the network
2) The size of the experience replay buffer can have a huge impact on the performance of the DQN. When the buffer size is small, the agent won't learn from the experiences that are important. The agent may suffer from high variance in the updates when the buffer size is small.
   Larger buffer means that the agent will learn from various experiences. It also means that the learning will be slow since the agent will be visiting past experiences more than the new ones. The training time increases on increasing the experience replay buffer size.
3) It's usually recommended to have the buffer size proportional to the number of steps in the episode. To mitigate the problems associated with using large replay buffer size and small replay buffer size

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

- **Introducing the target network**

```
model = Sequential(
     [Dense(32, input_shape=(state_size,), activation='relu'),
      BatchNormalization(),
      Dense(32, activation='relu'), Dense(action_size,
activation='linear')])
   model.compile(loss='MSE', optimizer=Adam(learning_rate=0.01),
metrics=['accuracy'])
   model.summary()
```

1) The model is a Sequential model in Keras which means that all the layers are stacked on top of each other linearly.
2) The model has three Dense layers
3) Input to the model is a state vector of size : state_size
4) Output of the model: action_size representing the Q values of each possible action
5) The first dense layer has 32 nodes and uses ReLU activation. The second layer has 32 nodes and uses ReLu Activation.
6) The final Dense layer has an action_size number of nodes and uses a linear activation function.
7) The model is compiled using the mean squared error loss function. The model is optimized using the adam optimizer. The learning rate of which is set to 0.01.

• Representing the Q function as qˆ(s, w)

In function approximation methods like DQNs, the Q function is represented as q^(s,w) where w is the weights of a neural network that approximates the Q function. The weights w are updated using stochastic gradient descent inorder to minimize the mean squared error loss function.

$$Q(s, a; \theta) = r + \gamma \max_{a'} Q(s', a'; \theta')$$

Q(s, a, w) ≈ Q π (s, a)

DQN basically 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. Briefly describe 'CartPole-v1', the second complex environment, and 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.

CartPole-v1

1) CartPole-v1 is a RL environment that requires an agent to balance a pole on top of a cart by moving the cart left or right.
2) The possible actions that the agent can take are to move the cart to the left or right
3) State of the environment is represented by 4 continuous variables -> cart velocity, pole angle, pole velocity, cart position
4) The goal of the agent is to keep the pole balanced as long as possible, with a reward of +1 for each time pole remains upright

OpenAI Atari Breakout:

State: State refers to the current configuration of the game board. State includes the speed and position of the ball, the position of the paddle, orientation of the paddle, and also the arrangement of the bricks.

Actions: The Atari agent can paddle left/right. It can launch the ball at the start of the game.

Goals: Goal of the game is to clear all the bricks on the board by bouncing the ball off the paddle into the bricks.

Rewards: The agent receives rewards for hitting bricks. It receives a higher reward for hitting bricks that are high up on the screen. The Atari agent receives a reward for clearing the entire screen.

Grid World Environment

The number of states : 16. I defined a 4x4 grid world.
We have an action space of 4 different actions - [UP,DOWN,RIGHT,LEFT]

1) Down, action = 0
2) Up, action = 1
3) Right, action = 2
4) Left, action = 3

The number of rewards:

a) Based on the whether the agent is inside the grid or is going out of bounds it is awarded with one of the three values [0,-1,-2]
b) Based on proximity value, the agent is awarded with [-1,0,1]
c) If the agent reaches the goal position, he is rewarded with 10 points
d) If the agent lands up in the bonus2 position and is closer to the goal position than it was before, the agent is rewarded with +5 points. Additionally, if the environment is stochastic with a probability less than 0.5, its awarded with an additional +2 points
e) If the agent lands up in the bonus3 position and is closer to the goal position than it was before, the agent is rewarded with 5 points. Additionally, if the environment is stochastic with a probability less than 0.5, its awarded with an additional 2 points
f) If the agent lands up in the bonus1 position and is closer to the goal position than it was before, the agent is rewarded with 5 points. Additionally, if the environment is stochastic with a probability less than 0.5, its awarded with an additional +2 points
g) If the agent lands up in a pit position, it's rewarded with -5 points. Additionally, if the environment is stochastic with a probability less than 0.75, the agent is given an extra reward of -3

CliffWalking Environment:

CliffWalking environment is reinforcement learning problem that essentially involves an agent navigating a gridworld

States: Environment is represented as a gridworld. Each cell is a state and the agent can occupy any cell. There are start and terminal states. Additionally, there is a cliff region in the gridworld.

Actions: UP,DOWN,LEFT, RIGHT. The actions are deterministic. However, if the agent moves into a cell in the cliff region, it will fall off the cliff and come to start state, getting a negative reward.

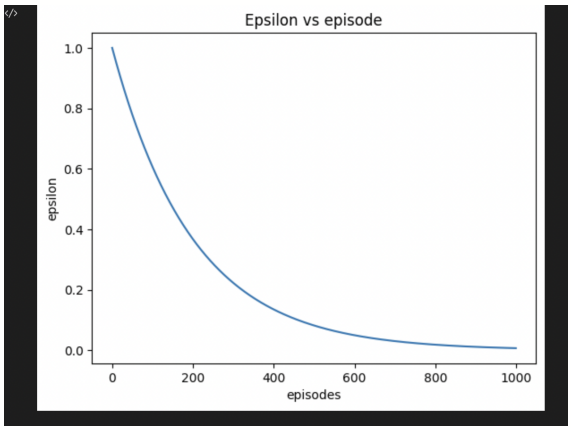
Rewards: It receives a reward of -1 for every timestep in the gridworld, except when the agent falls off in which case it receives a reward of -100. \

Goals: Goal of the agent is to maximize cumulative reward. The agent should reach the terminal state as quickly as possible.

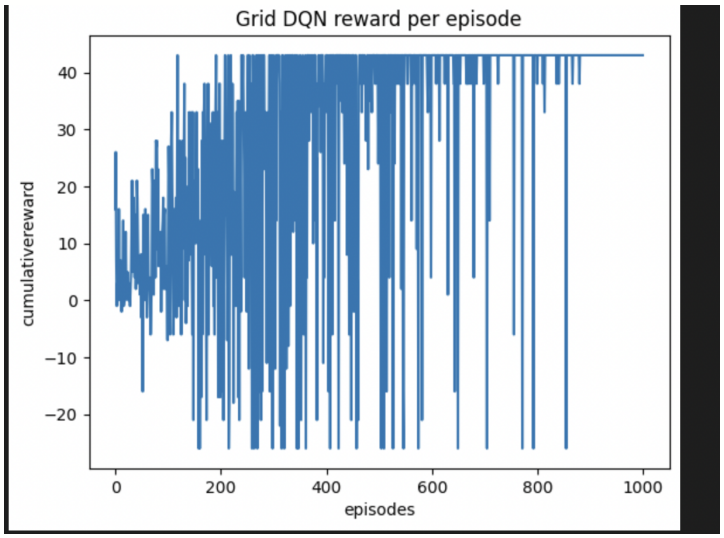
3. Show and discuss your results after applying your DQN implementation on the three environments. Plots should include epsilon decay and the total reward per episode.

1) **Grid World**

Epsilon Decay:

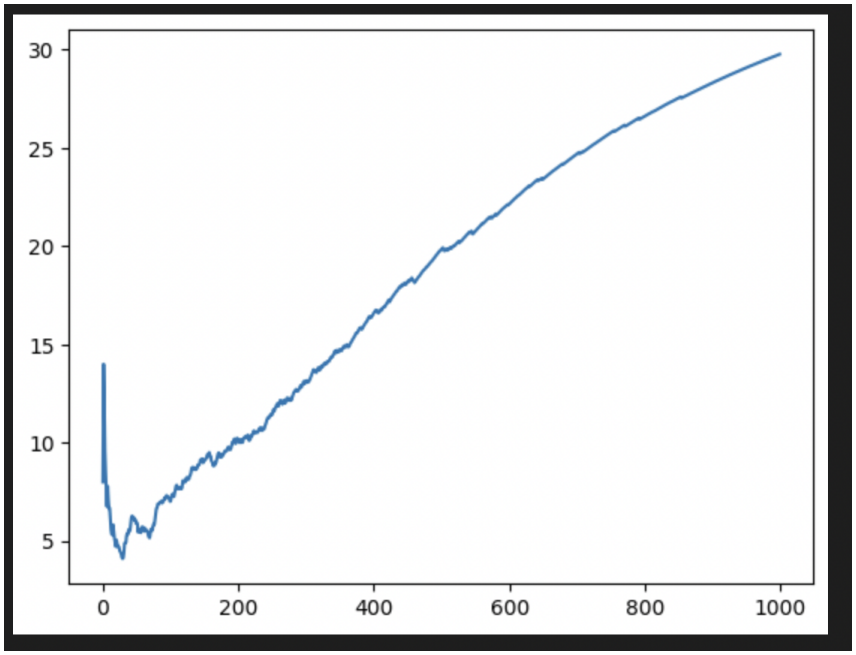


total reward per episode:



From the above we can see that the reward is reaching a maximum value of 40. You can see that the graph has converged.

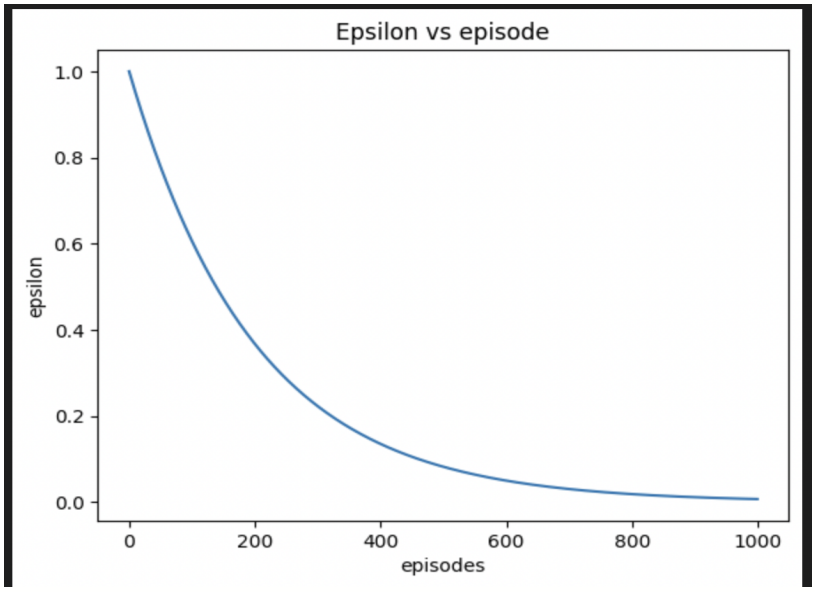
Average reward per episode graph



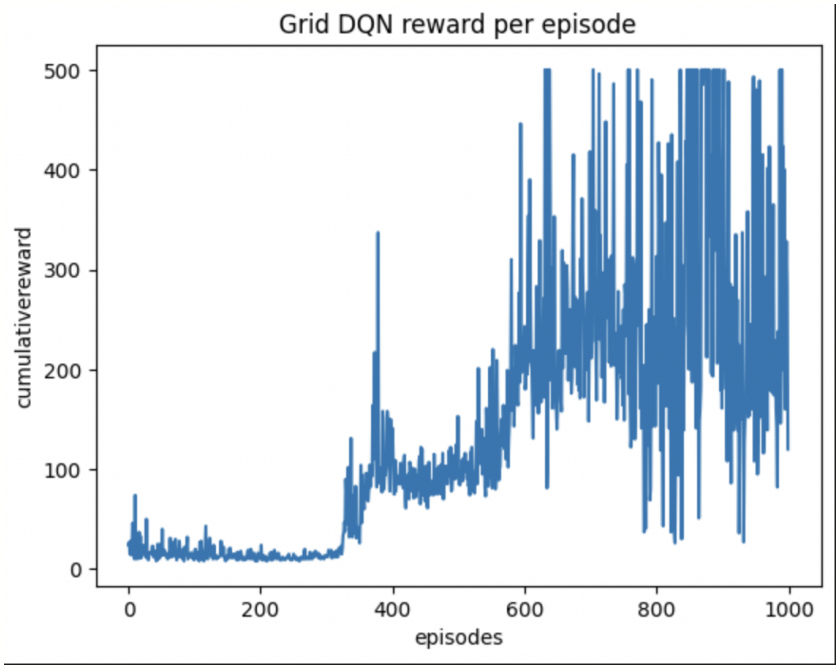
Average reward for the environment is consistently growing indicating that the environment is learning.

2) CartPole-v1

Epsilon Decay:

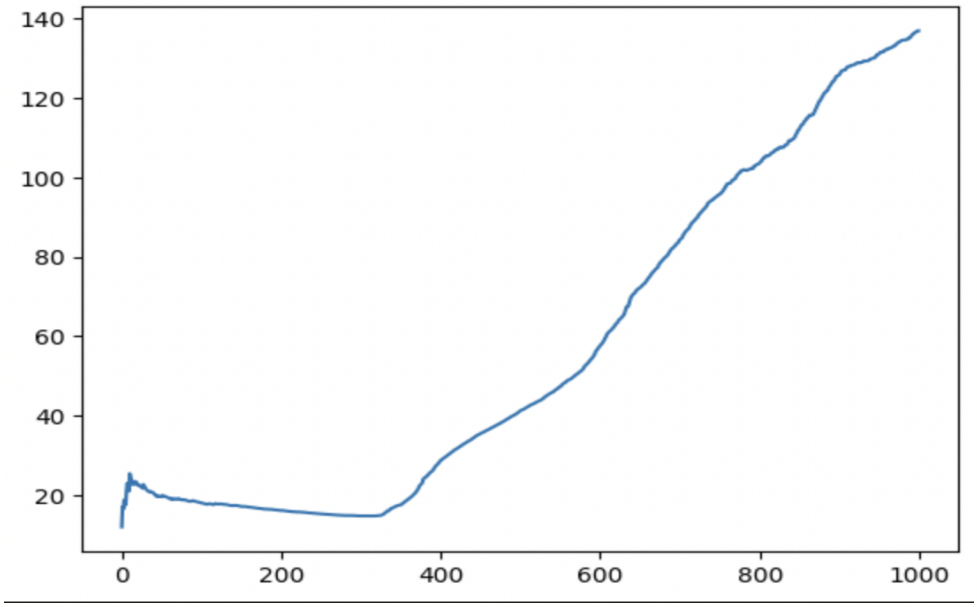


total reward per episode



The total no of rewards for the cartpole environment is reaching a maximum value of 500 indicating the agent is learning well.

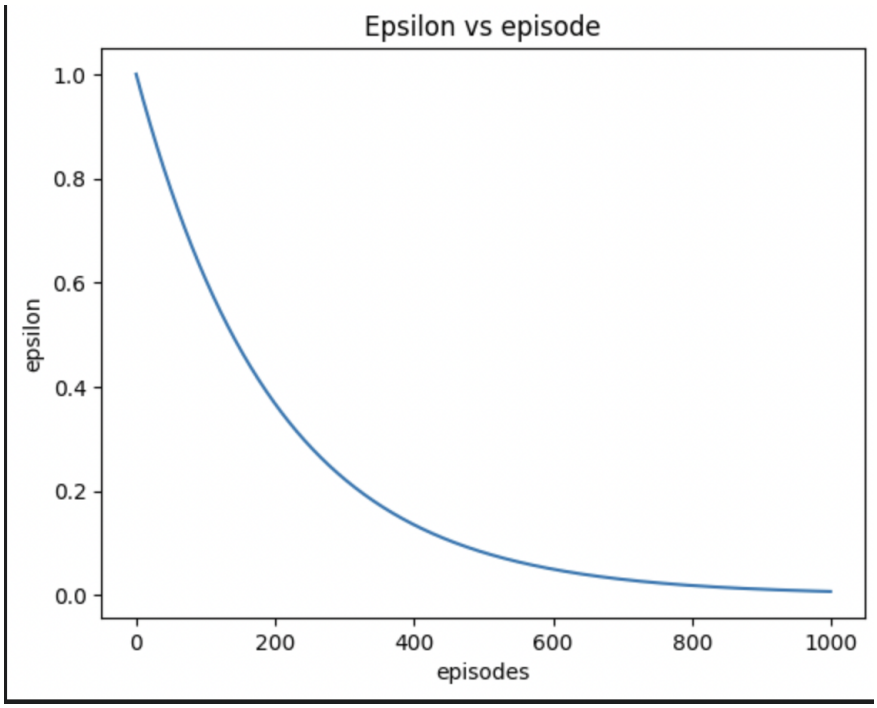
Average reward graph:



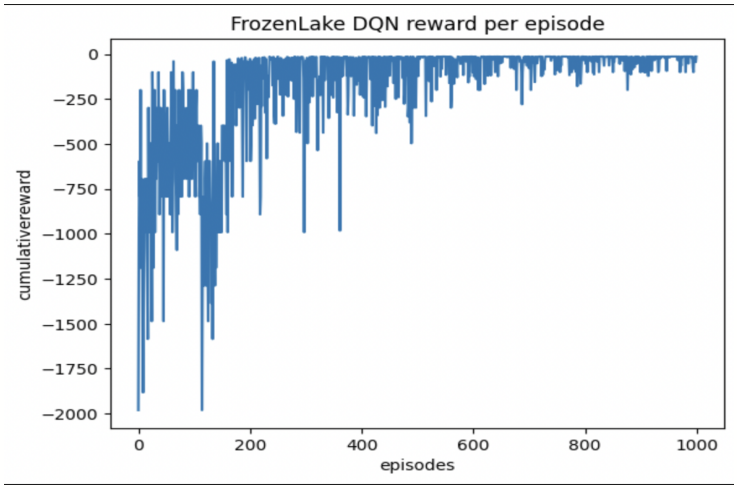
Even though there is a slight dip in the average no of rewards during the first 200 episodes, there's only a significant growth starting from episode 400 indicating that the agent is initially choosing random actions and getting less rewards. Later it is learning indicating a stark growth in the total rewards per episode.

CliffWalking:

Epsilon Decay:

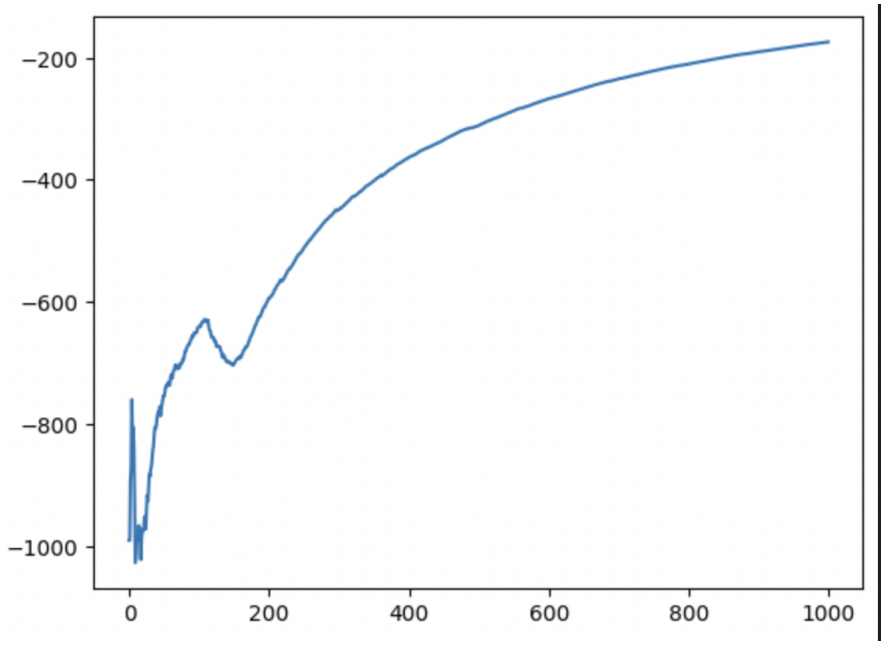


Total rewards per episode:



The total reward per episode graph has converged by the episode number 200 showing that the agent has learned.

Average reward per episode graph:

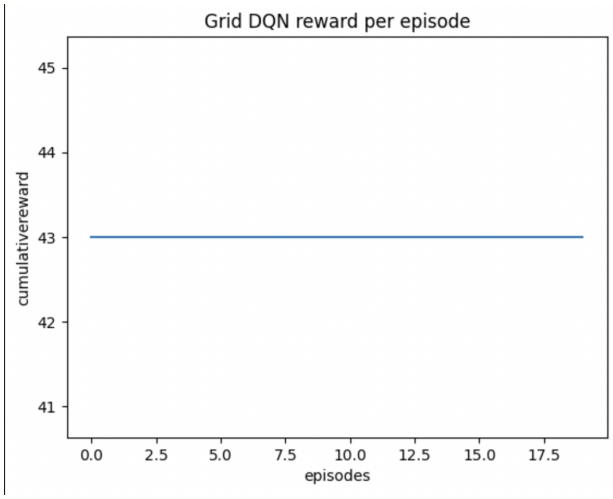


The average reward per episode graph is an increase curve indicating the agent is learning well. The maximum average reward achieved by the agent is -200. It started from -1000 and went on to become -200 by the time it reached 1000 episodes.

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

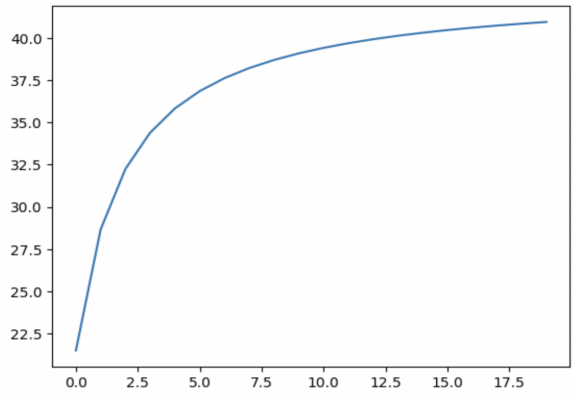
Grid World:

Total Rewards per episode



The average reward during the evaluation for the grid world is constant in a deterministic environment, The agent was able to achieve a maximum reward of 43.

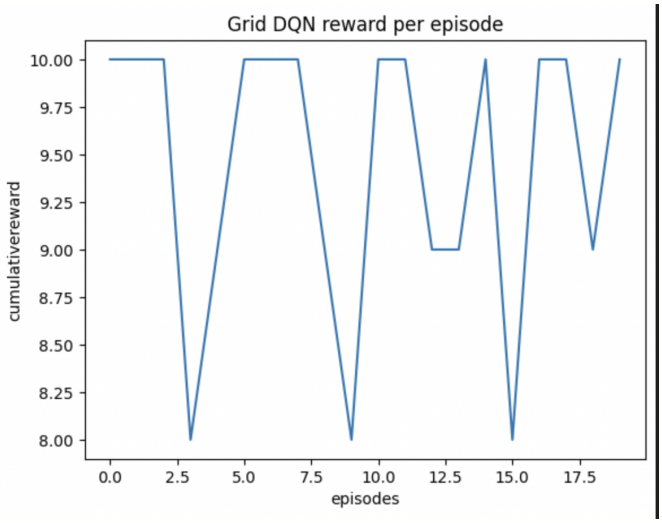
Average rewards per episode



The average reward has consistently grown during the evaluation for the grid world environment. The maximum average reward for grid world environment being 17.5

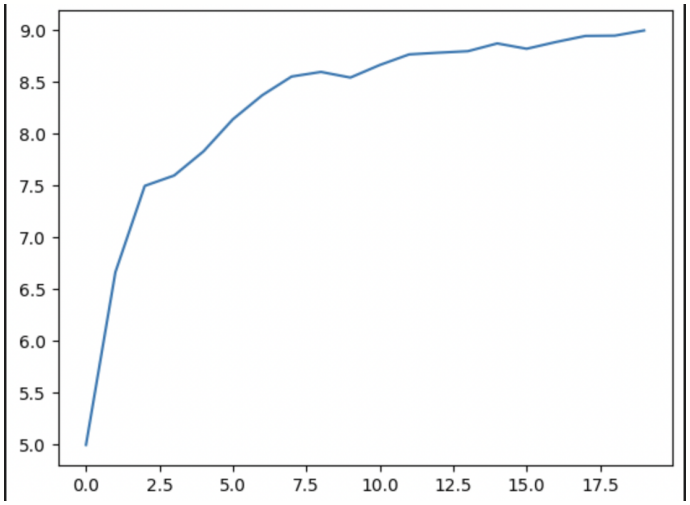
CartPole-v1

Total Rewards per episode

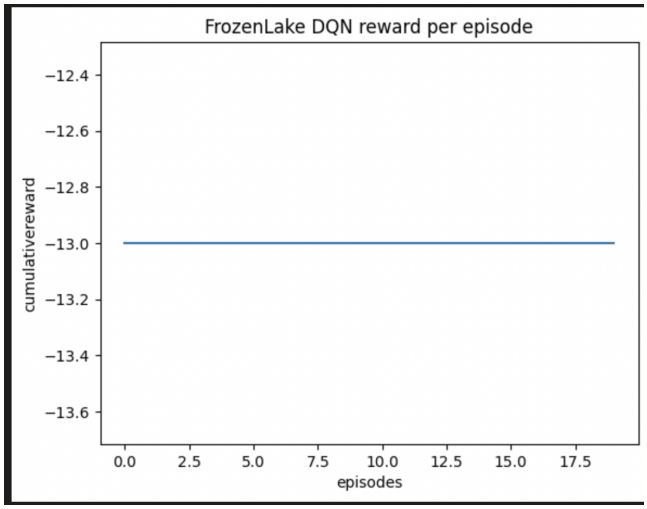


The total rewards graph for the cartpole is zig zag. But from the below, you can see that the average moving reward per episode is only growing and reaching to a maximum value of 9.0

Average running reward per episode

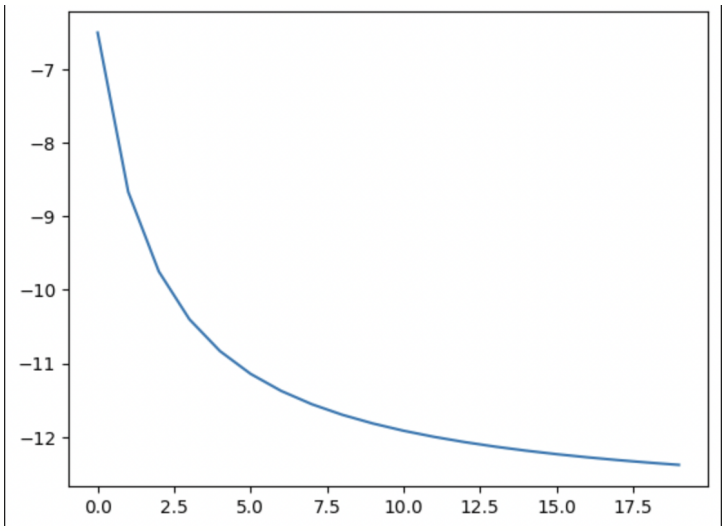


Cliff Walking:



Total rewards per episode remains constant during the evaluation. Cliff Walking is essentially a Deterministic Grid World.

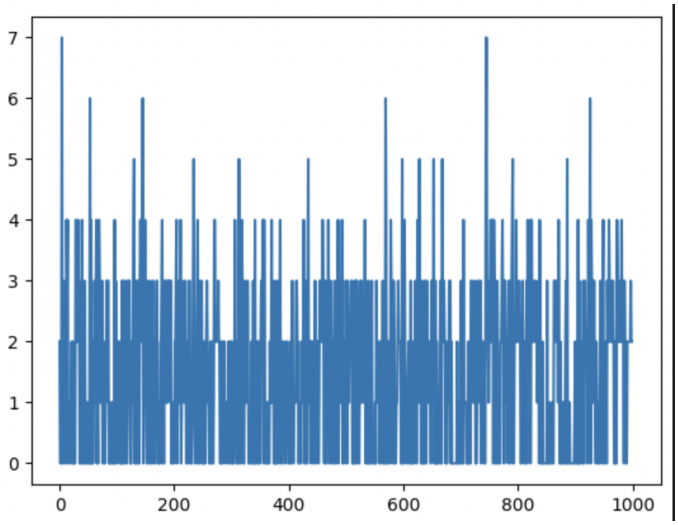
Average reward per episode graph:



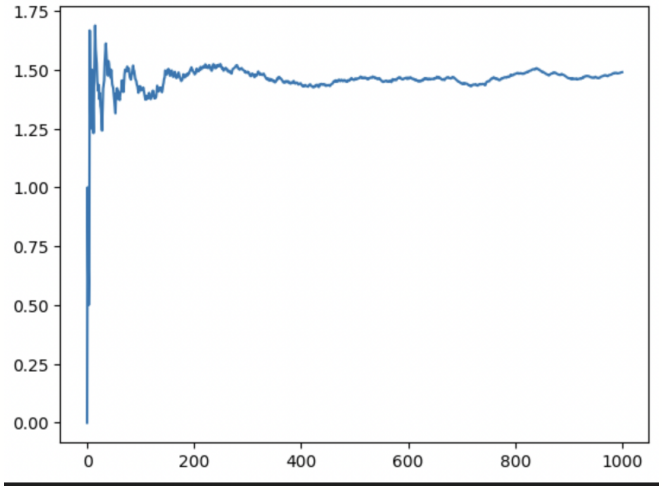
BONUS:

3. OpenAI Atari breakout

Total rewards per episode graph:



Running Average Reward per episode:



From the above graph, you can see that the agent is learning. It is able to achieve positive rewards. From being 0 in the beginning to reaching to 1.75 is a significant achievement for the agent. We need to run for more iterations to achieve higher rewards.

The Atari environment considers each frame of the video as a timestamp, this causes the system to take atleast 100 timestamps for the ball in the breakout env to go from top to bottom of the screen. And in our code we see that the amount of time the environment runs increases from 2 seconds to 280sec. This shows that the agent is learning to balance the ball without dropping it. Rewards will be generated after the ball hits the bricks and learning this would take atleast 10000 time steps per episode. Running each iteration for 10k steps takes around 120-150sec per episode even on colab and upto 200 sec on CCR. This caused the training time of the system to go upward of 10hours for just 1000 iterations. Hence we couldn't train it enough to generate a good enough reward. But the increase in timesteps before episode is a good indicator that the episode keeps going on and the agent is learning to keep the ball balancing. The few times the agent is managing to hit the brick it is getting a minor reward and hence pushing it to keep the ball bouncing and to not drop.

5. Provide your interpretation of the results. E.g. how the DQN algorithm behaves on different envs.

1) DQN algorithm has performed really well on the grid world environment. The graph of the total rewards per episode and average reward per episode have only grown
2) Even for the Cartpole environment, the graph of the total rewards per episode and average running reward per episode indicate the agent was able to learn and achieve really good rewards
3) For the Atari environment, the agent achieved positive reward from being at 0 initially indicating the agent has learnt.
4) For CliffWalking environment, DQN algorithm has performed really well. The total reward per episode graph kept increasing until a certain point and then converged during the training. During the evaluation, the agent was able to achieve constant rewards since it was in a deterministic environment.

Overall, the DQN algorithm has performed really well on all the three environments i.e. Grid World, Cartpole, Atari, and CliffWalking

Include all the references that have been used to complete this part:

1) **https://www.tensorflow.org/tutorials/reinforcement_learning/actor_critic#:~:text=CartPole%2Dv0,-In%20the%20CartPole&text=A%20reward%20of%20%2B1%20is,2.4%20units%20from%20the%20center.**

2) **https://blog.tensorflow.org/2018/07/deep-reinforcement-learning-keras-eager-execution.html**

In your report for Part III:

1. Discuss the algorithm you implemented

a) Dueling DQN uses a neural network to estimate the Q values. Unlike DQN, it separates the Q-value function into two parts: the state value function and the advantage function. By combining both, Dueling DQN improves the efficiency and stability of the DQN algorithm especially with large action spaces

b) Dueling DQN has been shown to perform better than DQN on various tasks including Atari Breakout games and other robotic games too. Dueling DQN has numerous variants such as Prioritized Dueling DQN and Double Dueling DQN .

c) The state value function in Dueling DQN represents how good it is to be in a particular state regardless of the action you take. It essentially estimates how good a state is. Whereas the advantage function determines how good it is to take a certain action given a state.

d) Given a function Q, it is hard to determine the values of V and A. Hence we force the highest value of Q to be equal to the value of V. This forces the maximum value of the advantage function to be zero.

e) Dueling DQN calculates the mean value of the advantage function instead of the maximum of the advantage function.

The model we used for Dueling DQN is as follows:

```
input = tf.keras.layers.Input(shape=state_size)
   dense1 = tf.keras.layers.Dense(64, activation='relu')(input)
   batch_norm = tf.keras.layers.BatchNormalization()(dense1)
   dense2 = tf.keras.layers.Dense(128, activation='relu')(batch_norm)
   value, advantage =
tf.keras.layers.Dense(1,activation='linear')(dense2),
tf.keras.layers.Dense(action_size,activation='linear')(dense2)
   q = value + (advantage - tf.reduce_mean(advantage, axis=1,
keepdims=True))
   return tf.keras.models.Model(inputs=input, outputs=q)
```

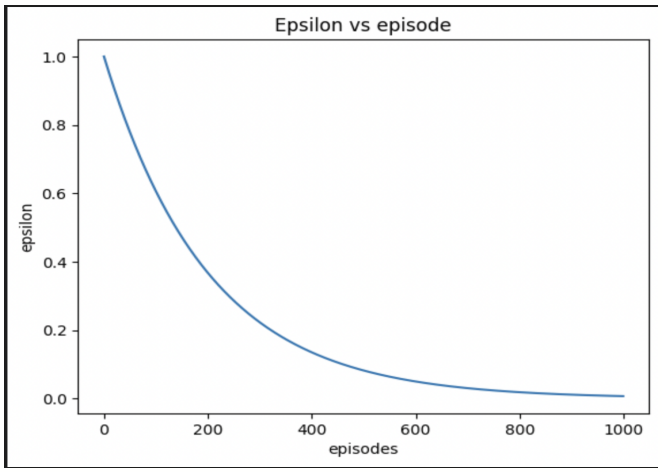
2) What is the main improvement over the vanilla DQN?

1) Separation of value and advantages functions in Dueling DQN allows us to estimate the advantages of one action over others. This division of functions allows the agent to understand the goodness of each action instead of randomly assigning values to each action.
2) Dueling DQN uses less memory in comparison to DQN.
3) By decoupling the value and advantage functions, Dueling DQN ensures that there is better exploration of the action space.

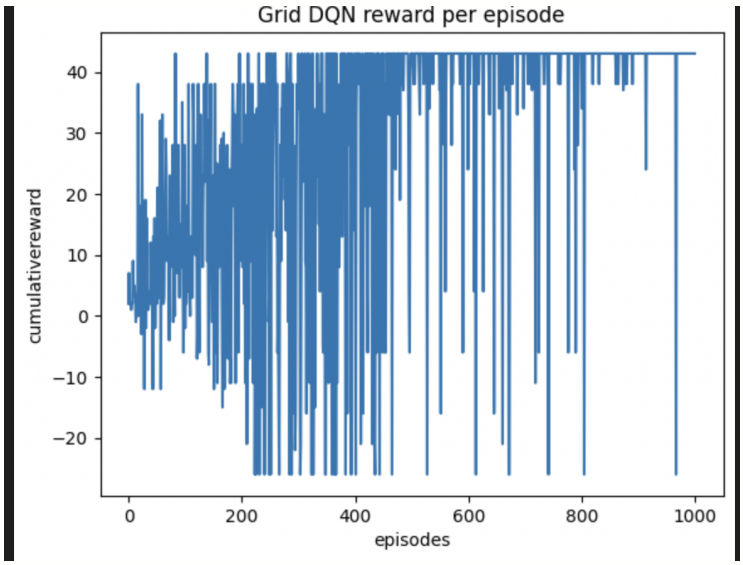
3)Show and discuss your results after applying your the two algorithms implementations to the environment. Plots should include epsilon decay and the total reward per episode.

Dueling DQN:

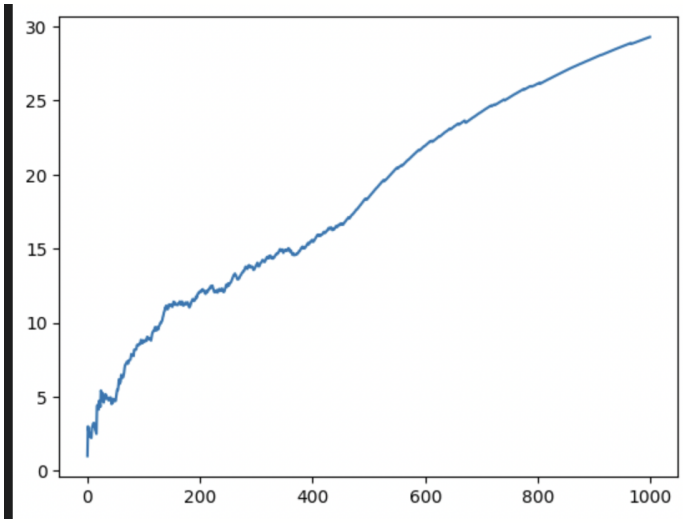
Epsilon Decay:



Total rewards per episode graph:



Average rewards per episode graph:

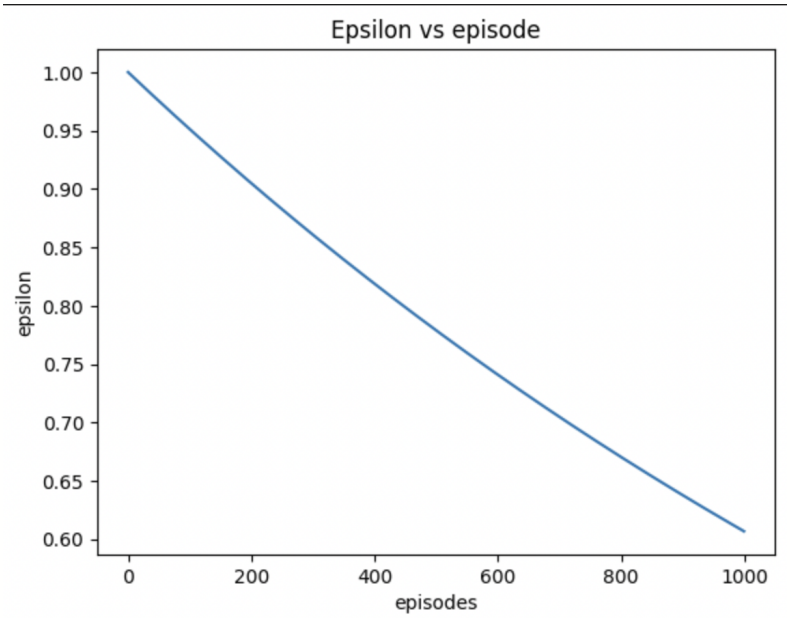


Results:

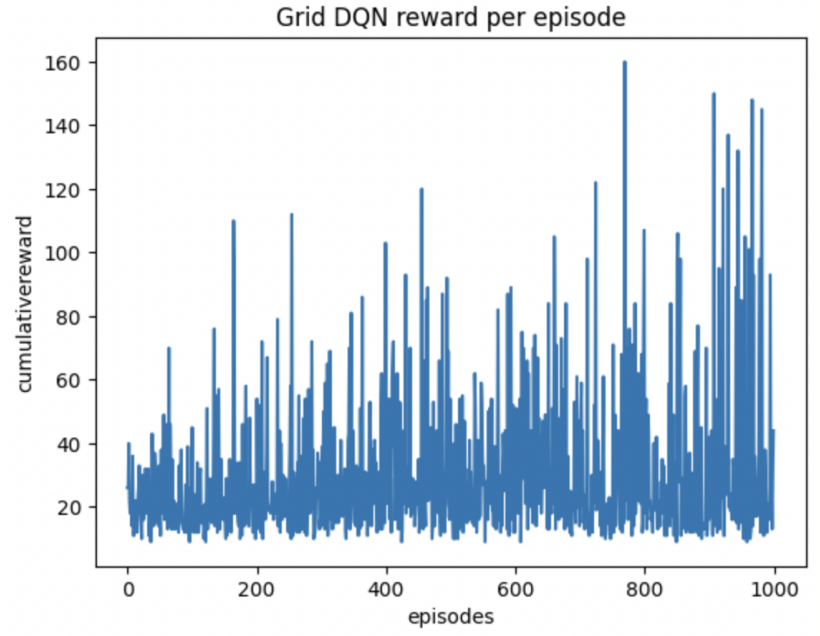
1) The average reward per episode is increasing much faster with Dueling DQN in comparison to DQN
2) The maximum reward obtained in the grid world environment using both the algorithms is 40.
3) Average reward per episode showed a slightly better increasing cover using the DQN environment than the Dueling DQN.
4) Clearly, we can say that the Dueling DQN algorithm performed better in comparison to the DQN algorithm for the grid world environment we designed.

CartPole for Dueling DQN:

Epsilon Decay graph:

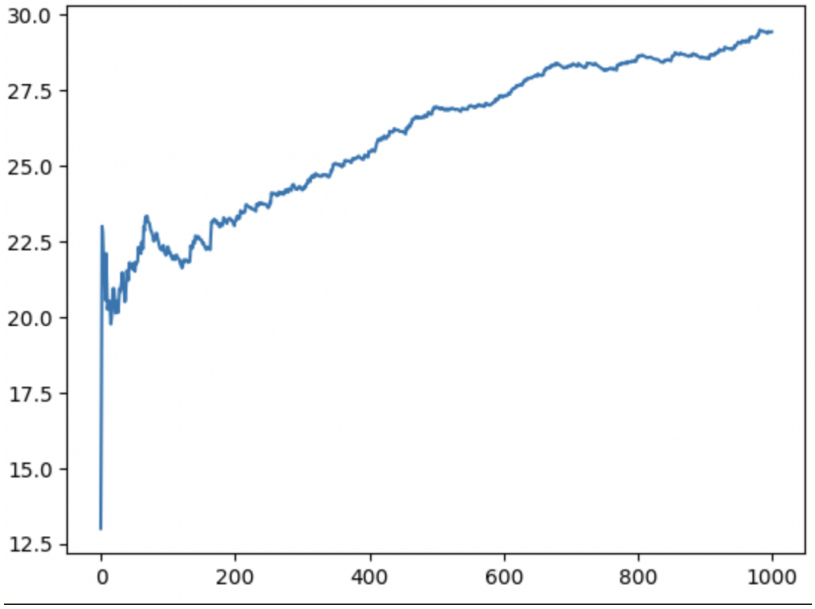


Total rewards per episode graph:



From the above graph, you can see that the total rewards is increasing with increasing no of episodes indicating that the agent is learning

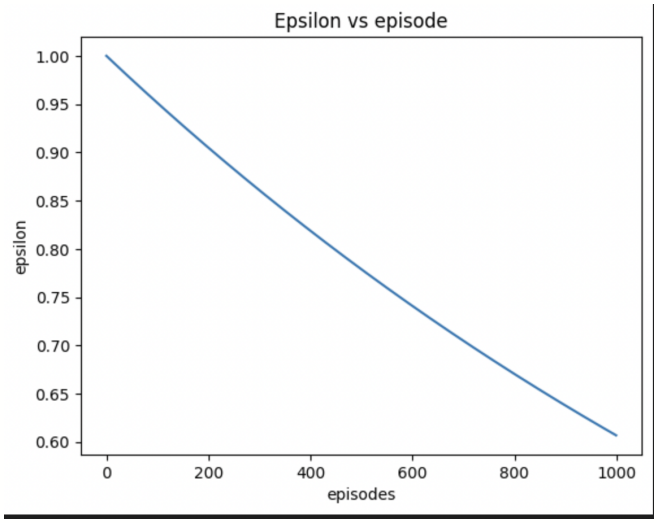
Average rewards per episode graph:



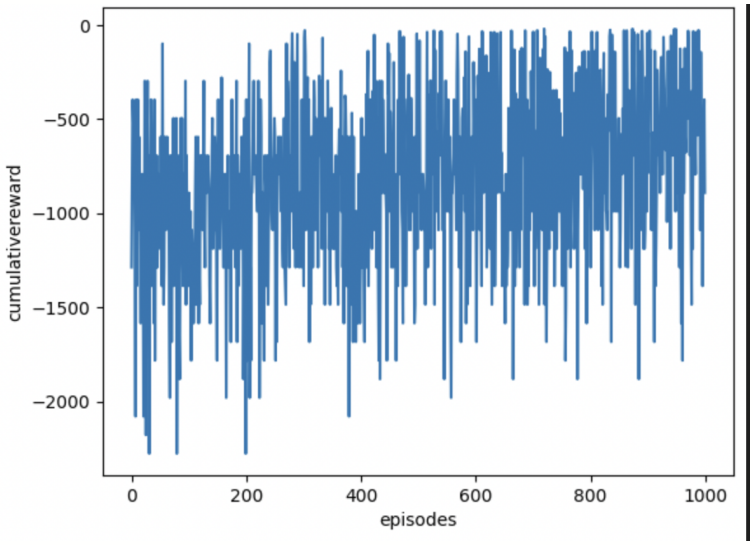
From the above, you can observe that the average no of rewards is increasing starting from the episode 10 indicating that the agent is learning. The average reward graph is an increasing curve indicating the growth.

Cliffwalking:

Epsilon Decay:

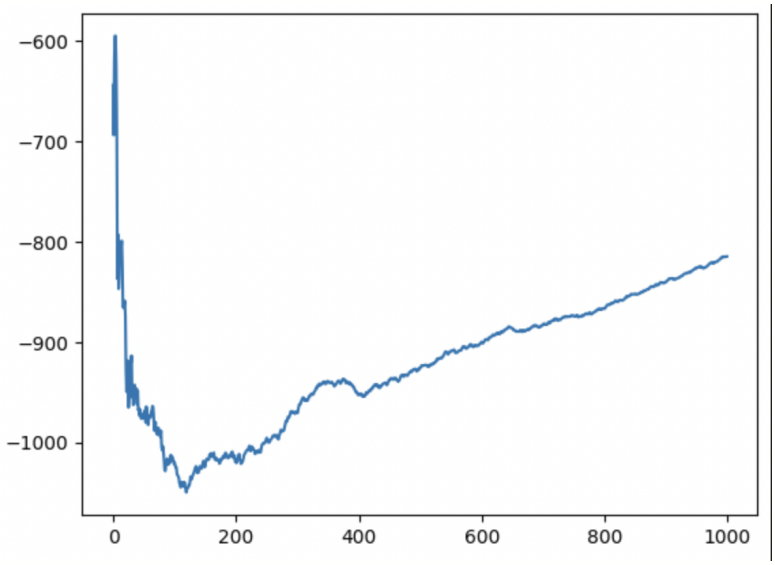


Total reward per episode:



You can see that the total rewards per episode is growing here indicating the algorithm is working really well for the environment.

Average reward per episode:

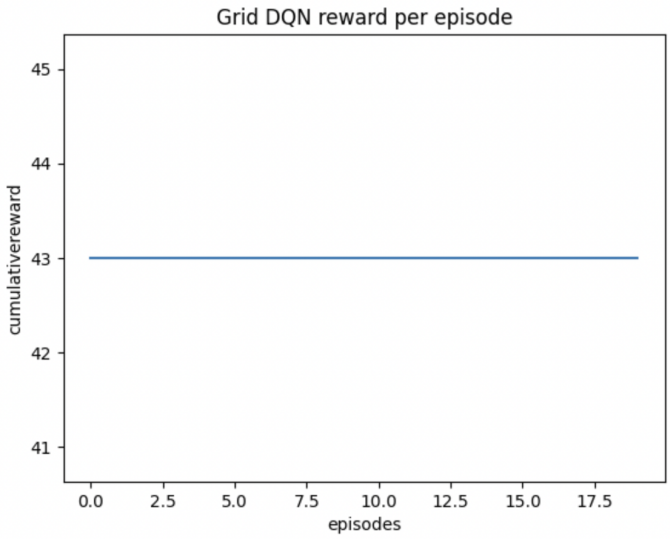


The above graph is an increasing curve indicating that the agent is learning. The rewards are consistently growing. Dueling DQN is working well really in this environment too.

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.

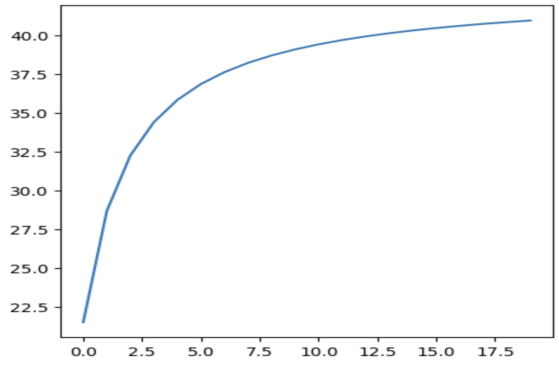
Grid World Environment:

1) **Total rewards per episode**



During the evaluation, the agent was able to achieve a maximum reward of 43 indicating that the agent has learned really well using the Dueling DQN algorithm.

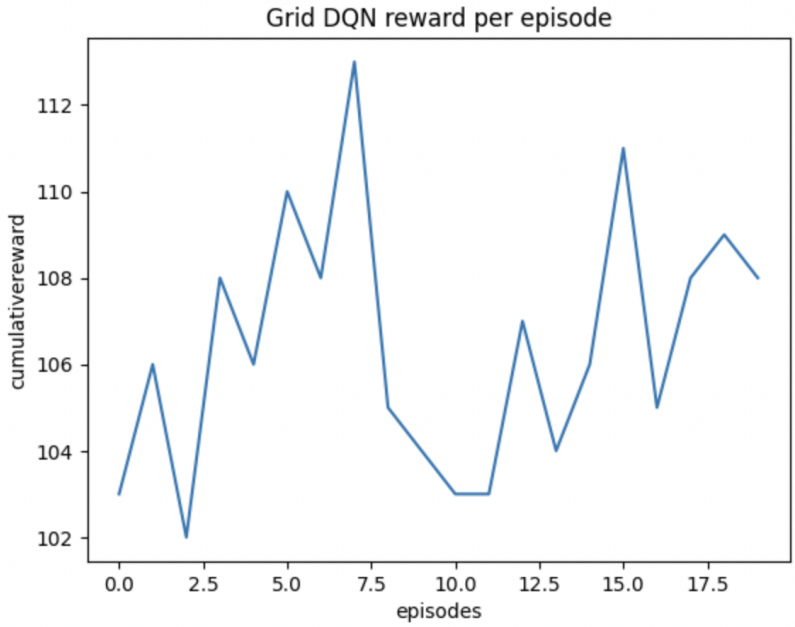
2) **Moving average reward per episode**



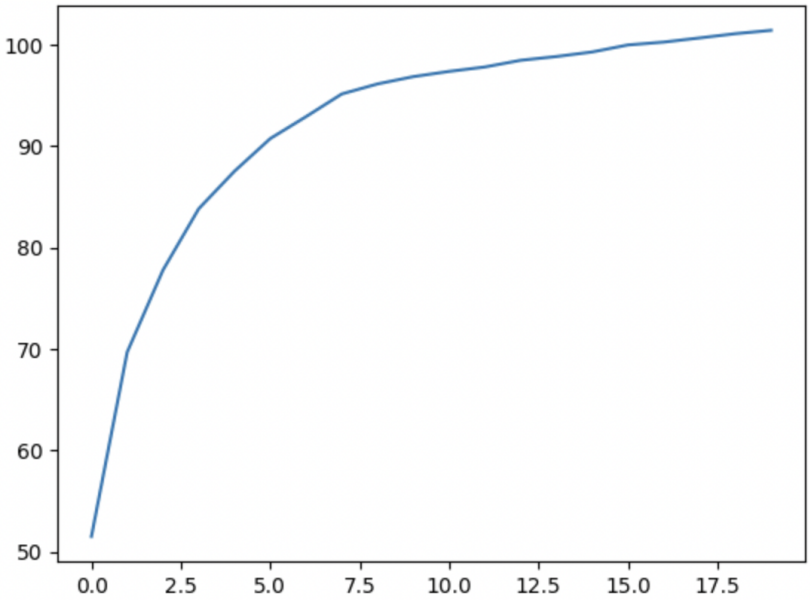
The average reward per episode graph is only growing, reaching upto a maximum of 40 rewards when it has reached 20 episodes.

CartPole-Dueling DQN evaluation results:

1)**Total rewards per episode**



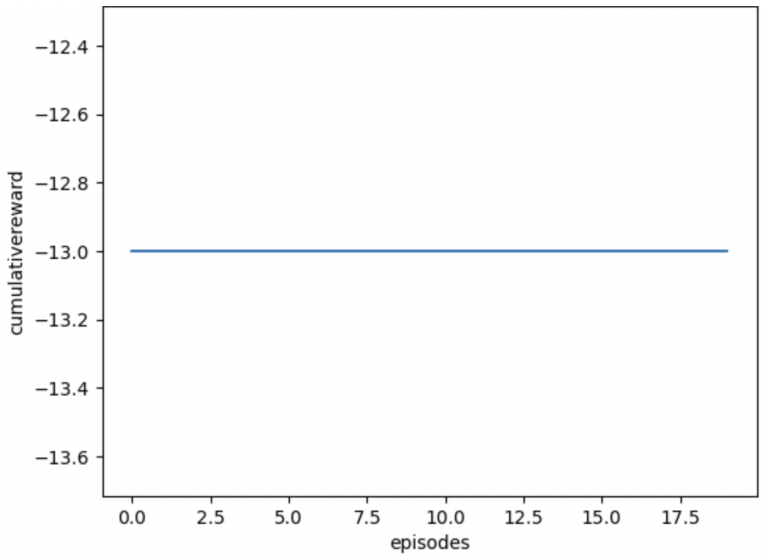
2) Average reward per episode graph



From the above two graphs, we can deduce that the agent was able to achieve an average reward of upto 100 by the time it has reached 20 episodes. The average rewards graph is increasing indicating that the agent has learned well.

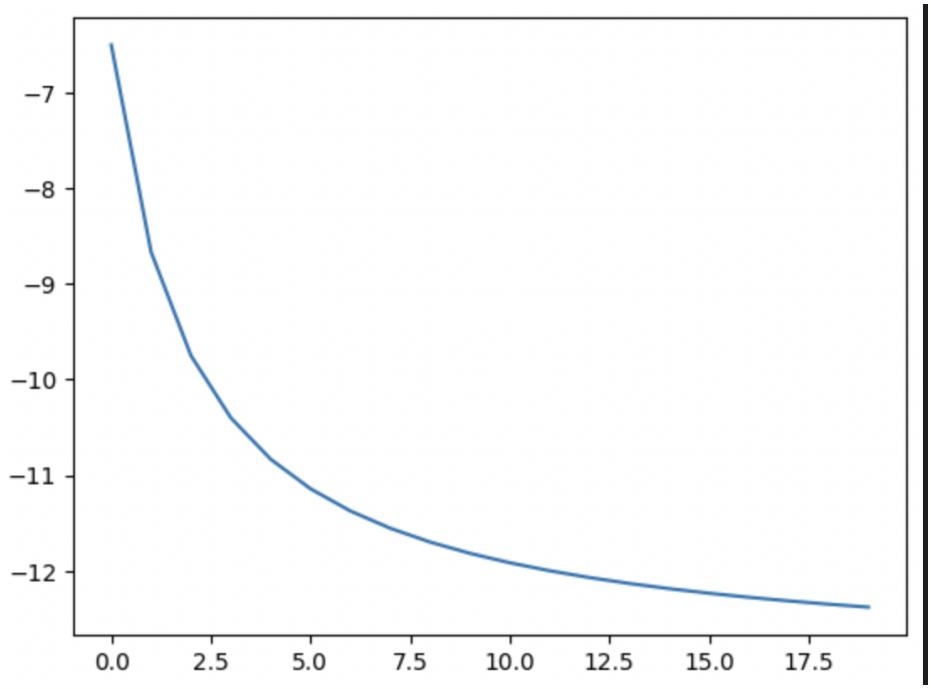
Cliffwalking Dueling DQN evaluation results:

Total rewards per episode:



Total rewards during the evaluation have remained constant in the Cliffwalking environment using the Dueling DQN algorithm

Average rewards per episode graph:

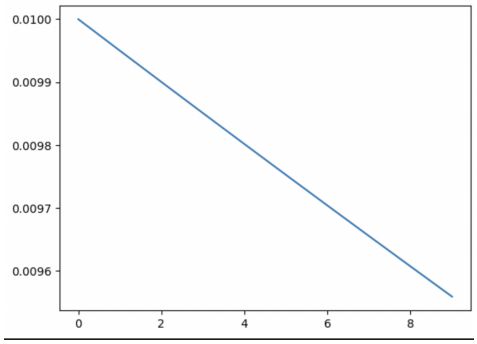


Bonus:

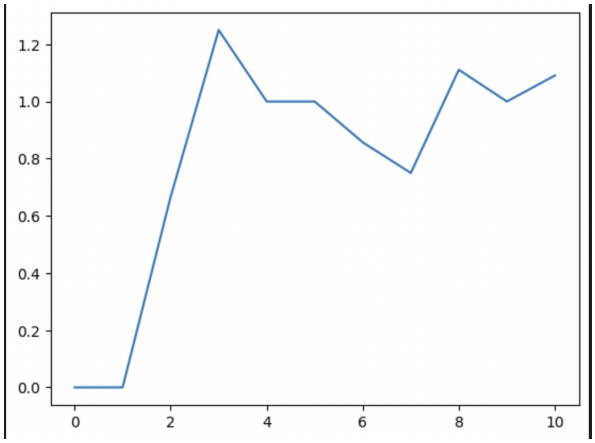
Atari Breakout-Dueling DQN evaluation results:

Atari Breakout - Dueling DQN:

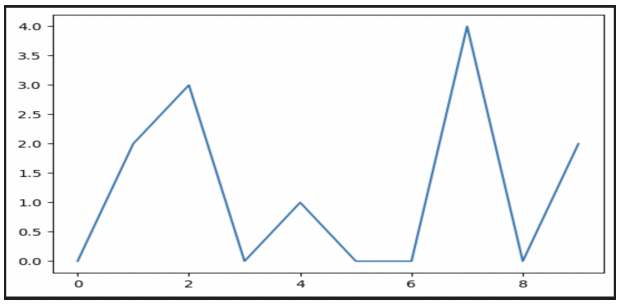
The below is an epsilon decay graph for the Open AI Atari Breakout environment



The below graph is total number of rewards per episode using the Dueling DQN during evaluation:



The below graph is average rewards per episode graph:



The Atari environment considers each frame of the video as a timestamp, this causes the system to take atleast 100 timestamps for the ball in the breakout env to go from top to bottom of the screen. And in our code we see that the amount of time the environment runs increases from 2 seconds to 280sec. This shows that the agent is learning to balance the ball without dropping it. Rewards will be generated after the ball hits the bricks and learning this would take atleast 10000 time steps per episode. Running each iteration for 10k steps takes around 120-150sec per episode even on colab and upto 200 sec on CCR. This caused the training time of the system to go upward of 10hours for just 1000 iterations. Hence we couldn't train it enough to generate a good enough reward. But the increase in timesteps before episode is a good indicator that the episode keeps going on and the agent is learning to keep the ball balancing. The few times the agent is managing to hit the brick it is getting a minor reward and hence pushing it to keep the ball bouncing and to not drop.

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:

• Grid-world environment

• 'CartPole-v1'

• Gymnasium envs or Multi-agent environment or any other complex environment

1) Grid World Environment:
   1)Both DQN and Dueling DQN performed relatively well for the Grid World Environment.
   2) Convergence rate is higher in Dueling DQN in comparison to DQN
   3) Dueling DQN reaches the maximum value by 200 episodes, where DQN takes around 400 episodes before the convergence.\
2) CartPole-v1
   1) For the Cartpole-v1 environment, DQN performed better than Dueling DQN
   2) The maximum average reward went upto 140 when running with DQN algorithm whereas the maximum average reward could go only upto 30 for Vanilla DQN indicating DQN performed better in comparison to Dueling DQN.
3) Cliffwalking
   1) For the Cliffwalking environment, DQN performed better in comparison to the Dueling DQN environment
   2) The maximum average reward for the cliffwalking environment went upto -200 using DQN. It started from -1000 and climbed upto -200. Whereas, when running with the Dueling DQN it started from -600 and become even lower and fell to -800. The total rewards per episode using both the algorithms converged.

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

Environment	Comparison of Results we obtained
Grid World	DQN < Dueling DQN
Cartpole	DQN > Dueling DQN
CliffWalking	DQN > Dueling DQN

7. Include all the references that have been used to complete this part

1) **https://medium.com/@lgvaz/understanding-q-learning-the-cliff-walking-problem-80198921abbc**
2) **https://arxiv.org/abs/1511.0658**

Contributions:

Name	**Assignment Part**	**Contributions**
Vivek Kuchibotla and Anurima Vaishnavi Kumar	**Part 1**	**50% and 50%**
Vivek Kuchibotla and Anurima Vaishnavi Kumar	**Part 2**	**50% and 50%**
Vivek Kuchibotla and Anurima Vaishnavi Kumar	**Part 3**	**60% and 40%**
Vivek Kuchibotla and Anurima Vaishnavi Kumar	**Bonus**	**70% and 30%**