Report – Project 2

Name: Sandesh Kumar Srivastava

UBIT: Sandeshk

Part 1 - Implementing and applying DQN

In this assignment, we are implementing value function approximation algorithm, deep Q learning (DQN) following DeepMind's paper. Following are some of the salient features of DQN algorithm:

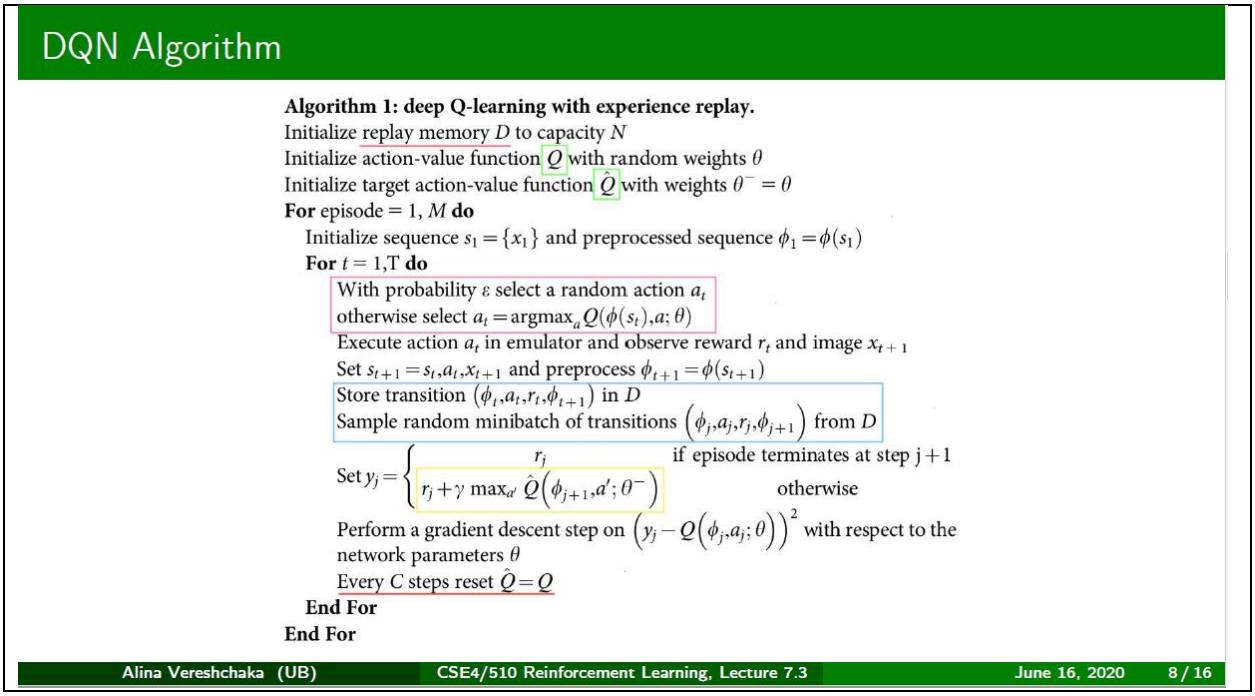
1. Experience replay: In order to efficiently select data for training of large neural networks, DeepMind team introduced the technique of experience replay. Experience replay allows us to store the experience gained (S_t, a_t, r_t, S_{t+1}) in a memory segment for various episodes and then sample a subset of these experiences for training the neural network. Experience replay provides various benefits such as better efficiency as each experience is used in many iterations as training data; random sampling breaks the correlation among training data and it also avoids negative feedback loops that might shift the model to an unwanted local minima.
2. Target network: We use the concept of target network in order to solve the moving target problem by using a separate model for generating target. We periodically sync the target model with the actual model weights. Target network provides more stability to the algorithm by introducing a delay between an update to action value function Q and its effect on training data thus avoiding oscillations.
3. Approximate action-value function Q<- q(s,w): Since determining the action-value function optimally using Bellman equation, $Q_{i+1}(s, a) = E [r + gamma* max_{a'} Q_i(s', a') | s, a]$, is impractical for large systems, we use a function approximator to estimate the action-value functions. Furthermore, instead of linear function approximator, DeepMind team used a non-linear function approximator in neural network.

Environment:

I have used Open AI Gym's **CartPole-v1** environment for checking DQN performance. Some of the details of this environment are as follows:

a) Observation/State: It is of the form of Box (4) where the four values correspond to cart position, cart velocity, pole angle and pole angular velocity.
b) Actions: It is of the form of Discrete (2) and consists of Action set (A) = {0,1} where 0 means push cart to left and 1 means push cart to right
c) Reward: It is 1 for each step till the game terminates. Reward set ® = {1}
d) Episode termination: Episode terminates when pole angle is more than 12 degrees or cart reaches the edge of display (cart position more than 2.4) or episode length is more than 500.
e) Starting state: It is randomly selected from all possible states.

DQN Algorithm:



Implementation details:

1. I have implemented two classes:
   - DQNAgent: It is the agent class where all the main logic to implement DQN algorithm resides.
   - Environment: It is the class which prepares the CartPole-v1 environment using gym's environment and uses DQNAgent as agent to solve this environment.
2. In DQNAgent class, I have defined following methods:
   - __init__: Is used to initialize the environment variables and hyper parameters. It also creates variables for storing rewards metrics and also creates two neural network models viz. model and target.
   - build_model: Function to create neural network used by the algorithm.
   - step: Implements epsilon-greedy approach of selecting a random action with probability epsilon and optimal action with probability 1 – epsilon.
   - sync_target: Is used to update the target model weights with model weights after every C steps.
   - replay: Is used to sample minibatch of experiences from memory and use it for performing gradient descent step.
   - update_epsilon(): It updates the epsilon value after every episode.
3. In Environment class, I have defined following methods:
   - __init__: Is used to create the environment from Open AI gym class and create an agent instance of DQNAgent. It is also used to initialize some hyper parameters like number of episodes for training etc.
   - train(): Here we perform different steps for various episodes. It calls different agent methods for performing various tasks.

Results:

We record the award collected by the agent for each episode and observe that initially the agent performs poorly collecting very few rewards for each episode. However, after sufficient training, the agent starts performing better and towards the end it is able to achieve the maximum award quite a number of times.

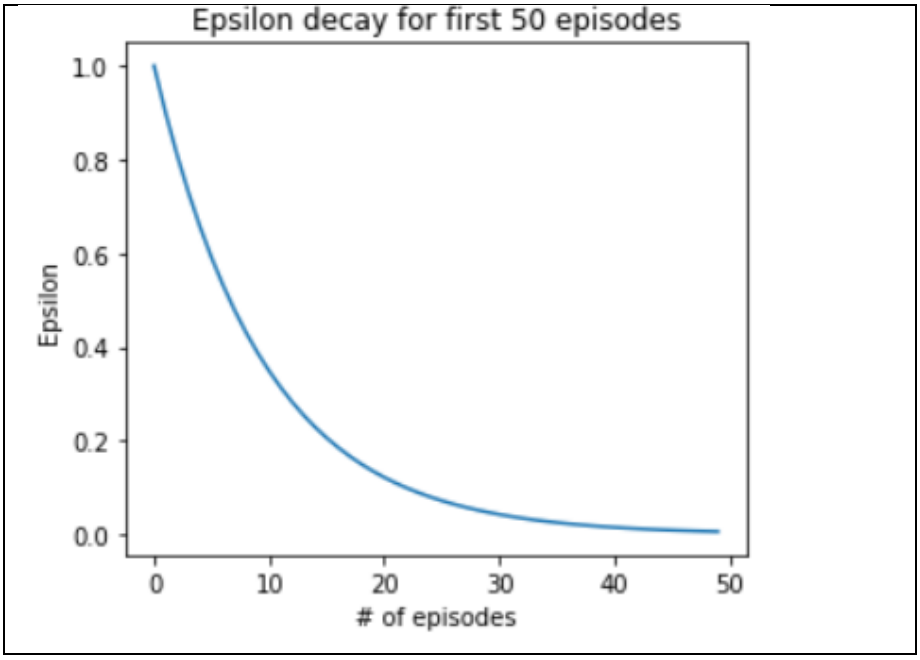
Hyper-parameters used for capturing below results are listed here:

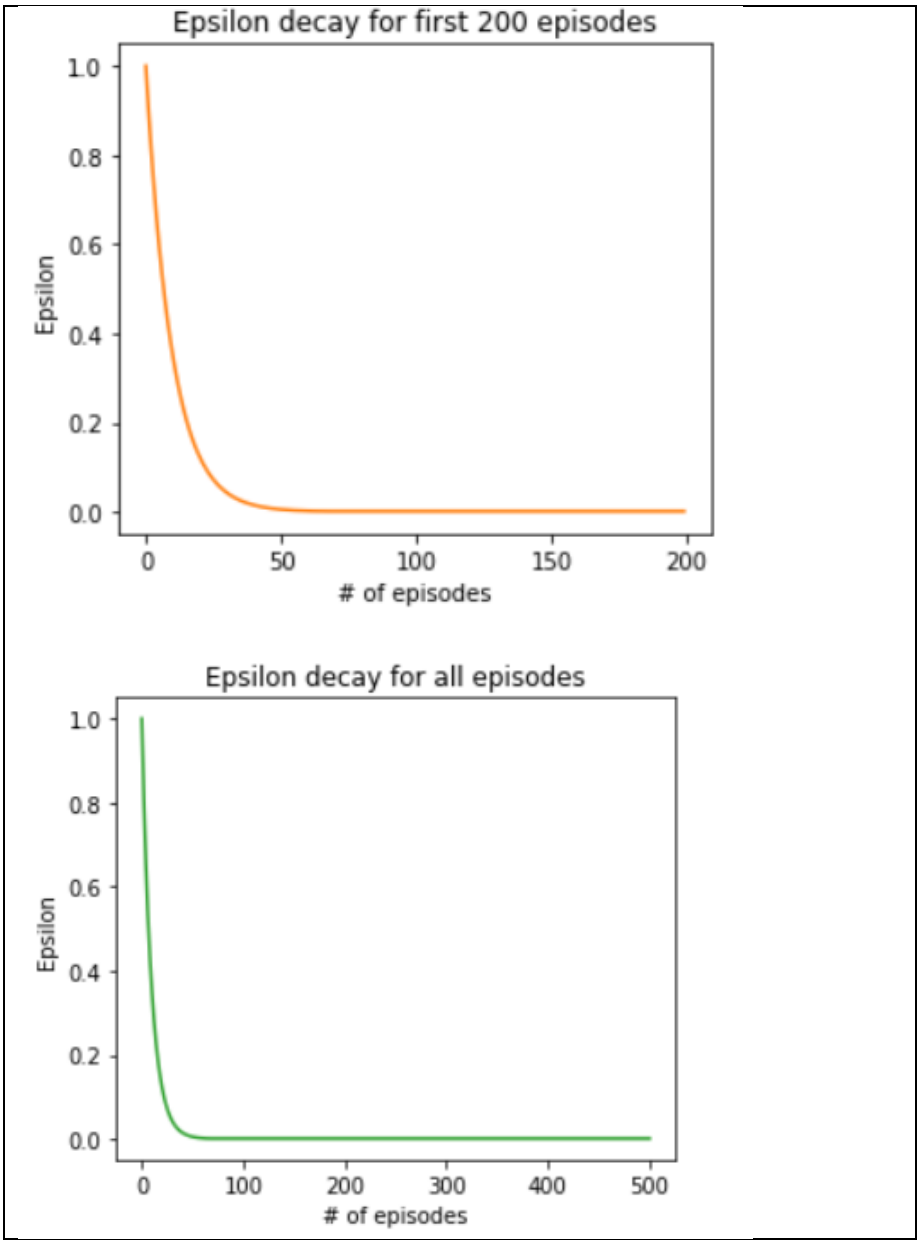
```
self.memory = deque(maxlen=50000)
self.gamma = 0.9
self.epsilon_max = 1.0
self.epsilon_min = 0.001
self.epsilon_decay = 0.9
self.batch_size = 32

self.number_of_episodes = 500
self.C = 1
self.reward_average_episodes = 10
```

The results are captured with following graphs:

1. **Epsilon decay graph**: Here I have shown the variation of epsilon value with the number of episodes encountered. To illustrate it better, I have captured 3 graphs of epsilon-decay with first 50 episodes then with first 200 episodes and finally with all episodes.

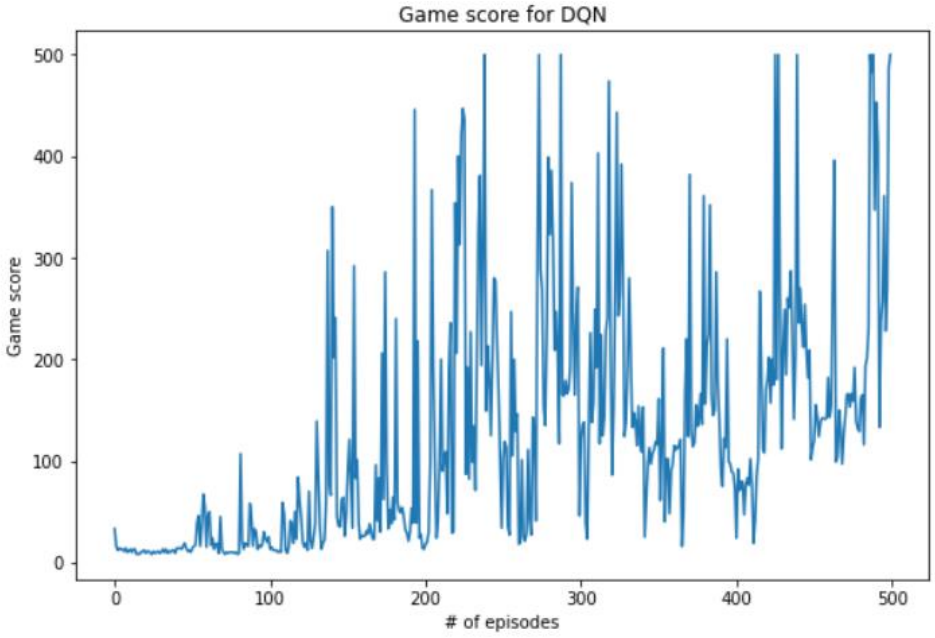




2. Game score graph: This graph captures the reward collected by agent for each episode.

```
Game score for DQN:  [33.0, 16.0, 12.0, 14.0, 13.0, 13.0, 11.0, 14.0, 10.0, 11.0, 13.0, 10.0, 13.0,
13.0, 8.0, 9.0, 8.0, 10.0, 11.0, 12.0, 9.0, 11.0, 11.0, 9.0, 8.0, 11.0, 9.0, 10.0, 11.0, 9.0, 10.0,
13.0, 10.0, 13.0, 9.0, 11.0, 10.0, 12.0, 12.0, 9.0, 14.0, 14.0, 14.0, 13.0, 16.0, 19.0, 14.0, 11.0,
12.0, 10.0, 14.0, 16.0, 17.0, 39.0, 46.0, 16.0, 36.0, 67.0, 55.0, 15.0, 46.0, 50.0, 17.0, 24.0,
13.0, 18.0, 18.0, 9.0, 45.0, 13.0, 10.0, 8.0, 10.0, 9.0, 10.0, 10.0, 10.0, 9.0, 10.0, 8.0, 9.0,
107.0, 21.0, 13.0, 19.0, 17.0, 16.0, 58.0, 43.0, 16.0, 33.0, 30.0, 13.0, 17.0, 15.0, 18.0, 30.0,
23.0, 20.0, 25.0, 13.0, 15.0, 12.0, 12.0, 12.0, 10.0, 11.0, 10.0, 59.0, 48.0, 12.0, 9.0, 15.0, 41.0,
36.0, 19.0, 50.0, 23.0, 84.0, 60.0, 44.0, 20.0, 15.0, 19.0, 12.0, 70.0, 21.0, 14.0, 25.0, 38.0,
139.0, 85.0, 36.0, 13.0, 19.0, 22.0, 60.0, 307.0, 79.0, 66.0, 350.0, 202.0, 241.0, 45.0, 36.0, 35.0,
62.0, 64.0, 26.0, 48.0, 91.0, 121.0, 77.0, 34.0, 292.0, 83.0, 101.0, 38.0, 23.0, 26.0, 26.0, 26.0,
31.0, 28.0, 38.0, 31.0, 23.0, 23.0, 96.0, 41.0, 83.0, 30.0, 206.0, 62.0, 286.0, 99.0, 33.0, 52.0,
38.0, 64.0, 42.0, 240.0, 59.0, 51.0, 49.0, 54.0, 45.0, 32.0, 29.0, 21.0, 30.0, 53.0, 39.0, 446.0,
39.0, 218.0, 24.0, 28.0, 14.0, 13.0, 18.0, 20.0, 30.0, 97.0, 367.0, 189.0, 100.0, 24.0, 41.0, 102.0,
200.0, 90.0, 105.0, 109.0, 48.0, 189.0, 236.0, 29.0, 30.0, 354.0, 206.0, 400.0, 313.0, 421.0, 447.0,
437.0, 87.0, 192.0, 82.0, 227.0, 99.0, 134.0, 71.0, 200.0, 331.0, 381.0, 194.0, 314.0, 500.0, 149.0,
213.0, 182.0, 125.0, 197.0, 280.0, 277.0, 217.0, 168.0, 98.0, 34.0, 103.0, 119.0, 113.0, 35.0, 27.0,
247.0, 105.0, 200.0, 129.0, 146.0, 18.0, 20.0, 101.0, 25.0, 21.0, 29.0, 111.0, 42.0, 27.0, 143.0,
132.0, 41.0, 308.0, 500.0, 288.0, 269.0, 153.0, 135.0, 230.0, 399.0, 323.0, 386.0, 318.0, 209.0,
```

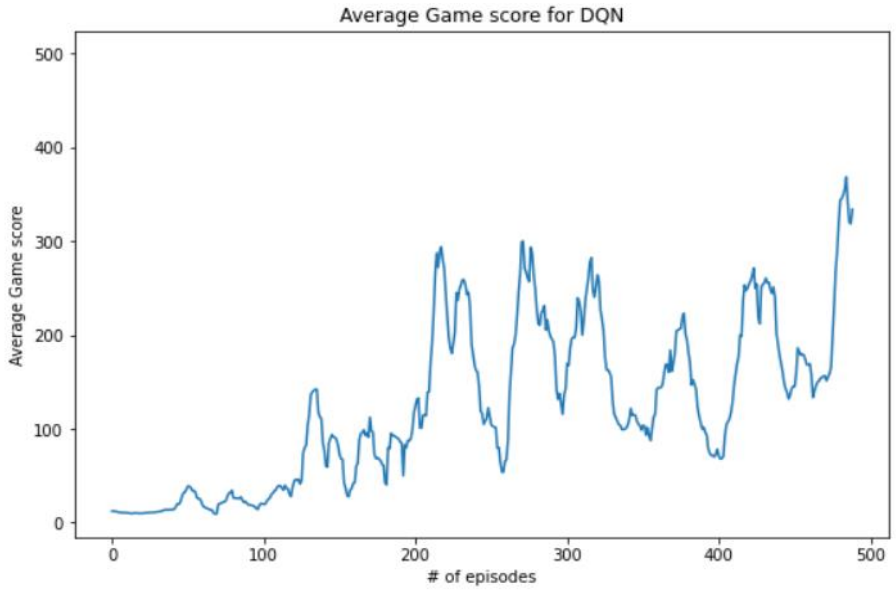
```
247.0, 206.0, 117.0, 500.0, 165.0, 164.0, 179.0, 166.0, 170.0, 191.0, 374.0, 238.0, 165.0, 243.0,
271.0, 46.0, 122.0, 136.0, 138.0, 39.0, 23.0, 132.0, 226.0, 138.0, 156.0, 249.0, 192.0, 403.0,
117.0, 224.0, 125.0, 143.0, 227.0, 241.0, 474.0, 211.0, 86.0, 152.0, 268.0, 443.0, 243.0, 264.0,
392.0, 292.0, 124.0, 140.0, 195.0, 280.0, 202.0, 133.0, 147.0, 139.0, 115.0, 154.0, 124.0, 109.0,
153.0, 25.0, 60.0, 97.0, 113.0, 97.0, 107.0, 111.0, 119.0, 116.0, 161.0, 61.0, 121.0, 211.0, 40.0,
102.0, 102.0, 48.0, 90.0, 97.0, 115.0, 111.0, 114.0, 113.0, 121.0, 16.0, 50.0, 168.0, 220.0, 124.0,
382.0, 130.0, 114.0, 118.0, 155.0, 134.0, 143.0, 166.0, 136.0, 361.0, 156.0, 213.0, 225.0, 352.0,
166.0, 145.0, 150.0, 286.0, 178.0, 141.0, 91.0, 75.0, 122.0, 113.0, 220.0, 99.0, 97.0, 89.0, 88.0,
73.0, 24.0, 92.0, 71.0, 80.0, 79.0, 47.0, 76.0, 83.0, 76.0, 102.0, 76.0, 19.0, 41.0, 85.0, 102.0,
267.0, 194.0, 112.0, 108.0, 170.0, 181.0, 202.0, 157.0, 200.0, 175.0, 500.0, 180.0, 500.0, 268.0,
112.0, 208.0, 249.0, 185.0, 260.0, 251.0, 287.0, 223.0, 141.0, 206.0, 500.0, 236.0, 270.0, 233.0,
212.0, 254.0, 209.0, 182.0, 209.0, 101.0, 113.0, 120.0, 155.0, 144.0, 124.0, 137.0, 142.0, 142.0,
141.0, 142.0, 182.0, 143.0, 154.0, 283.0, 396.0, 99.0, 104.0, 150.0, 132.0, 97.0, 125.0, 143.0,
165.0, 166.0, 153.0, 166.0, 159.0, 192.0, 140.0, 132.0, 129.0, 161.0, 165.0, 116.0, 194.0, 201.0,
230.0, 500.0, 483.0, 500.0, 347.0, 453.0, 414.0, 133.0, 236.0, 258.0, 361.0, 228.0, 270.0, 487.0,
500.0]
```



3. Average score graph: This graph captures the average reward collected by the agent for every `self.reward_average_episodes`.

```
Average Game score for DQN:  [12.1, 12.2, 12.1, 11.6, 11.2, 10.9, 10.5, 10.6, 10.7, 10.3, 10.4,
10.2, 9.8, 9.8, 10.0, 10.1, 10.1, 10.1, 9.8, 9.9, 10.1, 10.0, 10.4, 10.5, 10.5, 10.6, 10.8, 10.9,
10.9, 11.3, 11.4, 11.8, 11.8, 12.5, 13.3, 13.7, 13.6, 13.6, 13.7, 13.7, 13.9, 14.2, 16.8, 19.8,
19.5, 21.7, 27.3, 31.6, 32.1, 35.3, 38.7, 38.7, 37.2, 33.9, 34.1, 32.3, 26.5, 25.5, 25.3, 21.7,
17.5, 16.8, 15.3, 15.0, 14.2, 13.4, 13.4, 9.9, 9.4, 9.3, 19.2, 20.3, 20.7, 21.6, 22.3, 22.9, 27.8,
31.1, 31.9, 34.3, 26.6, 25.8, 26.2, 25.8, 25.9, 27.3, 23.8, 21.5, 22.4, 20.4, 18.9, 18.8, 18.3,
18.0, 17.2, 15.3, 14.0, 17.9, 20.2, 20.1, 19.5, 19.8, 22.7, 25.1, 26.0, 29.9, 31.2, 33.7, 34.9,
38.1, 39.2, 39.2, 37.0, 34.6, 39.7, 36.8, 35.9, 30.0, 27.8, 37.3, 43.8, 45.9, 45.3, 46.0, 41.2,
45.1, 74.4, 79.8, 82.6, 103.7, 115.4, 135.9, 139.1, 140.8, 142.1, 142.3, 118.0, 112.7, 110.9, 85.0,
76.9, 60.5, 59.4, 85.0, 89.8, 93.7, 91.1, 90.8, 88.6, 82.1, 72.6, 68.0, 67.4, 42.0, 36.8, 29.0,
27.5, 34.8, 36.3, 42.0, 42.4, 59.9, 63.3, 88.1, 94.9, 95.9, 98.8, 93.0, 95.3, 91.2, 112.2, 97.5,
96.4, 72.7, 68.2, 69.4, 67.4, 66.5, 62.2, 61.0, 42.3, 40.3, 79.8, 78.8, 95.2, 93.1, 92.7, 91.2,
90.4, 89.2, 85.9, 85.0, 50.1, 82.9, 80.0, 87.6, 87.2, 89.9, 98.8, 117.0, 124.0, 131.5, 132.7, 100.8,
100.8, 114.4, 114.9, 113.8, 139.0, 139.6, 170.6, 191.4, 222.6, 262.5, 287.3, 272.4, 288.7, 293.9,
281.2, 270.5, 243.9, 219.7, 197.6, 186.0, 180.4, 191.1, 203.3, 245.1, 237.3, 248.7, 253.5, 258.9,
258.6, 253.5, 243.1, 245.4, 230.8, 190.6, 179.1, 168.1, 161.8, 160.6, 144.4, 119.1, 116.1, 104.9,
108.1, 111.2, 122.4, 113.9, 104.0, 102.8, 101.8, 101.2, 79.4, 80.0, 64.2, 54.0, 53.7, 65.1, 67.2,
87.9, 135.4, 162.1, 186.1, 190.3, 199.6, 219.9, 245.5, 264.6, 299.1, 300.1, 271.0, 266.9, 260.6,
257.0, 293.5, 287.0, 263.5, 249.1, 227.1, 212.3, 210.5, 223.2, 226.4, 231.2, 205.5, 216.1, 204.3,
198.6, 195.6, 192.4, 177.2, 142.1, 131.5, 137.6, 127.1, 115.6, 135.9, 142.9, 169.6, 167.5, 186.0,
196.2, 197.3, 197.4, 207.7, 239.5, 235.7, 225.1, 200.0, 215.1, 237.0, 248.8, 260.9, 277.4, 282.5,
247.5, 240.4, 251.3, 264.1, 257.5, 226.5, 216.9, 204.4, 176.7, 162.9, 162.9, 159.8, 155.6, 130.1,
115.9, 112.3, 108.9, 104.7, 103.9, 99.6, 99.1, 99.8, 100.6, 104.2, 110.3, 121.7, 114.4, 114.9,
114.4, 108.1, 105.2, 103.3, 98.7, 103.7, 103.0, 93.2, 101.3, 92.7, 87.5, 99.5, 112.5, 115.2, 141.9,
143.8, 143.8, 144.3, 147.7, 159.5, 168.8, 168.6, 160.2, 183.9, 161.3, 169.6, 180.7, 204.1, 205.2,
206.3, 207.0, 219.0, 223.2, 201.2, 194.7, 180.9, 170.6, 146.7, 152.1, 147.5, 142.2, 122.5, 113.5,
```

```
106.7, 100.0, 101.7, 96.6, 93.3, 79.2, 74.0, 71.9, 71.3, 70.1, 73.0, 78.2, 70.9, 67.9, 68.4, 70.7,
92.7, 104.5, 107.4, 110.6, 117.4, 127.9, 146.2, 157.8, 169.3, 176.6, 199.9, 198.5, 237.3, 253.3,
247.5, 250.2, 254.9, 257.7, 263.7, 271.3, 250.0, 254.3, 218.4, 212.2, 251.0, 253.8, 255.9, 260.7,
255.9, 256.2, 248.4, 244.3, 251.1, 240.6, 201.9, 190.3, 178.8, 169.9, 161.1, 149.4, 142.7, 138.7,
131.9, 136.0, 142.9, 145.2, 145.1, 159.0, 186.2, 182.4, 178.6, 179.4, 178.5, 174.0, 168.3, 168.3,
169.4, 157.7, 133.4, 140.1, 145.6, 149.8, 150.6, 154.1, 154.5, 156.3, 156.3, 151.3, 155.4, 158.9,
166.0, 196.8, 231.1, 267.9, 289.7, 318.9, 343.8, 345.5, 349.7, 355.4, 368.5, 341.3, 320.0, 318.7,
334.0]
```



Part 2 - Improving DQN:

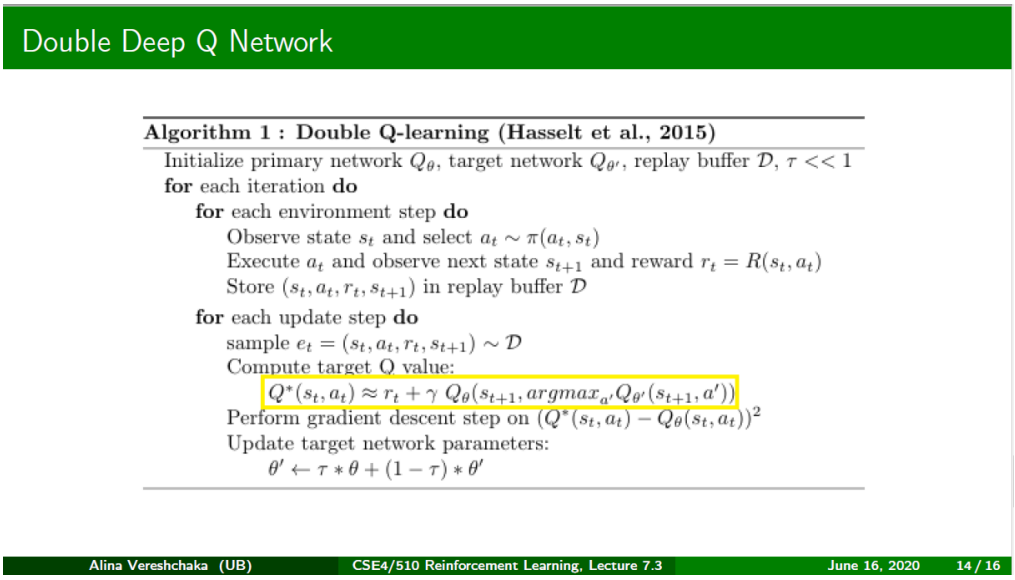
DeepMind also observed that DQN suffers from overestimations for some games in the Atari 2600 domain so they suggested improvements in the form of Double DQN (DDQN) algorithm. For this section, I have implemented DDQN algorithm which is based on using two estimators – one for obtaining the best action and the other for evaluating state-action value Q for evaluating the selected action. The basic idea for DDQN is that it is less likely for both estimators to overestimate at the same time.

DDQN Algorithm:

Implementation details:

I have implemented two classes:

- `DDQNAgent`: It is the agent class where all the main logic to implement DQN algorithm resides. and in similar to `DQNAgent` class.
- `dEnvironment`: It is the class which prepares the CartPole-v1 environment using gym's environment and uses `DDQNAgent` as agent to solve this environment. It is analogous to `Environment` class.



The algorithm for DDQN is very similar to DQN and the only change is where we select the best action and evaluate the state-action value Q:

DQN	DDQN
`Q_prime = max(self.target_model.predict(state prime)[0])`	`best_action = np.argmax(self.model.predict(state_prime)[0])`
`state_target[0][action] = reward + self.gamma*Q_prime`	`Q_prime = self.target_model.predict(state_prime)[0][best_action]`
	`state_target[0][action] = reward + self.gamma*Q_prime`
`self.target_model.set_weights(self.model.get_weights())`	`target weights[i] = model weight*self.tau + target_weight * (1-self.tau)`
	`self.target model.set weights(target weights)`

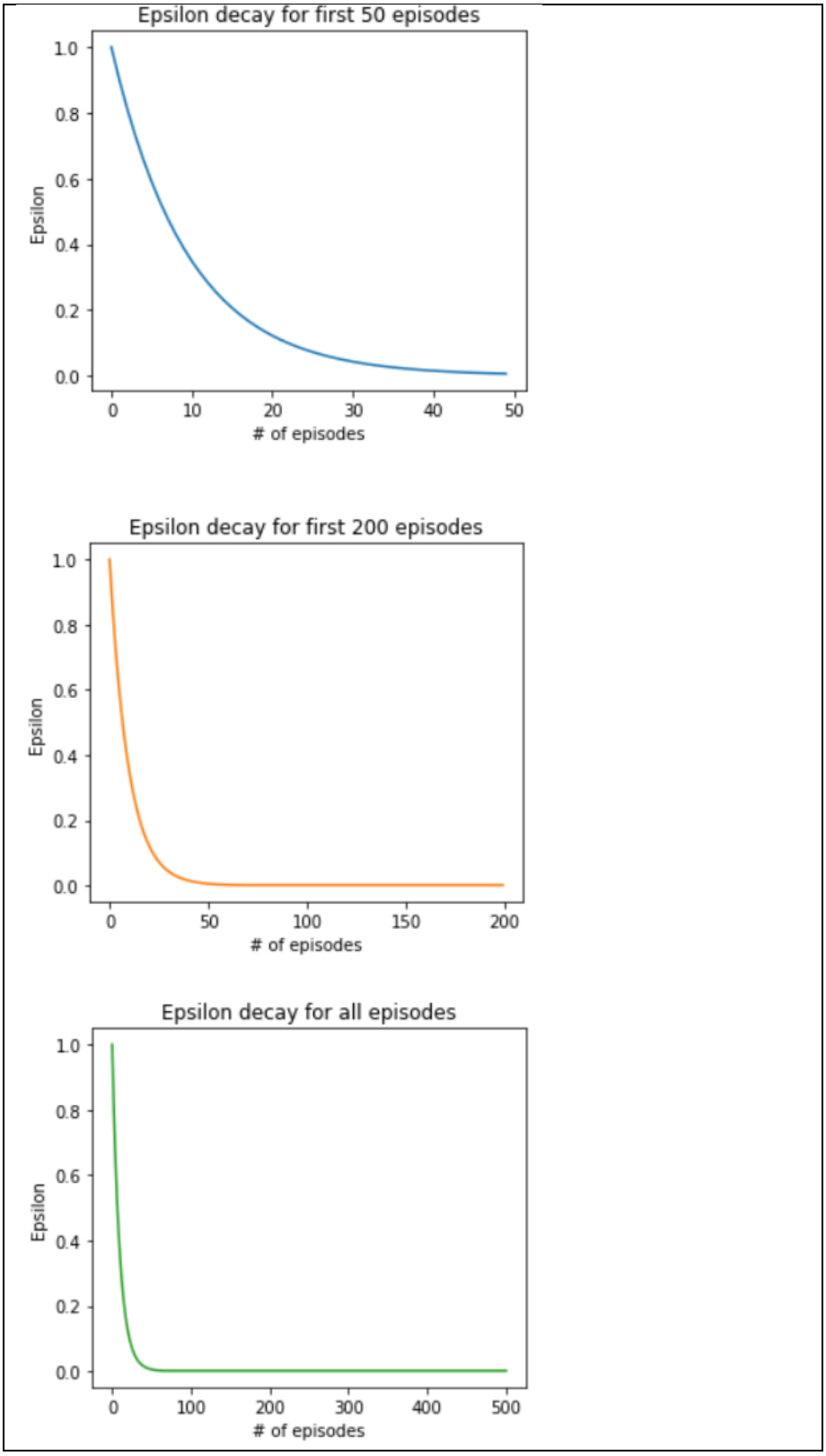
Results:

The results are captured with following graphs by keeping all hyper parameters same as that for DQN except for tau:

```
self.tau = 0.99
```

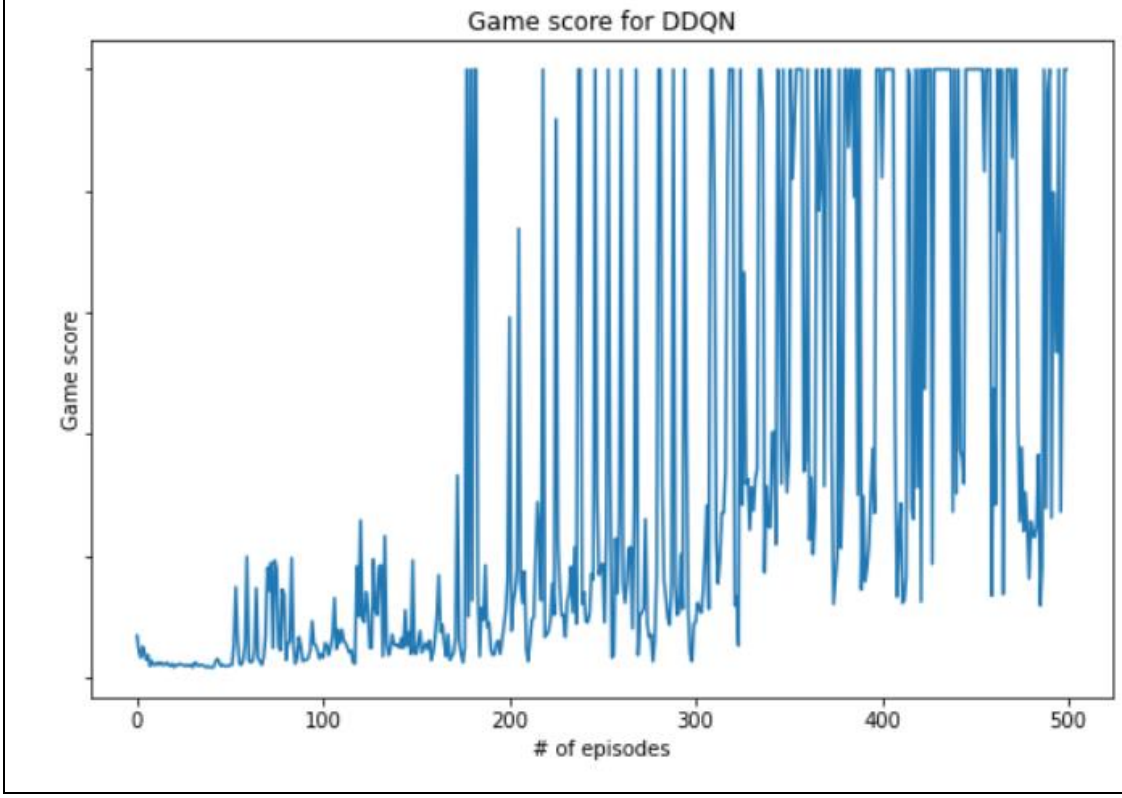
1. **Epsilon decay graph**: Here I have shown the variation of epsilon value with the number of episodes encountered. To illustrate it better, I have captured 3 graphs of epsilon-decay with first

50 episodes then with first 200 episodes and finally with all episodes. Since the parameters used for epsilon decay are same as that for DQN so the two graphs are identical.



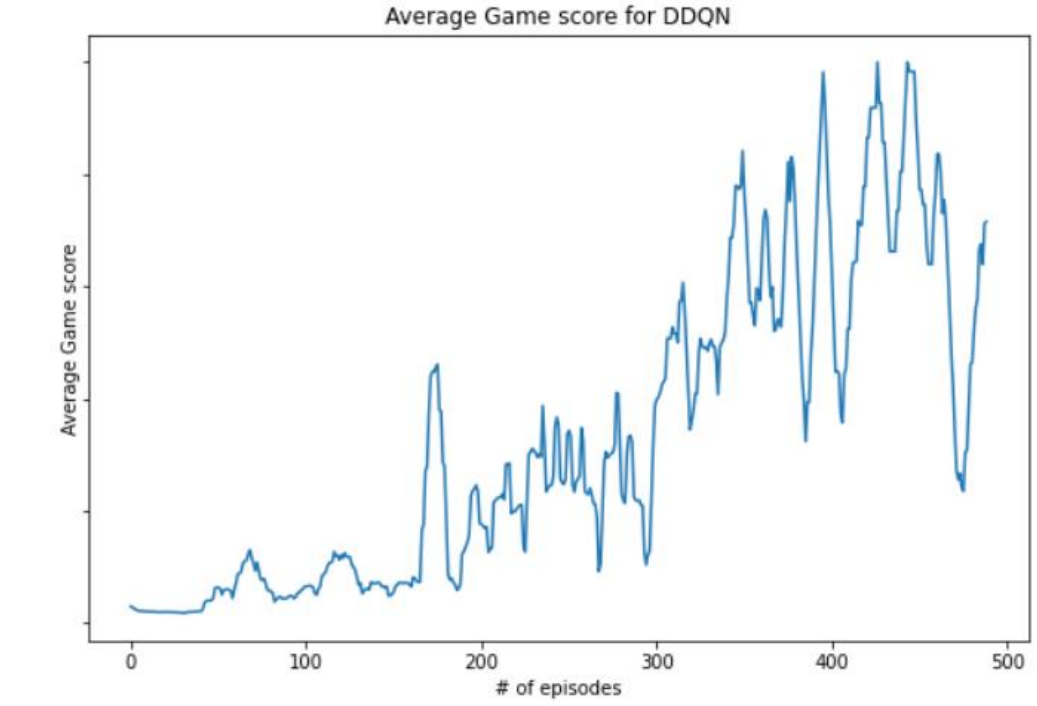
1. **Game score graph**: This graph captures the reward collected by agent for each episode.

```
Game score for DDQN:  [34.0, 21.0, 16.0, 25.0, 20.0, 14.0, 18.0, 9.0, 13.0, 10.0, 10.0, 12.0, 10.0,
12.0, 10.0, 10.0, 12.0, 10.0, 9.0, 11.0, 8.0, 10.0, 10.0, 11.0, 10.0, 10.0, 9.0, 10.0, 9.0, 10.0,
8.0, 12.0, 11.0, 9.0, 10.0, 10.0, 9.0, 8.0, 9.0, 8.0, 8.0, 8.0, 12.0, 15.0, 13.0, 9.0, 10.0, 9.0,
9.0, 9.0, 10.0, 10.0, 27.0, 74.0, 31.0, 11.0, 10.0, 14.0, 27.0, 99.0, 14.0, 12.0, 13.0, 22.0, 73.0,
15.0, 13.0, 10.0, 16.0, 34.0, 90.0, 71.0, 94.0, 24.0, 96.0, 89.0, 25.0, 22.0, 72.0, 68.0, 14.0,
28.0, 29.0, 98.0, 26.0, 11.0, 16.0, 33.0, 25.0, 13.0, 14.0, 14.0, 16.0, 22.0, 46.0, 27.0, 26.0,
21.0, 15.0, 19.0, 16.0, 28.0, 26.0, 18.0, 25.0, 34.0, 65.0, 23.0, 39.0, 28.0, 39.0, 30.0, 28.0,
24.0, 20.0, 22.0, 12.0, 11.0, 91.0, 50.0, 129.0, 48.0, 45.0, 70.0, 58.0, 24.0, 24.0, 97.0, 55.0,
51.0, 91.0, 92.0, 17.0, 116.0, 28.0, 18.0, 23.0, 35.0, 27.0, 26.0, 27.0, 24.0, 32.0, 24.0, 55.0,
26.0, 37.0, 19.0, 96.0, 20.0, 19.0, 29.0, 38.0, 20.0, 24.0, 28.0, 23.0, 30.0, 14.0, 20.0, 32.0,
51.0, 84.0, 37.0, 43.0, 27.0, 17.0, 35.0, 14.0, 17.0, 20.0, 32.0, 166.0, 31.0, 19.0, 12.0, 24.0,
500.0, 50.0, 500.0, 63.0, 500.0, 500.0, 72.0, 17.0, 57.0, 47.0, 92.0, 41.0, 45.0, 23.0, 18.0, 19.0,
28.0, 30.0, 19.0, 30.0, 42.0, 55.0, 81.0, 296.0, 38.0, 61.0, 71.0, 87.0, 369.0, 79.0, 61.0, 87.0,
23.0, 13.0, 30.0, 47.0, 49.0, 99.0, 144.0, 113.0, 63.0, 500.0, 33.0, 35.0, 38.0, 48.0, 77.0, 51.0,
459.0, 111.0, 74.0, 51.0, 51.0, 33.0, 62.0, 69.0, 91.0, 53.0, 107.0, 44.0, 500.0, 500.0, 60.0, 70.0,
46.0, 45.0, 52.0, 85.0, 80.0, 500.0, 152.0, 84.0, 91.0, 93.0, 45.0, 119.0, 500.0, 171.0, 16.0, 19.0,
114.0, 69.0, 131.0, 500.0, 77.0, 61.0, 77.0, 106.0, 107.0, 40.0, 142.0, 500.0, 18.0, 50.0, 53.0,
56.0, 130.0, 44.0, 33.0, 35.0, 13.0, 30.0, 82.0, 500.0, 500.0, 159.0, 79.0, 65.0, 48.0, 43.0, 77.0,
500.0, 76.0, 51.0, 53.0, 101.0, 56.0, 500.0, 205.0, 53.0, 19.0, 13.0, 44.0, 45.0, 61.0, 56.0, 53.0,
72.0, 104.0, 141.0, 56.0, 500.0, 500.0, 399.0, 112.0, 77.0, 98.0, 135.0, 135.0, 171.0, 410.0, 500.0,
500.0, 500.0, 59.0, 66.0, 26.0, 500.0, 142.0, 333.0, 159.0, 163.0, 121.0, 156.0, 137.0, 163.0,
171.0, 500.0, 500.0, 465.0, 86.0, 157.0, 124.0, 123.0, 201.0, 202.0, 109.0, 500.0, 379.0, 159.0,
500.0, 199.0, 152.0, 183.0, 500.0, 410.0, 453.0, 500.0, 500.0, 500.0, 500.0, 169.0, 179.0, 500.0,
113.0, 164.0, 101.0, 132.0, 500.0, 384.0, 412.0, 500.0, 157.0, 414.0, 500.0, 500.0, 185.0, 60.0,
80.0, 100.0, 500.0, 106.0, 174.0, 500.0, 500.0, 436.0, 500.0, 500.0, 395.0, 500.0, 150.0, 500.0,
72.0, 149.0, 79.0, 90.0, 109.0, 149.0, 188.0, 135.0, 500.0, 500.0, 500.0, 411.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 213.0, 66.0, 112.0, 143.0, 61.0, 63.0, 83.0, 500.0, 492.0, 138.0, 130.0,
500.0, 156.0, 500.0, 62.0, 500.0, 237.0, 500.0, 500.0, 500.0, 93.0, 500.0, 500.0, 500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 136.0, 500.0, 151.0, 500.0, 187.0, 180.0, 159.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 416.0, 500.0, 500.0, 500.0, 67.0,
237.0, 142.0, 500.0, 367.0, 500.0, 68.0, 319.0, 500.0, 500.0, 500.0, 427.0, 500.0, 500.0, 210.0,
128.0, 189.0, 120.0, 152.0, 125.0, 81.0, 128.0, 117.0, 115.0, 125.0, 183.0, 59.0, 91.0, 500.0,
139.0, 482.0, 500.0, 131.0, 399.0, 322.0, 267.0, 500.0, 136.0, 323.0, 500.0, 500.0]
```



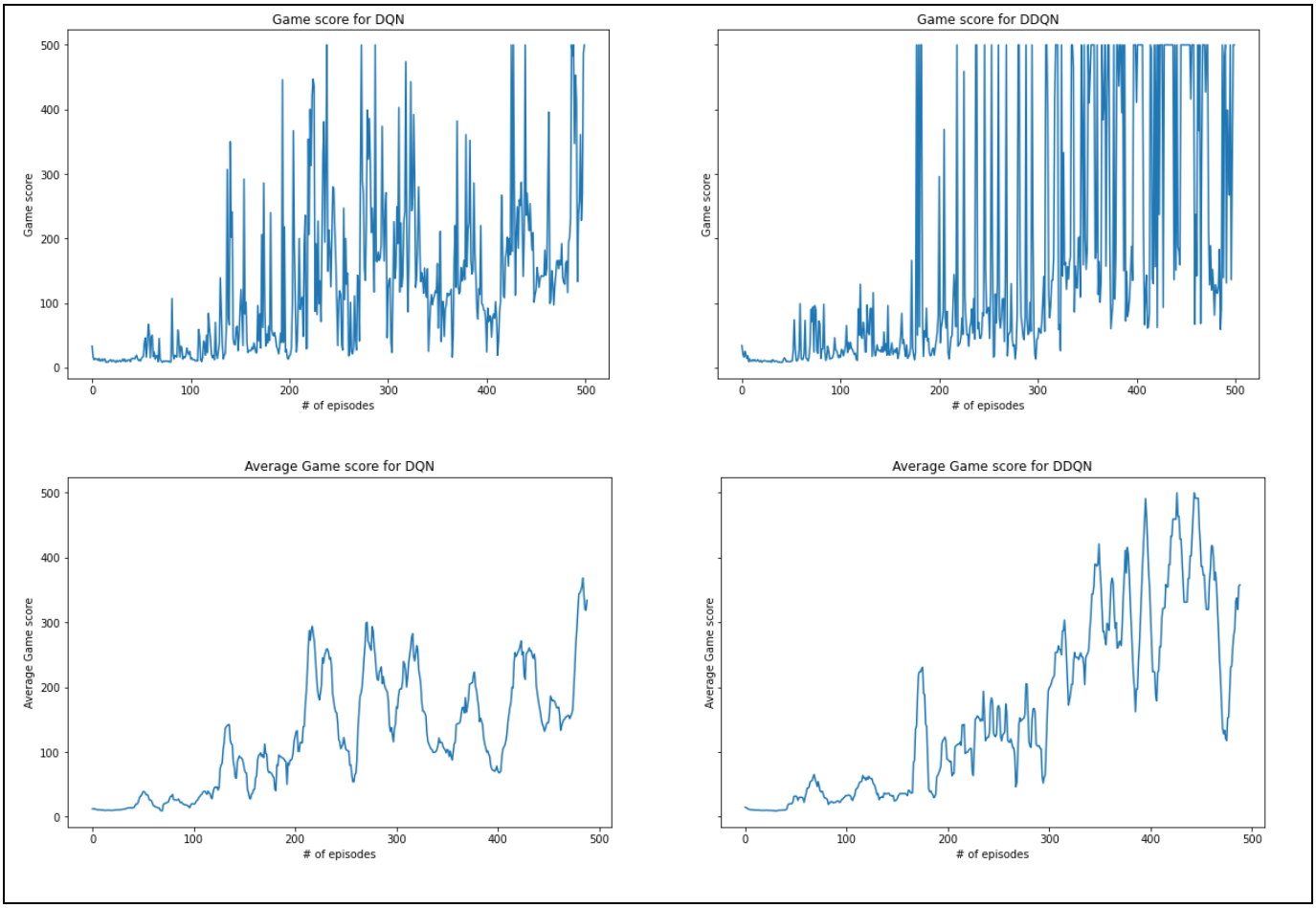
2. **Average score graph**: This graph captures the average reward collected by the agent for every `self.reward_average_episodes`.

```
Average Game score for DDQN:  [14.7, 14.1, 12.8, 11.8, 11.4, 10.8, 10.9, 10.5, 10.6, 10.4, 10.2,
10.2, 10.1, 10.1, 10.1, 9.8, 9.8, 9.8, 9.7, 9.7, 9.9, 10.0, 9.8, 9.8, 9.8, 9.8, 9.6, 9.6, 9.4, 9.4,
9.0, 9.1, 9.7, 10.0, 9.9, 10.0, 10.1, 10.1, 10.2, 10.4, 10.6, 12.1, 18.0, 19.8, 20.0, 20.0, 20.5,
22.3, 31.3, 31.7, 31.9, 30.5, 25.3, 29.5, 29.9, 30.2, 29.8, 28.7, 22.2, 29.8, 35.7, 43.8, 44.0,
46.3, 53.7, 54.9, 56.1, 61.7, 65.1, 57.5, 53.2, 46.7, 54.1, 47.1, 39.3, 38.4, 39.5, 34.8, 29.3,
29.3, 27.9, 26.6, 19.0, 21.0, 22.6, 23.6, 22.4, 21.4, 22.0, 22.2, 23.6, 24.6, 24.2, 22.1, 22.8,
26.7, 26.9, 29.3, 30.2, 32.5, 32.7, 32.9, 33.5, 33.0, 31.8, 26.5, 25.3, 30.5, 32.7, 41.7, 43.5,
45.2, 49.8, 53.6, 53.8, 55.0, 63.6, 60.0, 60.1, 56.3, 60.7, 57.9, 62.5, 59.5, 58.9, 58.8, 52.6,
49.8, 47.3, 40.9, 34.1, 35.6, 26.4, 29.1, 29.9, 31.3, 29.7, 36.6, 36.0, 35.2, 35.7, 36.3, 35.9,
32.8, 33.0, 31.6, 32.7, 24.5, 24.5, 25.8, 28.0, 32.6, 34.3, 36.2, 36.1, 35.5, 36.0, 36.0, 35.7,
34.5, 32.6, 40.8, 40.2, 37.8, 36.3, 37.0, 83.5, 87.1, 135.4, 139.7, 186.5, 219.9, 224.0, 223.8,
228.3, 230.6, 189.8, 188.9, 143.4, 139.4, 91.2, 43.1, 38.7, 40.0, 36.2, 34.5, 29.5, 30.9, 34.5,
61.8, 63.8, 68.0, 72.3, 78.0, 113.0, 117.9, 119.8, 123.0, 117.2, 88.9, 88.1, 86.7, 84.5, 85.7, 63.2,
66.6, 66.8, 108.1, 109.1, 111.3, 112.1, 112.2, 115.0, 110.2, 141.7, 141.5, 142.6, 97.7, 99.5, 99.3,
101.7, 103.8, 105.2, 105.4, 70.2, 63.5, 106.1, 151.0, 151.9, 155.6, 154.0, 151.6, 147.7, 150.9,
148.2, 193.8, 159.0, 117.4, 120.5, 122.8, 122.7, 130.1, 174.9, 183.5, 177.1, 129.0, 125.2, 123.7,
127.7, 168.4, 171.6, 165.8, 123.5, 117.0, 126.1, 128.2, 131.0, 174.1, 162.8, 117.8, 115.4, 114.9,
120.2, 114.0, 106.6, 106.1, 93.2, 46.2, 52.6, 97.6, 142.3, 152.6, 147.5, 149.6, 151.1, 151.9, 158.3,
205.3, 204.7, 159.8, 115.1, 109.3, 107.0, 150.5, 166.2, 167.2, 161.4, 112.7, 109.5, 108.9, 109.7,
105.2, 104.9, 62.1, 52.0, 60.8, 64.5, 113.2, 158.8, 194.2, 199.3, 201.4, 205.9, 212.2, 215.3, 218.3,
253.7, 253.7, 253.7, 263.8, 258.5, 257.4, 250.2, 286.7, 287.4, 303.6, 278.5, 244.8, 206.9, 172.5,
180.3, 190.0, 204.5, 204.5, 240.3, 253.5, 246.2, 245.6, 245.9, 242.6, 249.0, 252.9, 246.7, 246.7,
234.6, 204.0, 245.4, 249.6, 252.4, 258.4, 288.3, 309.1, 343.5, 343.5, 355.6, 389.7, 389.7, 386.7,
389.4, 421.1, 382.4, 357.8, 322.6, 285.8, 285.8, 274.2, 265.4, 298.5, 296.3, 287.7, 326.4, 360.0,
368.4, 361.2, 319.2, 290.8, 299.6, 260.2, 261.9, 270.5, 270.5, 264.1, 295.6, 339.6, 371.1, 411.1,
376.1, 415.5, 405.3, 370.2, 328.1, 293.5, 254.4, 219.3, 198.6, 162.1, 197.1, 197.1, 239.9, 266.1,
308.2, 349.2, 388.3, 423.4, 454.6, 491.1, 462.4, 419.0, 380.2, 353.4, 309.5, 265.8, 224.1, 224.1,
223.3, 187.1, 178.8, 222.2, 226.6, 262.3, 262.4, 306.1, 321.5, 321.5, 322.3, 358.5, 354.8, 354.8,
389.2, 389.2, 433.0, 433.0, 459.3, 459.3, 459.3, 459.3, 500.0, 463.6, 463.6, 428.7, 428.7, 397.4,
365.4, 331.3, 331.3, 331.3, 331.3, 367.7, 367.7, 402.6, 402.6, 433.9, 465.9, 500.0, 491.6, 491.6,
491.6, 491.6, 448.3, 422.0, 386.2, 386.2, 372.9, 372.9, 338.1, 320.0, 320.0, 320.0, 363.3, 382.3,
418.1, 418.1, 402.4, 365.2, 377.3, 357.4, 322.6, 285.1, 243.2, 213.3, 175.0, 136.5, 128.0, 133.5,
120.5, 117.6, 152.4, 153.8, 193.9, 231.1, 232.5, 260.9, 280.6, 289.0, 333.1, 337.6, 319.9, 356.0,
357.8]
```



Comparison of DQN and DDQN performance:

Next, we study the comparative performance of DQN and DDQN for the same initial set of hyperparameters:



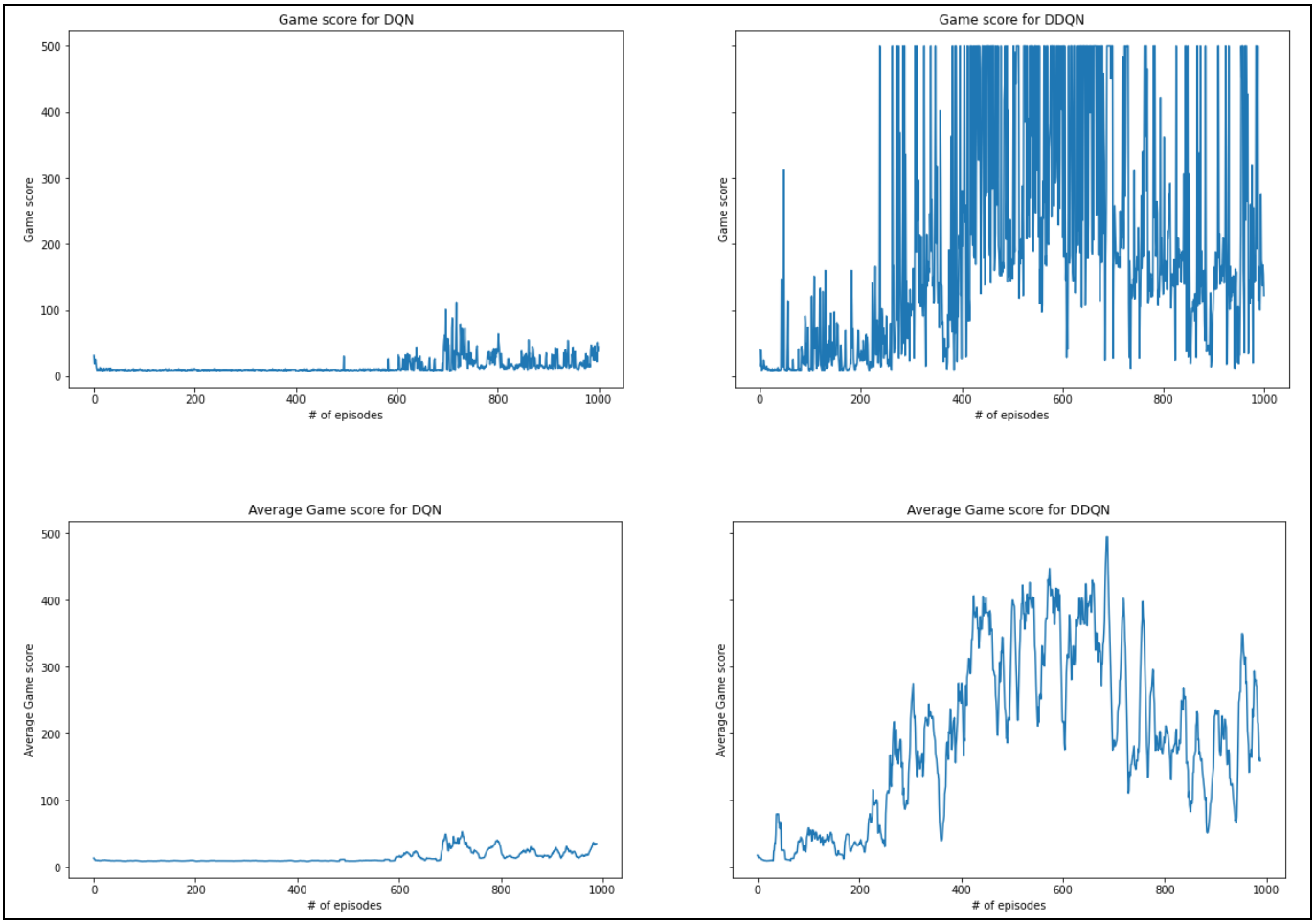
As we can see that keeping other hyper-parameters constant, normally DDQN performs better than DQN for the total and average scores collected. It can also be seen from the graphs that DDQN starts performing better earlier (fast learner) than DQN (slow learner) over the same set of episodes. So, we can say that DDQN learns better policy quickly as compared to DQN.

Effect of other parameters on performance of DQN and DDQN:

In this section, I have tried to show the effect of modifying some of the hyperparameters on the game score and average score calculated for DQN and DDQN.

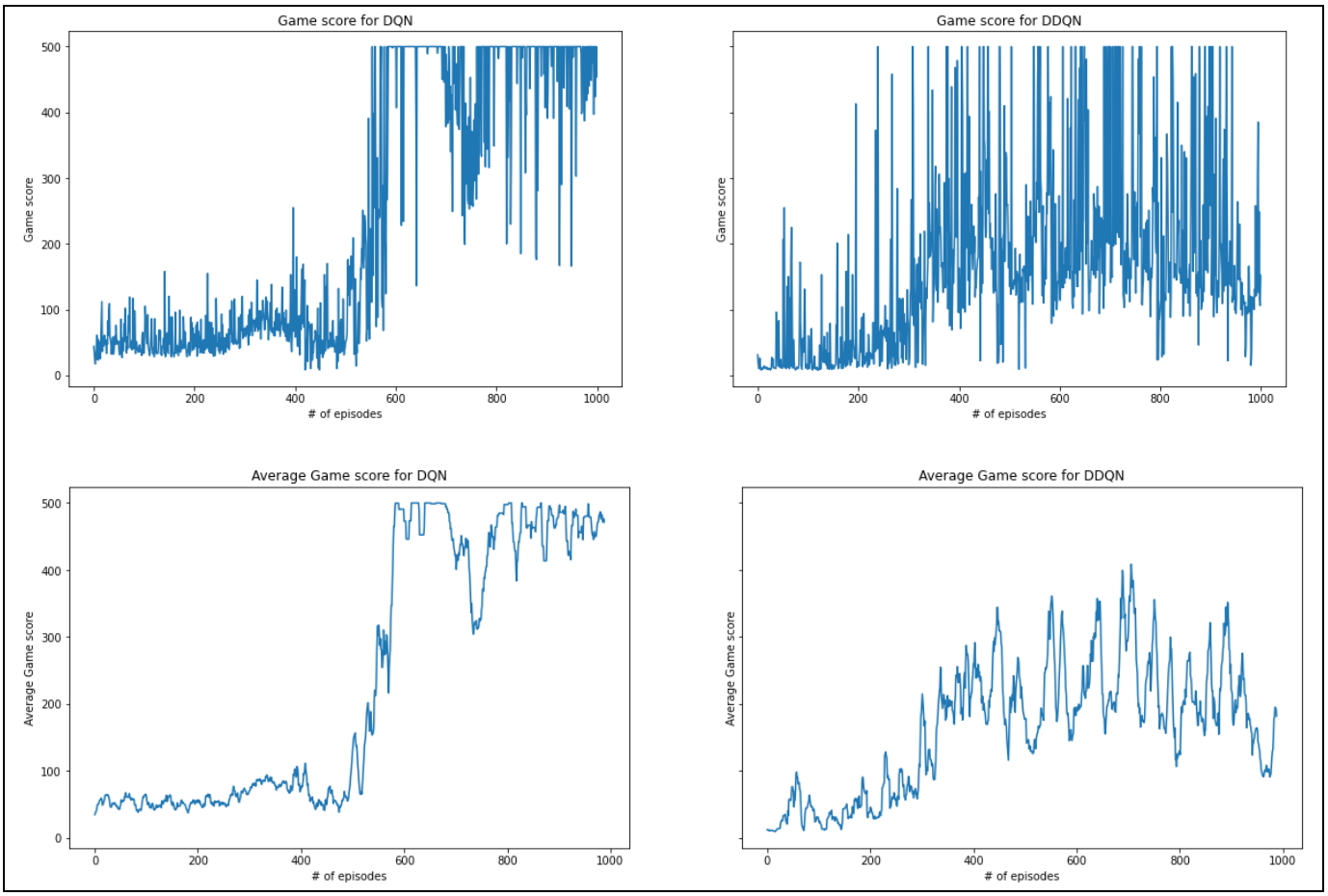
1.
```
self.number_of_episodes = 1000
self.C = 10
```

Here I have kept the sync parameter C to 10 and observed the behavior for 1000 episodes. As we can see in this case, DQN does not perform very well but DDQN performs very well. It can be due to the slow learning of DQN as compared to DDQN as the number of weight sync is reduced.



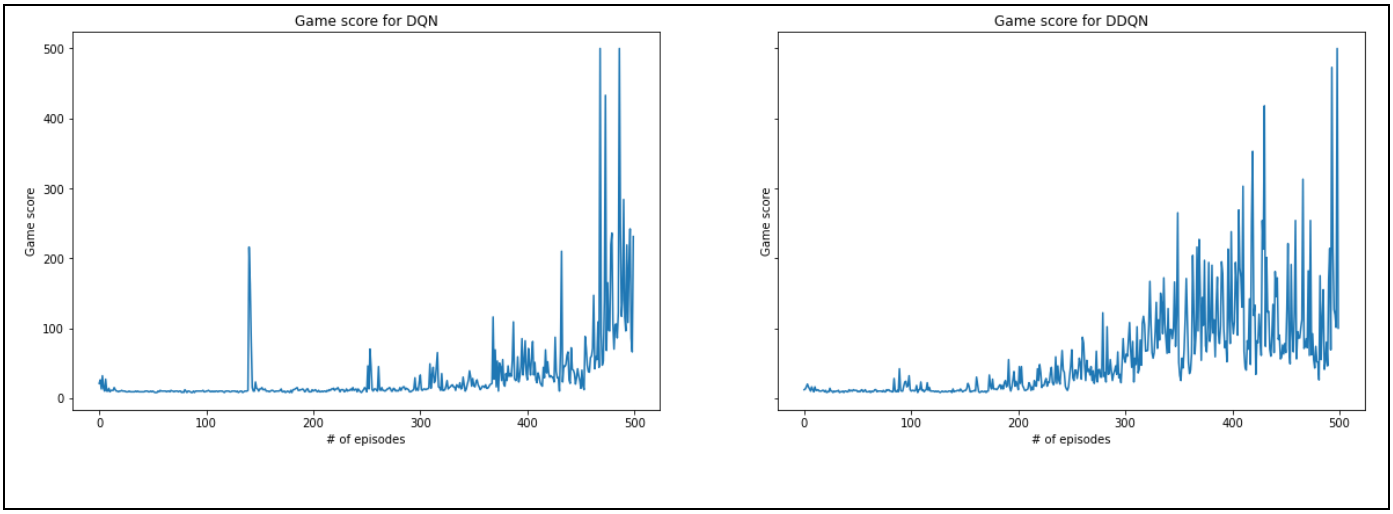
2.
```
self.number_of_episodes = 1000
self.C = 5
```

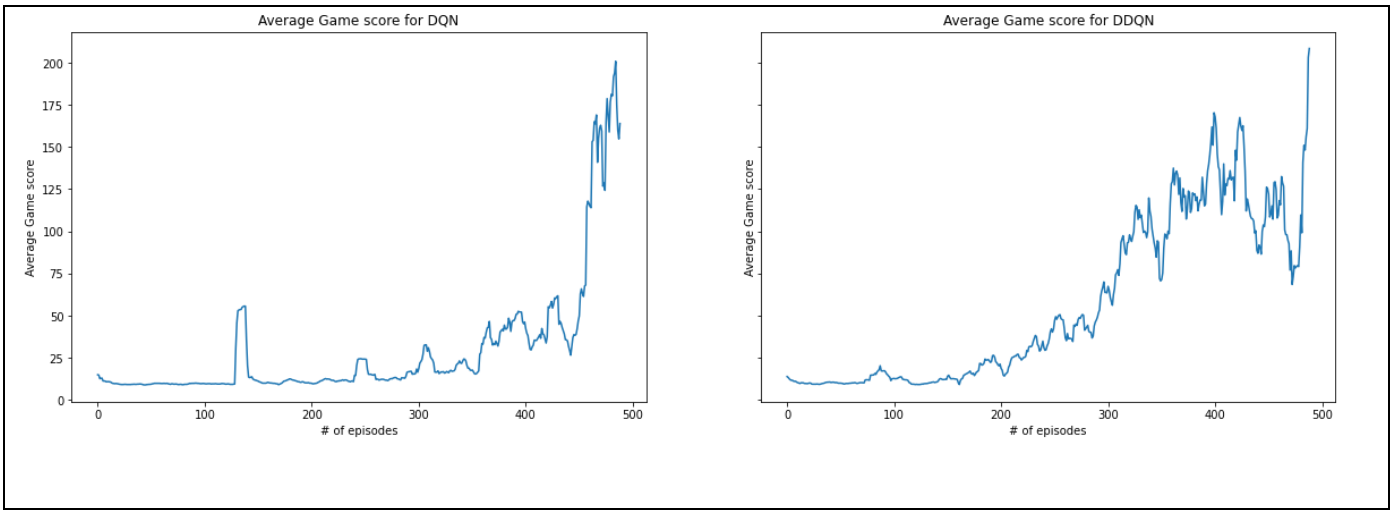
Here I have kept the sync parameter C to 5 and observed the behavior for 1000 episodes. As we can see in this case, DQN starts performing well later than DDQN but it picks up towards the later episodes.



3. `self.number_of_episodes = 500`
   `self.C = 5`

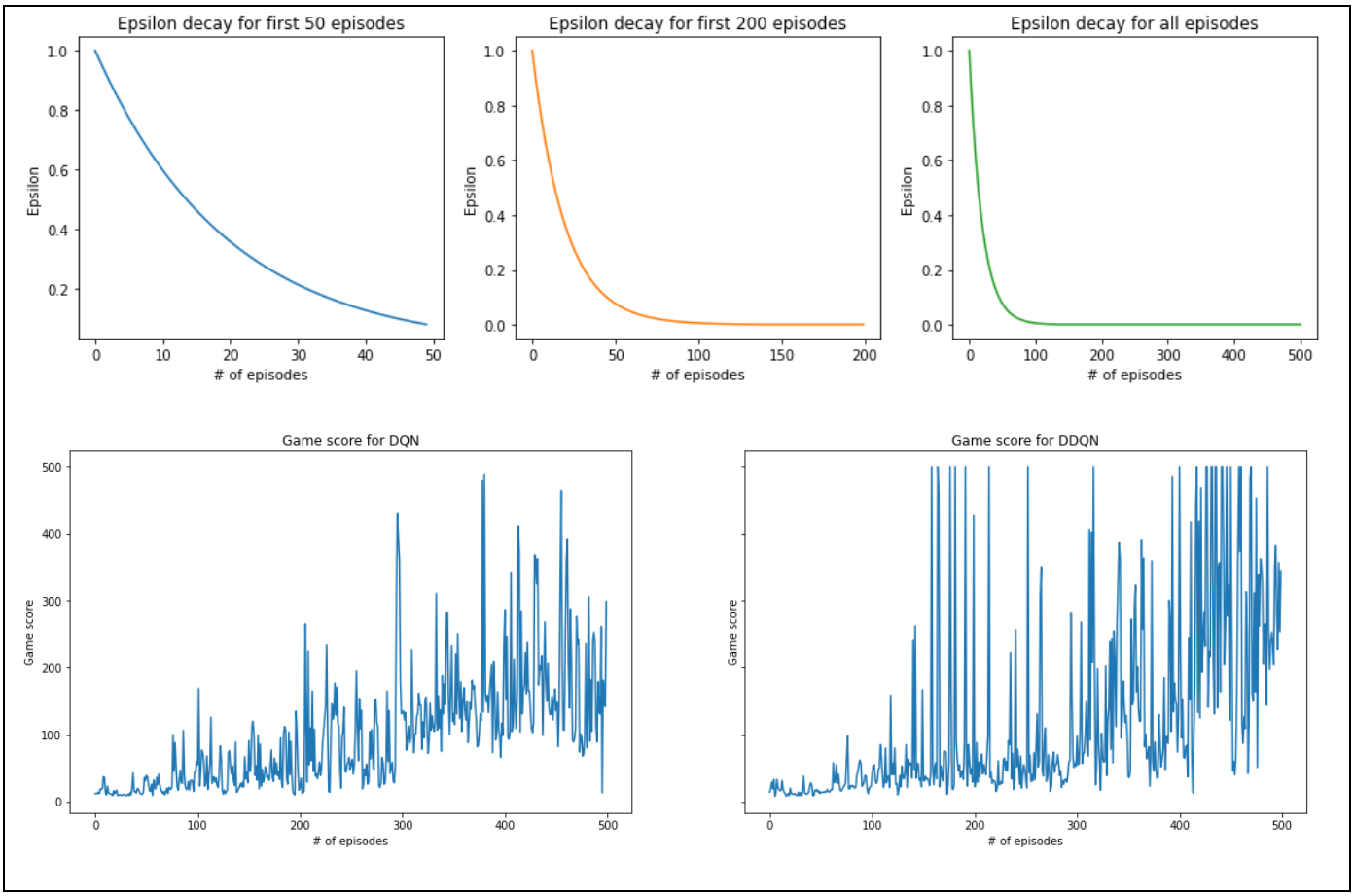
Here I have kept the sync parameter C to 5 and observed the behavior for 500 episodes. As we can see in this case, DQN does not perform very well but DDQN performs relatively well. We can again see the slow learning of DQN as compared to DDQN from these results.

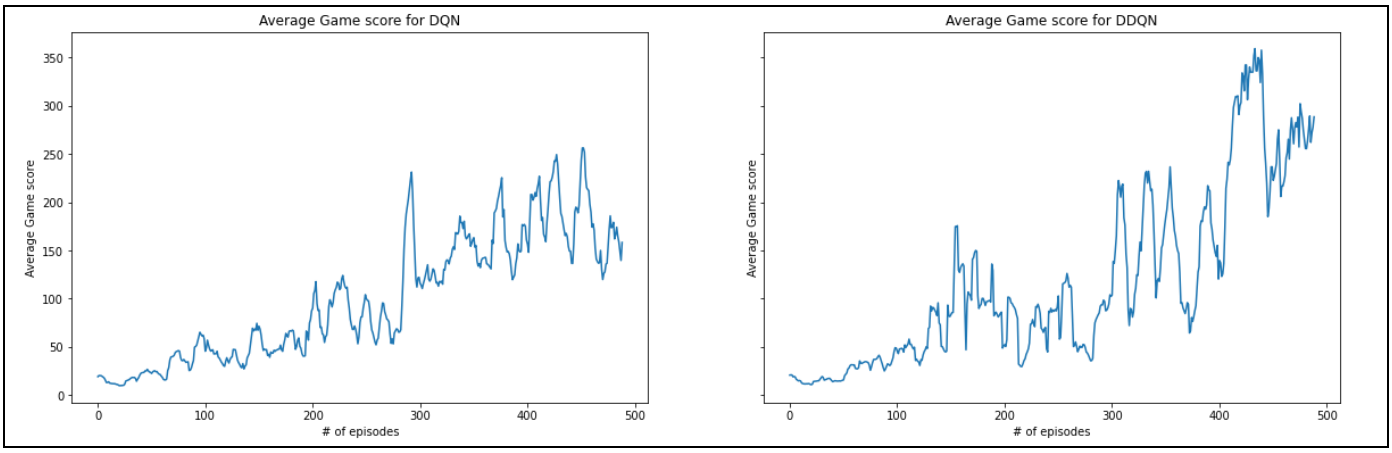





```
4. self.epsilon_decay = 0.95
   self.C = 1
```

Here I have kept the epsilon_decay parameter to 0.95 and observed the behavior for 500 episodes. As we can see in this case, epsilon decay happens relatively slower and DQN and DDQN both perform relatively well for later episodes. We can again see the slow learning of DQN as compared to DDQN from these results.





During different trials I have also observed that the results vary even with same set of parameters. It can be due to the neural network training, random minibatch selection and also different server configurations at runtime.

Conclusion:

From the results obtained, we can see that both DQN and DDQN perform poor initially but as the number of episodes increases they learn from the experiences and start performing better as the number of episodes increases. In comparison, DDQN learns faster and starts performing better soon as compared to DQN. Towards the end, we also saw the effects of different hyper parameter on the performance of both DQN and DDQN.