CPSC 8810: Deep Reinforcement Learning

Project 2

Question 1

Architecture:

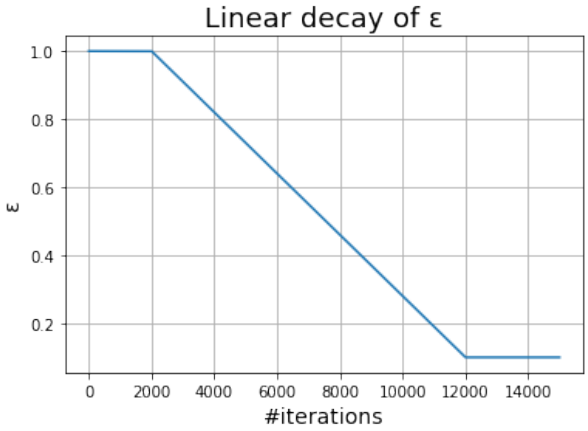
The neural network (NN) employed to solve the problem has three hidden layers with the first two layers having 60 neurons each and the final hidden layer having 30 neurons. As the number of inputs and outputs of the NN are 61 and 3 respectively, the resultant architecture of the NN is 61-60-60-30-3.

Training:

The NN is trained for a total of four seeds – 6, 8, 10 and 12. The values of various parameters that are used during the training process are as follows:

a. Batch size – 32
b. Epochs – 1
c. Learning rate α – $1x10^{-4}$
d. Discount factor γ – 0.93
e. Maximum steps (or experiences) – 30000
f. Weight update frequency C – every 100 time-steps
g. Experience-replay buffer size – 30000

The strategy followed to explore the state-action space is not pure epsilon-greedy. Instead, it is a variant of epsilon-greedy in which the value of ε remains equal to 1 for the first 2000 time-steps. This strategy ensures that the agent sufficiently explores the state-action space before it gradually starts acting greedily with respect to the predicted action-value functions. The following plot is a sample illustration of how ε decays as the #time-steps increases.



Results:

To analyze the performance of the NN for different seeds, two sets of roll outs are performed. While the first set of roll outs happen at the end of each episode, the second set happens after every 500 time-steps. The following are the entities that are extracted from the results of these roll outs:

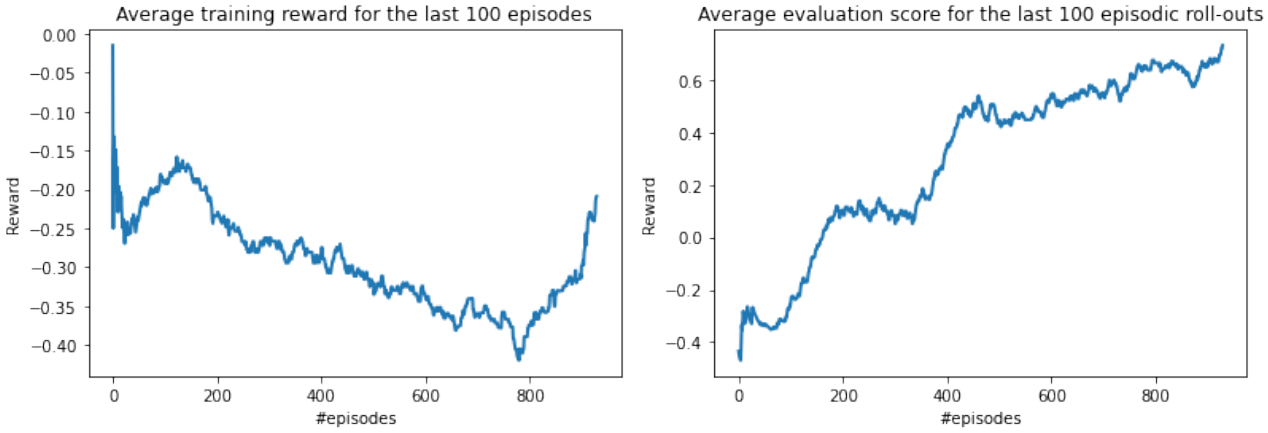
a. Mean evaluation reward for the last 100 episodic roll outs μ_{RO100}
b. Standard deviation of the evaluation rewards for the last 100 episodic roll outs σ_{RO100}
c. Mean evaluation reward of all time-step-based roll outs μ_{RO}
d. Standard deviation of the evaluation rewards of all time-step-based roll outs σ_{RO}

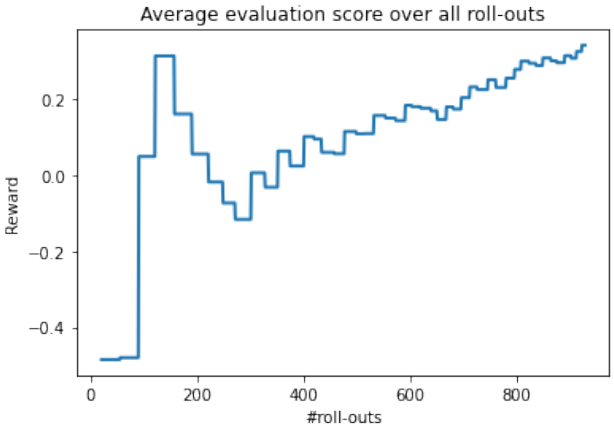
The values of these entities for different seeds are tabulated in the following table:

Seed	μ_{RO100}	σ_{RO100}	μ_{RO}	σ_{RO}
6	0.73	0.52	0.34	0.66
8	0.53	0.56	0.08	0.50
10	0.60	0.53	0.2	0.65
12	0.42	0.54	0.42	0.63
Overall results	0.57 ± 0.11	0.537 ± 0.01	0.26 ± 0.13	0.61 ± 0.06

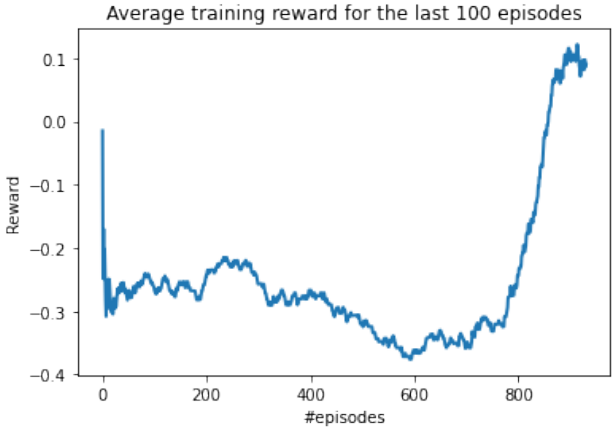
It is evident from the results that the NN achieves its best performance when the seed is 6. The plots of the results for all four seeds are given below:

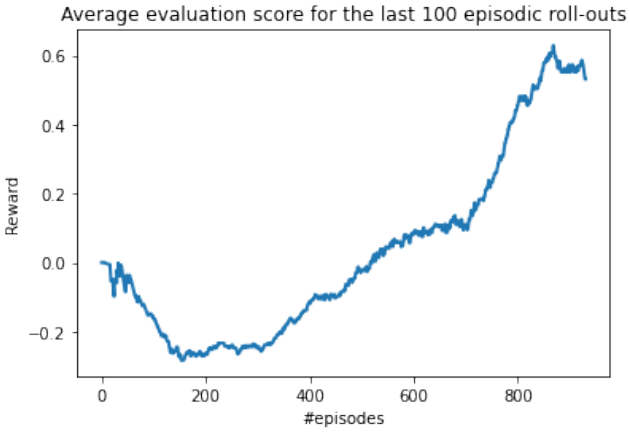
a. Seed 6:

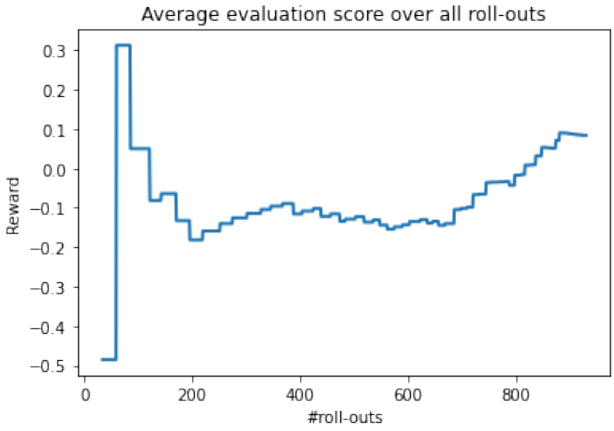



Average evaluation score over all roll-outs
Reward
0.2
0.0
−0.2
−0.4
0
200
400
600
800
#roll-outs

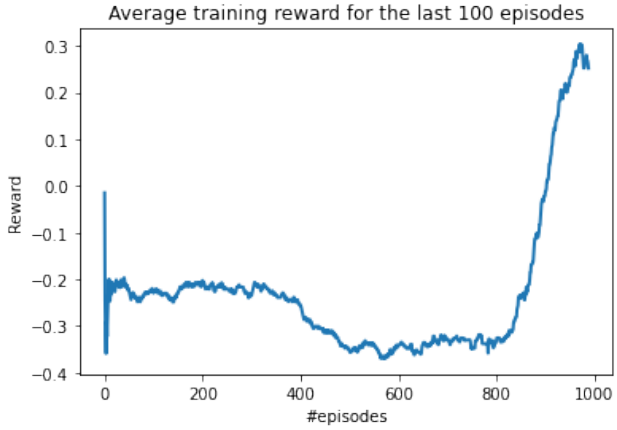
b. Seed 8:

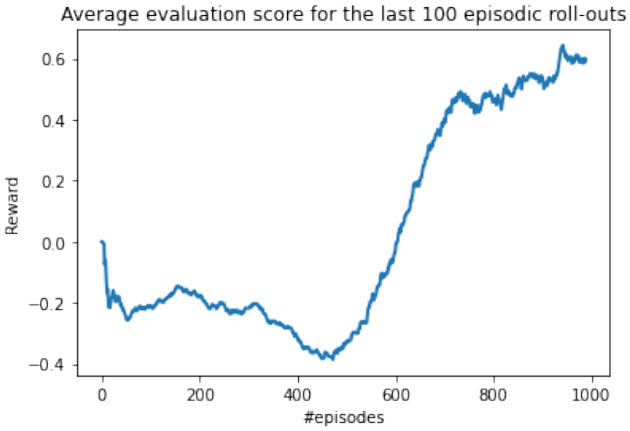

Average training reward for the last 100 episodes
Reward
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
200
400
600
800
#episodes

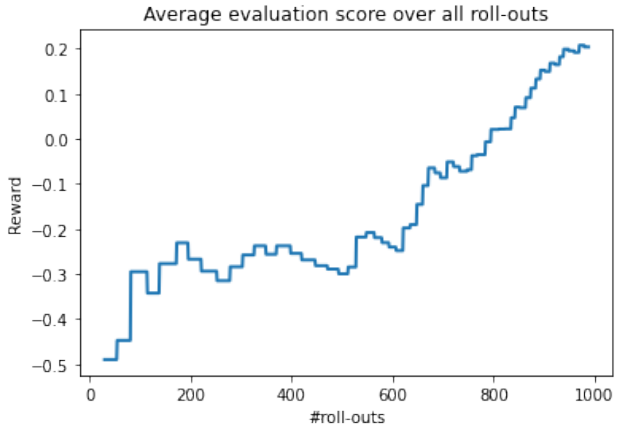

Average evaluation score for the last 100 episodic roll-outs
Reward
0.6
0.4
0.2
0.0
−0.2
0
200
400
600
800
#episodes


Average evaluation score over all roll-outs
Reward
#roll-outs

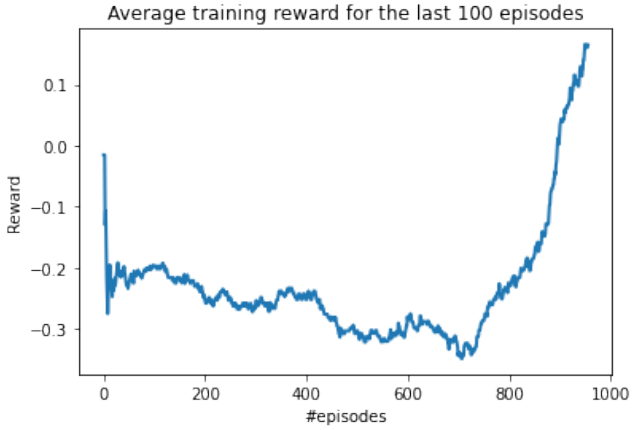
c. Seed 10:

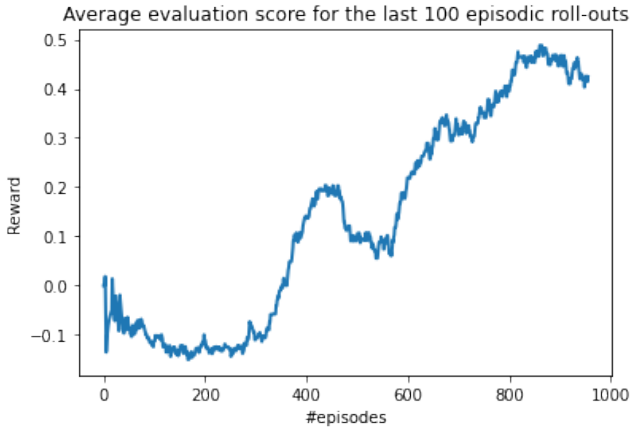

Average training reward for the last 100 episodes
Reward
#episodes

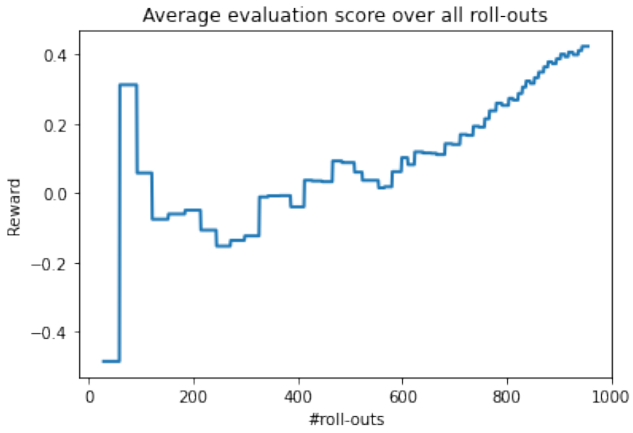

Average evaluation score for the last 100 episodic roll-outs
Reward
#episodes


Average evaluation score over all roll-outs
Reward
#roll-outs
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
0
200
400
600
800
1000

d. Seed 12:


Average training reward for the last 100 episodes
Reward
#episodes
0.1
0.0
−0.1
−0.2
−0.3
0
200
400
600
800
1000


Average evaluation score for the last 100 episodic roll-outs
Reward
#episodes
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
0
200
400
600
800
1000



Observations:

I witnessed during the initial rounds of training that most seeds have a peak evaluation score. Once the score hits the peak value, it stays so only for a short while. After this, the score will start dropping. To prevent this from happening and maximize the reward, I incorporated a crude form of early stopping with the code. The effect of early stopping is visible in the plot of the average evaluation for the last 100 roll outs.

I am aware that the architecture I used is not the best one since it could be possible to get a higher reward if a larger number of neurons is used in the hidden layers, especially the final hidden layer. The downside of that would be an increased risk of the network overfitting the experiences samples.

If the network architecture is modified to have only two hidden layers, the rewards would rarely cross 0.3. This is probably because the network is not non-linear enough to learn the complexities of the frogger environment. Moreover, it is highly likely that the resultant policy would cause the agent to get stuck in the middle of the road and vibrate back and forth until the episode terminates due to time constraints.

Question 2

Architecture:

The neural network (NN) employed to solve the problem has three hidden layers with the first two layers having 120 neurons each and the final layer having 90 neurons. The network is clearly larger than the one used in Question 1. This is because the agent's ability to move sideways in the environment 'frogger-v1' makes this problem harder to solve than the previous one. As the number of inputs and outputs of the NN are 123 and 5 respectively, the resultant architecture of the NN is 123-120-120-90-5.

Training:

The NN is trained for a total of four seeds – 6, 8, 10 and 12. The values of various parameters that are used during the training process are as follows:

a. Batch size – 64
b. Epochs – 1
c. Learning rate α – $1x10^{-4}$
d. Discount factor γ – 0.93
e. Maximum steps (or experiences) – 30000
f. Weight update frequency C – every 100 time-steps
g. Experience-replay buffer size – 30000

As opposed to the agent in Question 1, the agent in this question explores the state-action space by following the traditional epsilon-greedy strategy.

Results:

The performance of the NN is analyzed for different seeds by executing two sets of roll outs. While the first set is executed at the end of each episode, the second is executed after every 500 time-steps. The following are the entities that are extracted from the results of these roll outs:

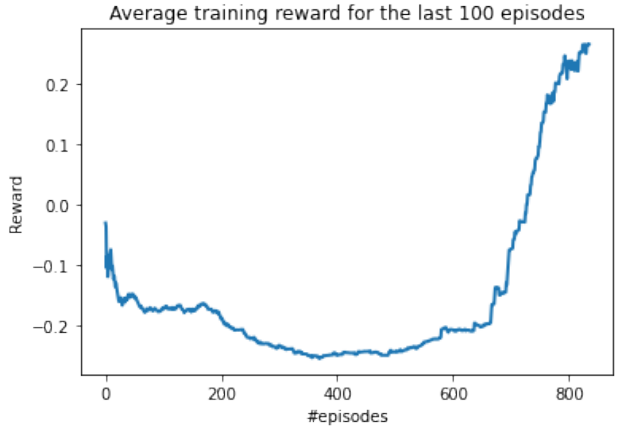
a. Mean evaluation reward for the last 100 episodic roll outs μ_{RO100}
b. Standard deviation of the evaluation rewards for the last 100 episodic roll outs σ_{RO100}
c. Mean evaluation reward of all time-step-based roll outs μ_{RO}
d. Standard deviation of the evaluation rewards of all time-step-based roll outs σ_{RO}

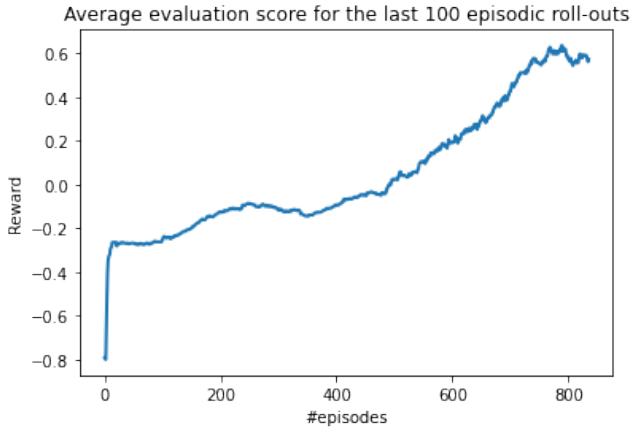
The values of these entities for different seeds are tabulated in the following table:

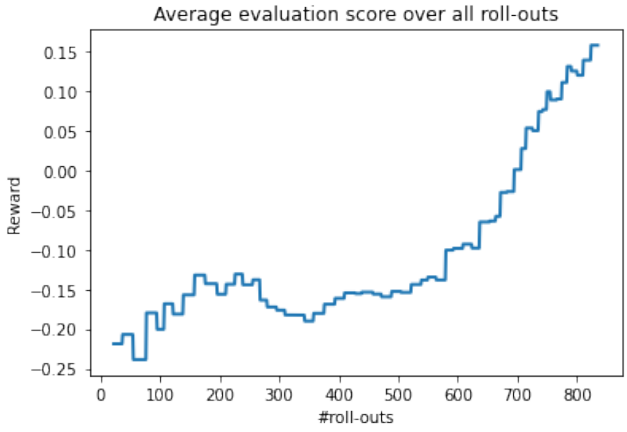
Seed	μ_{RO100}	σ_{RO100}	μ_{RO}	σ_{RO}
6	0.57	0.68	0.16	0.56
8	0.54	0.66	0.24	0.59
10	0.54	0.61	0.22	0.61
12	0.54	0.65	0.18	0.54
Overall results	0.547 ± 0.01	0.65 ± 0.02	0.2 ± 0.03	0.575 ± 0.02

It is evident from the results that the NN achieves its best performance when the seed is 6. The plots of the results for all four seeds are given below:

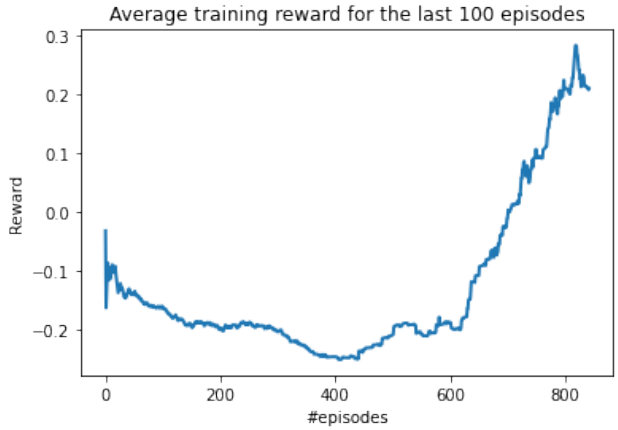
a. Seed 6:

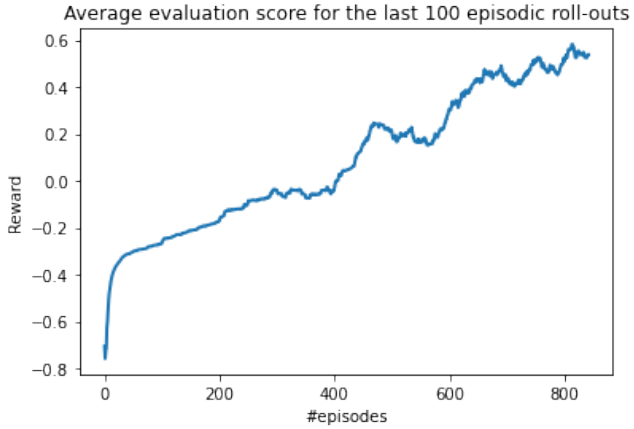


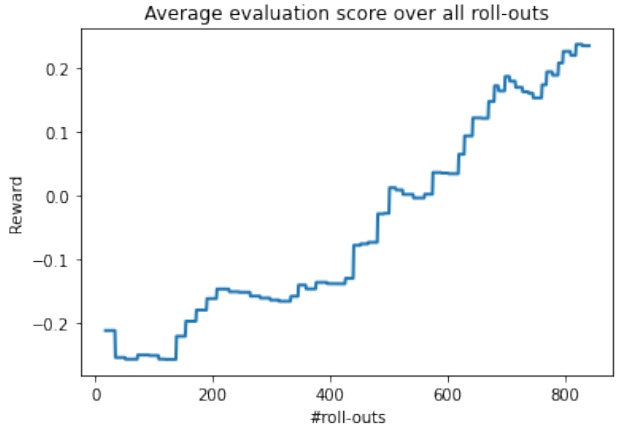




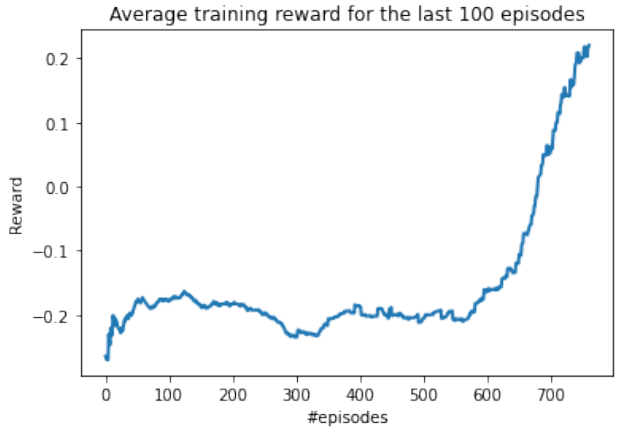
b. Seed 8:

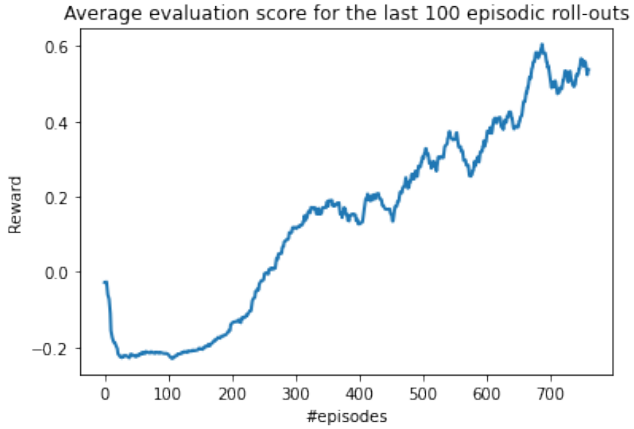

Average training reward for the last 100 episodes
Reward
#episodes
0.3
0.2
0.1
0.0
−0.1
−0.2
0
200
400
600
800

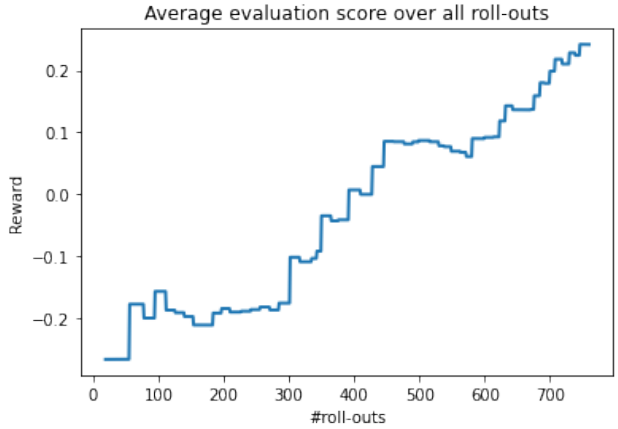

Average evaluation score for the last 100 episodic roll-outs
Reward
#episodes
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
200
400
600
800


Average evaluation score over all roll-outs
Reward
#roll-outs
0.2
0.1
0.0
−0.1
−0.2
0
200
400
600
800

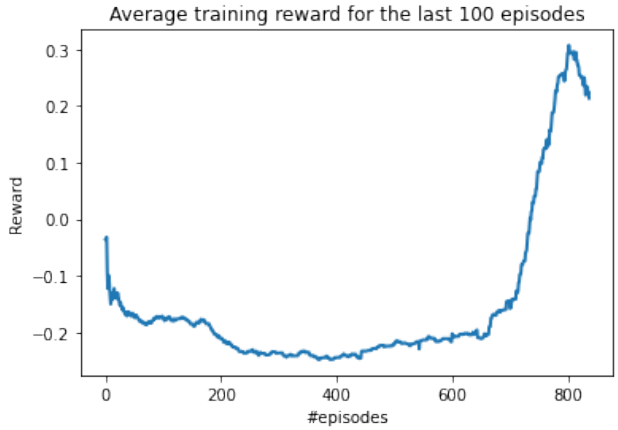
c. Seed 10:

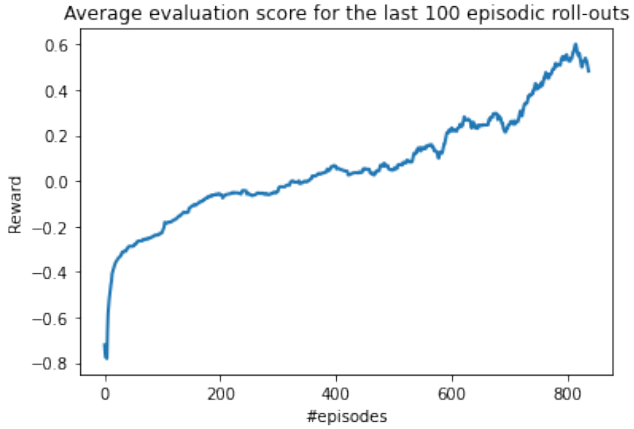

Average training reward for the last 100 episodes
Reward
#episodes
0.2
0.1
0.0
−0.1
−0.2
0
100
200
300
400
500
600
700

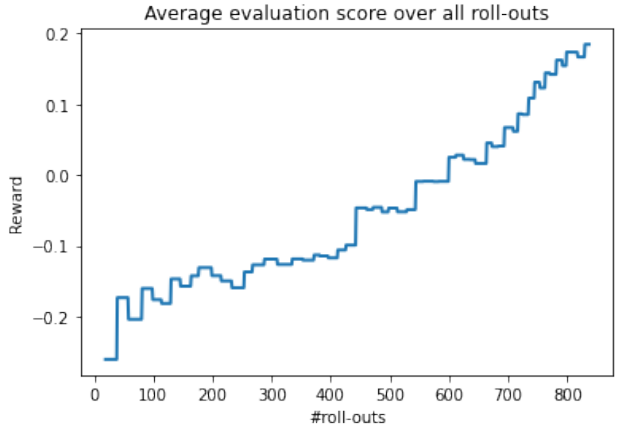

Average evaluation score for the last 100 episodic roll-outs
Reward
#episodes
0.6
0.4
0.2
0.0
−0.2
0
100
200
300
400
500
600
700


Average evaluation score over all roll-outs
Reward
#roll-outs
0.2
0.1
0.0
−0.1
−0.2
0
100
200
300
400
500
600
700

d. Seed 12:







Observations:

The following improvements have been observed in the performance of Double DQN when compared to DQN:

a. The mean evaluation reward takes only 6000 to 7000 time-steps to become positive. On the other hand, DQN takes upwards of 15000 time-steps to report its first positive value.
b. The variations in the mean evaluation reward in double DQN is not as bad as in DQN. Therefore, the results are more stable. This observation not only applies to a given seed but across all seeds.
c. It is much easier to reproduce the results of double DQN when compared to that of DQN.
d. The training time for Double DQN is almost the same as that of DQN even when the former uses a batch size that is double that of the former and employs a network that is much more complex. This shows that Double DQN can run faster than DQN.

It is observed that modifying the network architecture to reduce the number of hidden layers from three to two results in a huge dip in the average evaluation score. Moreover, the resultant policies often cause the agent to veer abruptly to the left and right without making any steps in the forward direction.