AE598 Reinforcement Learning : DQN

Manav Vora
Department of Aerospace Engineering
University of Illinois Urbana Champaign
Urbana, USA
mkvora2@illinois.edu

***Abstract*—In this assignment, we implemented the Deep Q-Network (DQN) algorithm on a simple pendulum problem with a continuous state space and discretized action space. We evaluated and compared the performance of four variants of the DQN algorithm. The difference between these variants was with respect to the update of the target network and the experience replay implementation. From simulations, we observe that difference in experience replay implementation affects the performance more significantly as compared to change in target network update. We also observe that DQN variants with experience replay produces the best results.**

***Index Terms*—deep reinforcement learning, DQN, neural network**

I. Introduction

An inverted pendulum represents many real-world systems like the Segway, human posture systems, launching of a rocket, and a bipedal self-balancing robot. Any system that requires vertical stabilization has dynamics that are similar to an inverted pendulum. However, a simple pendulum in the inverted position is an unstable system and hence a controller is required to maintain the inverted position.

In this assignment, we design a controller for maintaining a simple pendulum, with continuous state space and discrete action space, in the inverted position. The optimal control policy is obtained by implementing deep reinforcement learning. More specifically, we use the DQN algorithm [1]. DQN uses a neural network to approximate the Q-function and hence can be applied to systems with continuous state space. However, DQN still requires a finite action space.

We evaluate and compare four variants of the DQN algorithm for controlling the inverted pendulum. These algorithms differ in their implementations of the target network update and experience replay. Performing simulations on a simple pendulum environment, we observe that experience replay implementation has the most significant impact on the performance of the algorithm. Furthermore the simulation results show that DQN variants having experience replay outperform those without replay.

The rest of the report is organized as follows. Section II provides a brief overview of the DQN algorithm as given in [1]. Next, Section III consists of the simulation results obtained by implementing the variants of the DQN algorithm on the simple pendulum environment. Finally, Section IV presents the conclusions.

II. Method

We consider the scenario where an agent interacts with the simple pendulum environment by performing actions and obtaining rewards. The action performed by the agent causes the pendulum to transition to a new state. The goal of the agent is to maximize the future discounted reward starting from the current state. The optimal action-value function $Q^*(s,a)$ is the maximum expected return achievable by following any policy π, starting from a state s and taking action a at time instant t.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}\right],$$

where r_t denotes the reward obtained by the agent at time t and γ denotes the discount factor. The TD-update for the optimal action-value function is given by

$$Q^*(s,a) = r + \gamma \max_{a'} Q^*(s',a').$$

For the DQN algorithm, the action-value function is approximated using a Q-network, i.e., a neural network function approximator. More specifically, DQN uses two Q-networks. One for the approximate action-value function and the second for the target action-value function used in the TD-update. The weights for the approximation Q-network at iteration i are denoted by θ_i and those for the target Q-network are denoted by θ_i^-.

In this assignment, the loss function used is a mean-squared error (MSE) function. The gradient of the loss function, $\nabla_{\theta_i} L(\theta_i)$, is given by

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'}\Big[\Big(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\Big)\nabla_{\theta_i} Q(s,a;\theta_i)\Big].$$

Also for better convergence, the gradient is clipped such that its norm is always less than or equal to 1.

For this assignment, we use the RMSProp optimizer in PyTorch to optimize the weights θ_i of the neural network. The RMSProp optimizer has a learning rate of 0.00025, $\alpha = 0.95$, $\epsilon = 0.01$ and momentum = 0.95. In the standard DQN algorithm, the weights of the target network θ_i^- are updated and set equal to the weights θ_i after every k steps.

The Q-networks are trained using a technique called experience replay. This technique involves storing the agent's experiences at each time-step, into a replay memory. Each experience consists of a state, action, reward and next state. The weights of the Q-network, θ_i, are updated at the i^{th} iteration by randomly sampling a minibatch of experiences from the replay memory. This leads to greater data efficiency and reduces the variance of the weight updates. For this assignment, the replay memory is initialized to a size of 500 by populating it with experiences gained by the agent while operating on a random policy. This initialized replay memory is then used for learning.

In the no target variant of DQN, the weights of the target network are updated at each step and hence the target network is the same as the approximate action-value network. In the no replay variant of DQN, the size of the minibatch is the same as the size of the replay memory and hence the weights of the neural network are updated based on the current state, action, reward and next state. Now, we will discuss the results obtained by the different DQN variants.

For all the variants of the DQN, the exploration constant ϵ is initialized to a value of 1.0 and is linearly decreased upto a value of $\epsilon = 0.1$ over 10000 steps. This is done to ensure sufficient exploration and prevent convergence to a local optimum.

III. Simulation Results

The various hyperparameters used in the simulations are given in Table I. Since, the results vary from one run to the next, we analyze the average results for each algorithm over 10 runs. We will now look at the simulation results obtained using the different variants of DQN.

Hyperparameter	**Value**
Number of runs per algorithm	10
Number of episodes	200
Number of steps per episode	100
Learning rate	0.00025
Discount Factor γ	0.95
Initial ϵ	1.0
Minimum ϵ	0.1
Final Exploration Frame	14000
Batch size	32
Replay memory size	100000
Replay start size	500
Target network update frequency	1000

TABLE I
Hyperparameters

A. *Standard DQN*

The standard DQN algorithm was run 10 times for 200 episodes each. The mean return and the $1-\sigma$ bound for the return with respect to the episodes is shown in Figure 1. The mean return for the standard DQN algorithm is roughly 10 after the completion of the runs. As can be observed, the mean return increases with the number of episodes indicating that the agent is learning to stabilize the pendulum in the inverted position. This trend can be made clearer by increasing the number of episodes for each run. Also, we observe that there is an outlier around the 180 episode mark with a huge negative standard deviation.

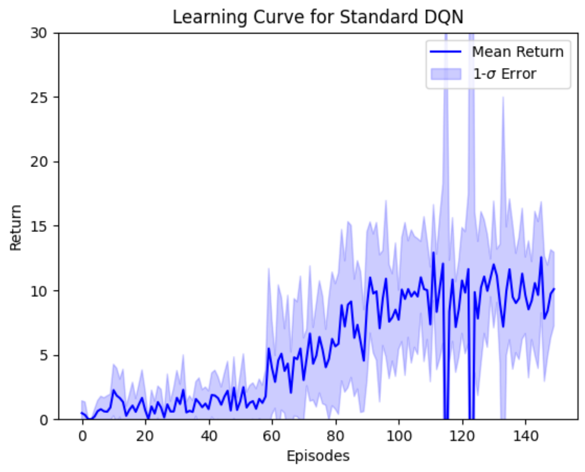

Fig. 1. Mean Return and 1-σ bounds for standard DQN : 10 runs

The optimal policy learned by the agent is shown in Figure 2.

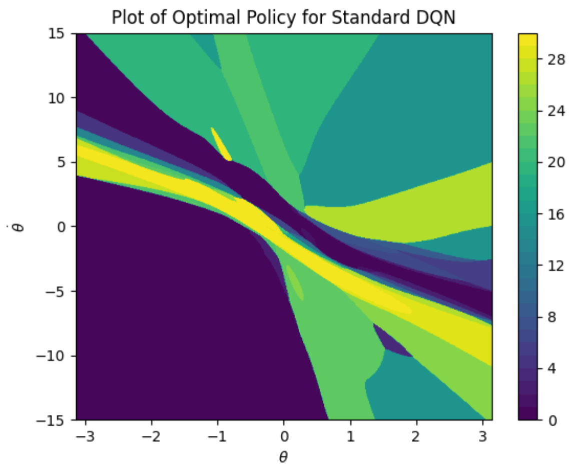

Fig. 2. Optimal Policy for Standard DQN

Furthermore, Figure 3 shows the state-value function for a trained agent. As can be clearly observed from Figure 3, states within $\theta = \pm 0.1\pi$ and $\dot{\theta} \approx 0$ are assigned the highest values. Also states with $\theta = \pm\pi$ and high values of $\dot{\theta}$ are also assigned high values. This is because the high $\dot{\theta}$ will transition the θ from π towards 0.

An example trajectory of the pendulum corresponding to an agent trained using standard DQN is shown in Figure 4. We see that the angle of the pendulum converges very close to 0 after approximately 20 steps, i.e., 2 seconds. Also, the torque τ applied to the pendulum keeps switching between two values to maintain θ close to zero. The animated version of this trajectory can be found in the github repository for this

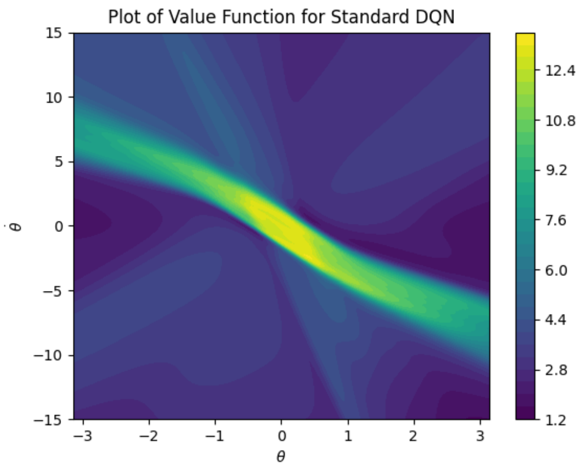

Fig. 3. State-Value Function for Standard DQN

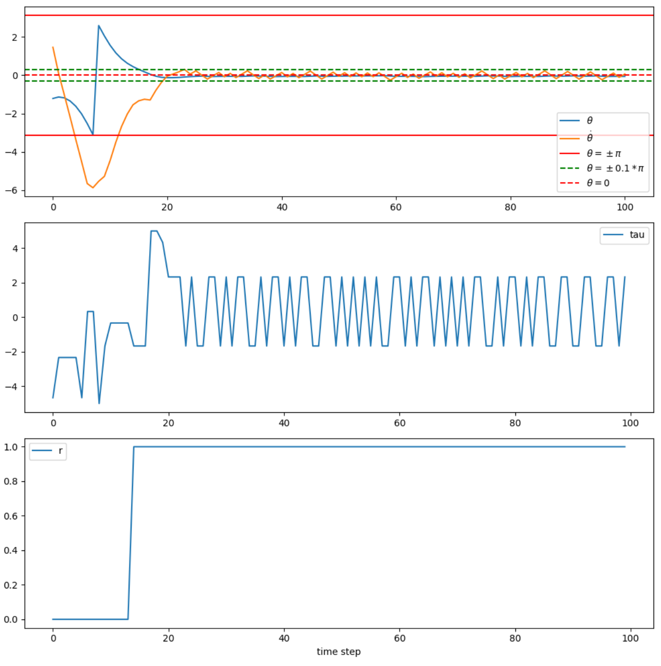

Fig. 4. Example Trajectory for Standard DQN

assignment [1].

B. DQN Without Target Network

This variant of the DQN algorithm was also run 10 times for 200 episodes each. However, the weights of the target network were updated at each step,i.e., target update frequency was 1 as opposed to a 1000 for standard DQN. The mean return and the $1-\sigma$ bound for the return with respect to the episodes is shown in Figure 5. Similar to standard DQN, the mean return reaches close to 10 towards the end of the run. Also, the mean return increases with the number of episodes, indicating that

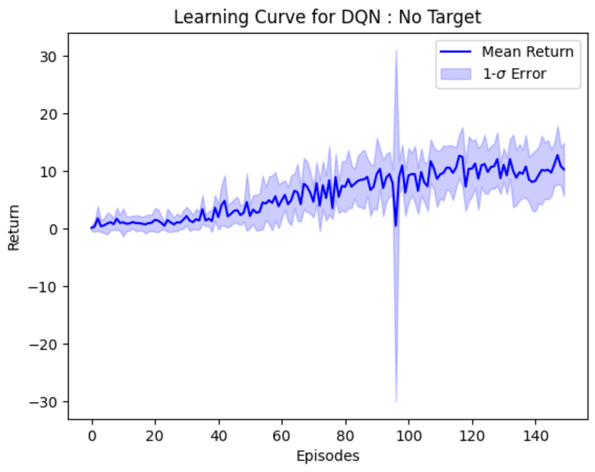

Fig. 5. Mean Return and 1-σ bounds for DQN without Target Network : 10 runs

the agent is learning to stabilize the pendulum in the inverted position. Hence, we can say that updating the target network at every step doesnot significantly affect the performance of the algorithm.

The optimal policy for a DQN agent trained without a target network is shown in Figure 6.

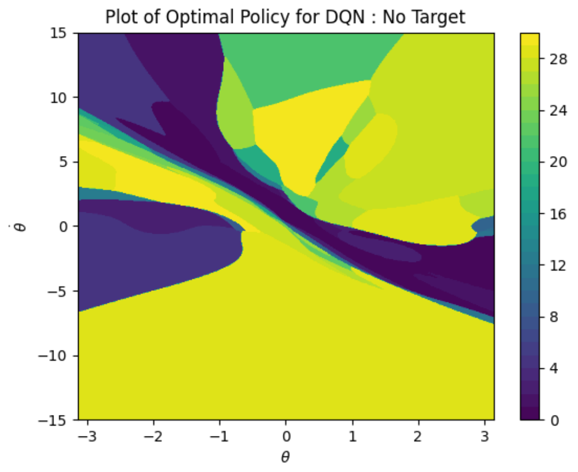

Fig. 6. Optimal Policy for DQN : No Target

Figure 7 shows the state-value function for an agent trained using the DQN algorithm without a target network. The plot is similar to the standard DQN algorithm, where states with $(\theta, \dot{\theta}) = (0, 0)$ are assigned the highest values.

An example trajectory generated using the policy learned by this agent is shown in Figure 8. As can be observed, the angle of the pendulum converges to 0 after roughly 40 seconds. An animation for the example trajectory of pendulum corresponding to this trained agent has been uploaded to the github repository [1].

[1] https://github.com/uiuc-ae598-rl-2023-spring/hw2-dqn-Manavvora

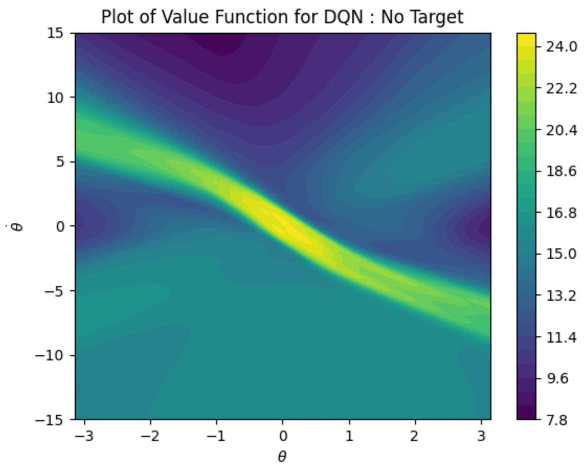

Fig. 7. State-Value Function for DQN : No Target

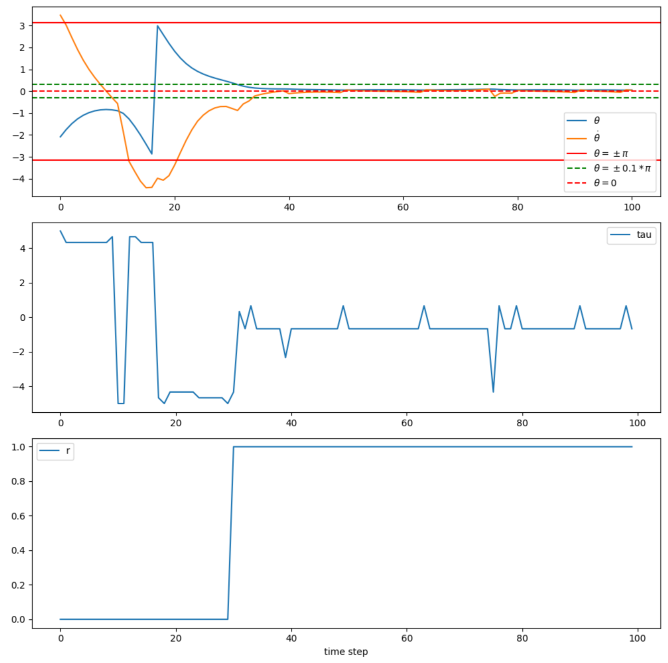

Fig. 8. Example Trajectory for DQN : No Target

C. DQN Without Experience Replay

This variant of the DQN was also run 10 times for 200 episodes each. However, this algorithm doesnot have experience replay. More specifically, the size of the replay memory is set to be the same as the minibatch size. Hence, instead of a random batch of experiences, the weights of the Q-network are updated using the current (s, a, r, s') tuple. The mean return and the $1 - \sigma$ bound for the return with respect to the episodes is shown in Figure 9. As can be seen from Figure 9, the mean return is very low for DQN without experience replay. Across 10 runs, there are only a few instances when the return reaches 10. We observe large negative returns for

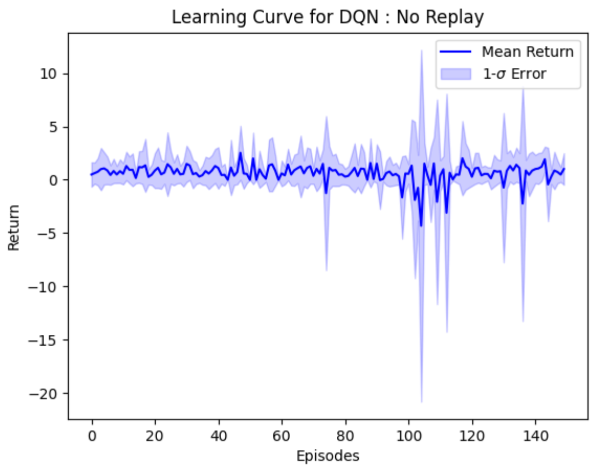

Fig. 9. Mean Return and 1-σ bounds for DQN without Experience Replay : 10 runs

some instances. Overall, we can see that not having experience replay significantly deteriorates the performance of the algorithm. Furthermore, we can say that without experience replay, the agent requires more episodes to learn the optimal policy as compared to variants with experience replay.

The optimal policy learned by the agent for DQN without experience replay is shown in Figure 10.

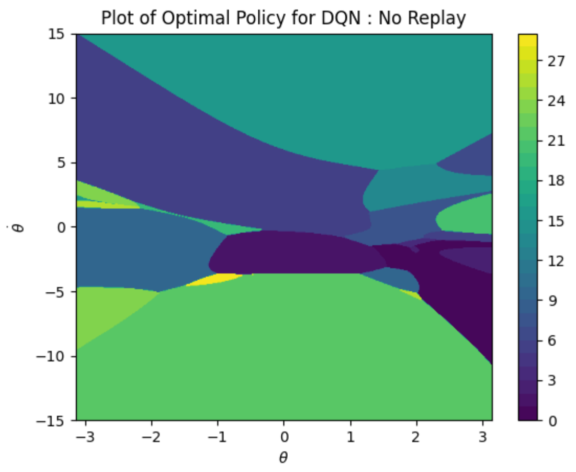

Fig. 10. Optimal Policy for DQN : No Experience Replay

The state-value function learned by this agent is shown in Figure 11. Apart from states with $\theta = 0$, states with $|\theta| > 0.1\pi$ are also assigned high values. This results in the pendulum not being stabilized in the inverted position.

An example trajectory generated using the policy learned by this agent is shown in Figure 12. As can be seen from the plot, the angle of the pendulum keeps oscillating between the two extreme values of $\theta = \pm\pi$. The pendulum is never stabilized in the inverted position.

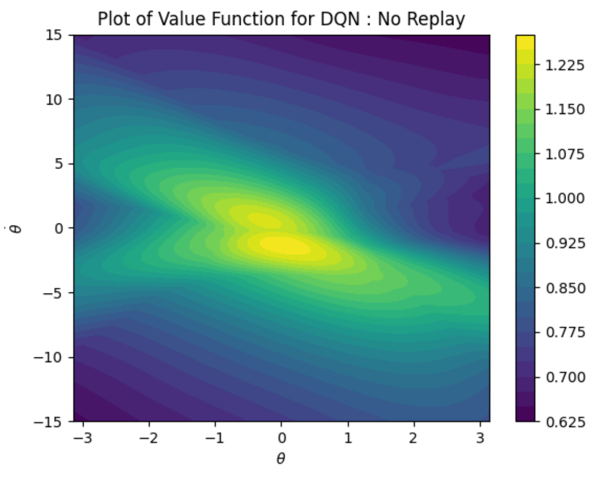

Fig. 11. State-Value Function for DQN : No Experience Replay

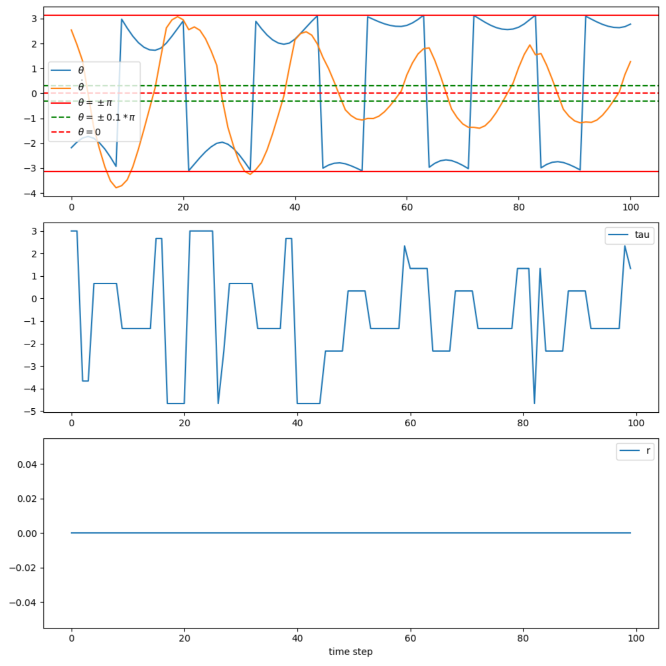

Fig. 12. Example Trajectory for DQN : No Replay

An animation for the example trajectory of pendulum corresponding to this trained agent has been uploaded to the github repository [2].

D. DQN Without Experience Replay and Target Network

This variant of DQN was also run 10 times for 200 episodes each. However in this case the weights of the target network are updated at each step and there is no experience replay for updating the parameters of the Q-Network. The mean return and the $1-\sigma$ bound for the return with respect to the episodes is shown in Figure 13. The learning curve shows a trend similar to that for DQN without experience replay. Hence, we can say that experience replay is the dominating factor affecting the performance of the DQN algorithm.

[2] https://github.com/uiuc-ae598-rl-2023-spring/hw2-dqn-Manavvora

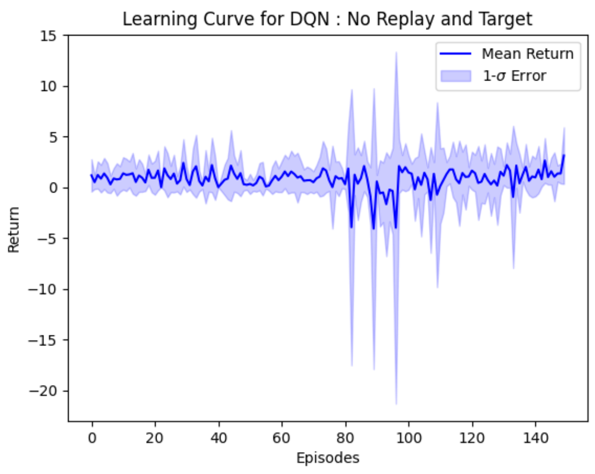

Fig. 13. Mean Return and 1-σ bounds for DQN without Experience Replay and Target Network : 10 runs

The policy learned by an agent without experience replay and target network is shown in Figure 14.

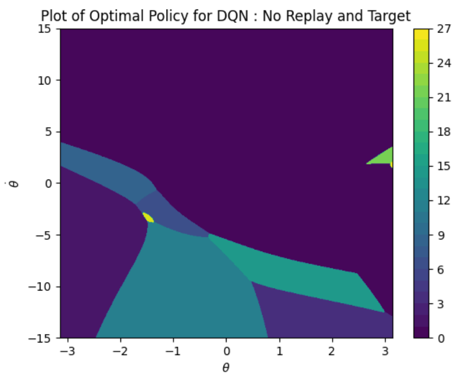

Fig. 14. Optimal Policy for DQN : No Experience Replay and Target Network

The state-value function learned by this agent is shown in Figure 15. Clearly, states with $\theta = 0$ are assigned low values. This results in the pendulum not being stabilized in the inverted position.

An example trajectory generated using the policy learned by this agent is shown in Figure 16. As can be seen from the plot, the angle of the pendulum keeps oscillating between the two extreme values of $\theta = \pm\pi$. The pendulum is never stabilized in the inverted position. However, the angle of pendulum does remain in the acceptable region of $\theta = \pm 0.1\pi$ while transitioning between the extremes. This results in spikes in the reward obtained by the agent.

An animation for the example trajectory of pendulum corresponding to this trained agent has been uploaded to the github

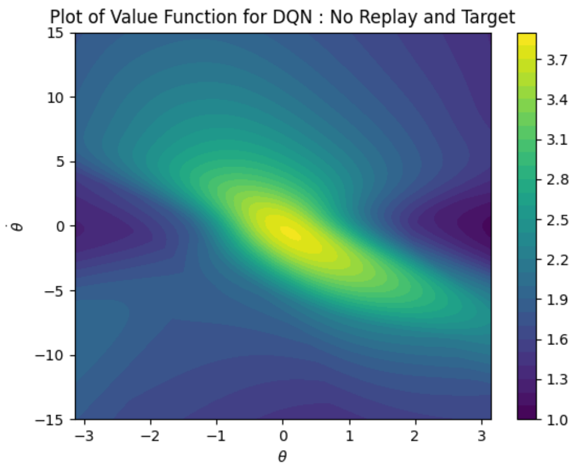

Fig. 15. State-Value Function for DQN : No Experience Replay

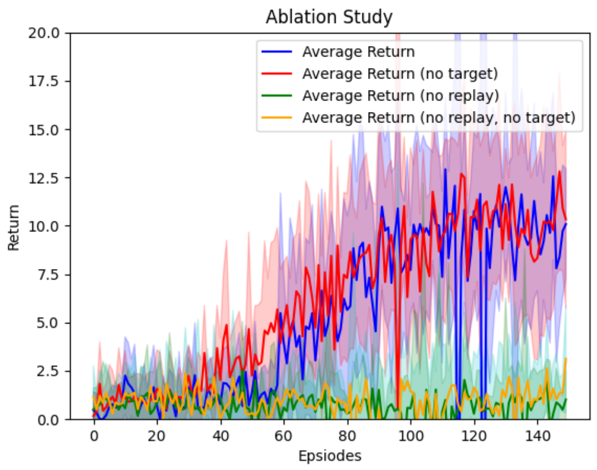

Fig. 17. Ablation Study Plot

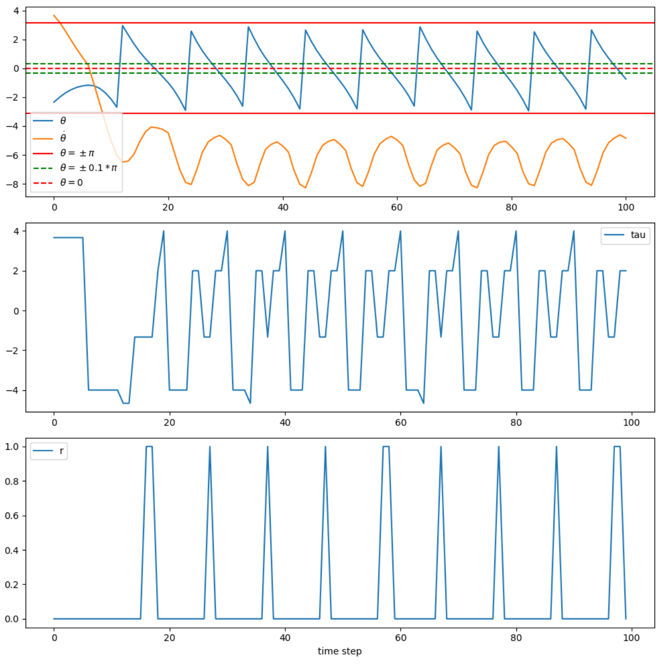

Fig. 16. Example Trajectory for DQN : No Experience Replay and Target Network

repository [3].

E. Ablation Study

We now compare the performance of the above mentioned 4 variants of DQN by plotting their respective learning curves in a single plot. Note that we set the y-axis lower limit to 0 to observe the trends more clearly. As can be observed from Figure 17, standard DQN and DQN without target network have a similar performance, and both these algorithms outperform the non-experience replay variants.

[3] https://github.com/uiuc-ae598-rl-2023-spring/hw2-dqn-Manavvora

IV. Conclusions

In this assignment, we solved the problem of stabilizing a simple pendulum in the inverted position. Four variants of the DQN algorithm [1] were used for the same. From simulations, we observe that the updating frequency of the target network weights doesn't significantly affect the performance of the algorithms. On the other hand, implementation of experience replay plays a huge role in the performance. More specifically, we see that variants with experience replay outperform the variants without replay. Variants of DQN with experience replay are able to stabilize the pendulum in the inverted position whereas the variants without replay are not able to do so.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, pp. 529–533, February 2015.