HW2 - Deep Q-network (DQN)

Alen E. Golpashin[1]
University of Illinois, Urbana, Illinois, 61801, USA

The goal of this report is to implement a Deep Q-network (DQN) learning algorithm to balance an inverted pendulum. The author has implemented DQN algorithm for a pendulum environment with continuous state and discrete action spaces. Different variations of the DQN are studied and a detailed ablation study is carried out. The result of the ablation study shows that the DQN algorithm is more likely to produce an optimal policy if the Experience Replay portion of the DQN algorithm is used. We also observed that convergence to the optimal policy occurred more robustly when the target Q-network update was carried out periodically.

I. Nomenclature

s	=	state
s_1	=	next state
a	=	action
r	=	reward
γ	=	discount factor
n	=	number of episodes
G	=	return
Q	=	control value function
$\hat{Q}$	=	Target control value function
V	=	State value function
c	=	number of steps before update

II. Introduction

The DQN algorithm is a model-free, online, off-policy reinforcement learning algorithm. The 'model-free' refers to the fact that the transition probabilities are not known, while off-policy means that the target control value function is maximized during the update step. The main idea behind this algorithm is to use a deep neural network to approximate the optimal action-value function $Q(s, a)$, which gives the expected total reward for taking action a in state s. The algorithm is implemented by repeatedly calling the emulator (in this case the pendulum environment – Figure 1) and updating the Q-function estimates based on the observed rewards and transitions. At each step of a trajectory, the agent selects an action using an epsilon-greedy policy, which in turn balances exploration and exploitation. The action is consequently applied, and the agent learns of the next state and reward. The experience tuple of *state*, *action*, *reward*, *next state* is stored in a replay memory. This memory buffer is then used to update the Q-function estimate by randomly sampling a tuple from the memory buffer. This is done to lower the correlation of the input data which will help with convergence to the optimal policy [1],[2].

The Q values are updated using a temporal difference update, which is in fact a variant of the Bellman equation. This update gives the optimal Q-value for a state-action pair as the sum of the immediate reward and the maximum Q-value of the next state. We use the mean squared error between the computed Q-value and the target Q-value to derive a loss function. This loss function is then used to update the neural network weights. The update is carried out using stochastic gradient descent (SGD) on batches of experience samples (randomly drawn from the replay buffer) [1],[2]. To address the problem of overestimation, and a "fast changing target Q", the DQN algorithm uses a separate target network to generate the target Q-values used in the update rule. The target network is created by copying the Q-network, and it is only updated every c-steps to diminish the correlation between the target and computed Q-values.

[1] Graduate Student, Department of Aerospace Engineering.

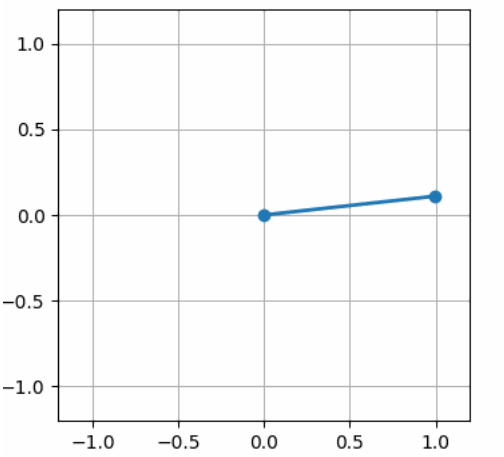

Fig. 1 The pendulum to be balanced inverted at the vertical position.

In the paper Mnih et al. [1], the DQN algorithm has been shown to be effective in learning to play different Atari games, achieving performances comparable or better than that of a human. In this homework report, we apply the DQN to the pendulum environment which has a continuous state and discrete action spaces. In doing so, we demonstrate that even though we do not have a closed form equation of a control gain, the optimal policy resulting from the trained agents is able to control the pendulum. The remaining of this report is organized as follows: in Section III, we present a high-level overview of the DQN algorithm. In section IV, we present the trajectories resulting from a single trained agent. In the same section, we present the results using different variations of DQN algorithm, i.e. by turning on or off the experience replay and target Q update portions of the algorithm. Similarly, in Section V, we will present the results of our ablation study. In the same section, we provide analysis and discussion of the ablation study. Finally in Section VI, we will provide some concluding remarks.

III. Algorithm

We would like to briefly point out that the DQN algorithm can be considered an extension of the famous Q-learning algorithm [2]. As pointed out in the previous section, the DQN uses the experience replay part of the algorithm to eliminate data correlation between target and computed Q-values. Below, we would like to provide a pseudo-algorithm which highlights some of the key steps of the DQN algorithm [1]:

Algorithm 1. DQN.

1. Initialize a replay buffer and a Q-network with random weights. The replay buffer stores experience tuples (s, a, r, s_1). The Q-network estimates the Q-values of state-action pairs.
2. For each episode, reset the environment and generate a random initial state s.
3. For each time step in the episode, select an action a using an epsilon-greedy policy based on the Q-values estimated by the Q-network.
   $a = \text{argmax}\big(Q(s, a)\big),$ if $p = 1 - \varepsilon$
   $a = \text{random}(action\ space),$ if $p = \varepsilon$

4. Execute the action and obtain the next state s_1 and the reward r through the step function of the environment.
5. Store the experience tuple (s, a, r, s_1) in the replay memory.
6. Sample a minibatch of experience tuples from the replay buffer.
7. Compute the target Q-values for each experience tuple in the batch:
   $\hat{Q}(s, a) = r + \gamma \max\big(Q(s_1, a_1)\big)$
   where γ is the discount factor that determines the importance of future rewards.
8. Compute the predicted Q-values for each s/a pair in the batch using the Q-network.
9. Compute the loss between the target Q-values and the predicted Q-values, and use the loss function to update the weights of the Q-network using stochastic gradient descent.
   $L = \Big(r + \gamma \max\big(Q(s_1, a_1)\big) - Q(s, a)\Big)^2 = \Big(r + \gamma \max\big(Q(s_1, a_1; \theta_i^-)\big) - Q(s, a, \theta_i)\Big)^2$
10. Evert c-steps, update a target network by copying the weights of the Q-network to the target network.
11. Repeat steps 2-10 for i-epsiodes, where i is an appropriate number where an optimal policy can be obtained.

Above, the hat operator for Q denotes the target network. The neural network used for this project is a network with two hidden layers that each have 64 units. A $tanh$ activation function is used at both hidden layers, while a linear activation function is used at the output layer. A few details left out in the above algorithm are as follow:
The stochastic gradient descent (SGD) method requires access to the gradient of the objective function. In this case, the optimizer in Python obtains this gradient numerically. However, this gradient can also be computed using the equation

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s_1}\left[\left(r + \gamma \max_{a_1} Q(s_1, a_1; \theta_i^-) - Q(s, a; \theta_i)\right)\nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

where $Q(s_1, a_1; \theta_i^-) = \hat{Q}(s_1, a_1)$, θ^- denotes the target network weights, and θ denotes the Q-network weights. The target Q-network update feature replaces θ^- with θ after every c-steps of the trajectory (step 10 of the algorithm). Moreover, we should point out that at the terminating step of every episode i, the $Q(s, a)$ is set equal to r, as opposed to equal to $r + \gamma \max\big(Q(s_1, a_1)\big)$. Further details on DQN algorithm and its variations can be found in references [1] and [2].

IV. Agent Trajectories

In this section, we will present the simulation trajectory results for four variations of the DQN algorithm. That is, the DQN trajectories 1) with replay, with target Q (i.e., the standard algorithm), 2) with replay, without target Q (i.e., the target network is reset after each step), 3) without replay, with target Q (i.e., the size of the replay memory buffer is equal to the size of each batch), and finally 4) without replay, without target Q (i.e., the target network is reset after each step and the size of the replay memory buffer is equal to the size of each minibatch). After each trajectory is presented, we shall present mean return (learning) curves for 20 realizations of each algorithm variation. In Table 1, we have tabulated the set of hyperparameters used throughout the results section of this report. The choice of some of these parameters is motivated by the parameters used in Reference [1].

Table. 1 Table of Hyperparameters.

Parameters	Values
γ	0.95
ε_{max}	1
$\varepsilon_{increments}$	0.0001
ε_{min}	0.1
n	200
α	0.00025
c	10 episodes
Batch Length	32
Memory Length	10000000
Number of Realizations	20

Note that as suggested in [1], the ε parameter for the greedy action is decayed gradually over every step to increase robustness in convergence.

A. With Experience Replay and Target Q

In this subsection, we would like to begin by examining the standard variation of the DQN, where both the replay and target Q update are carried out. We will look at the three plots of trajectories, policy contour, value function contour, and finally the mean return curves over 20 realizations.

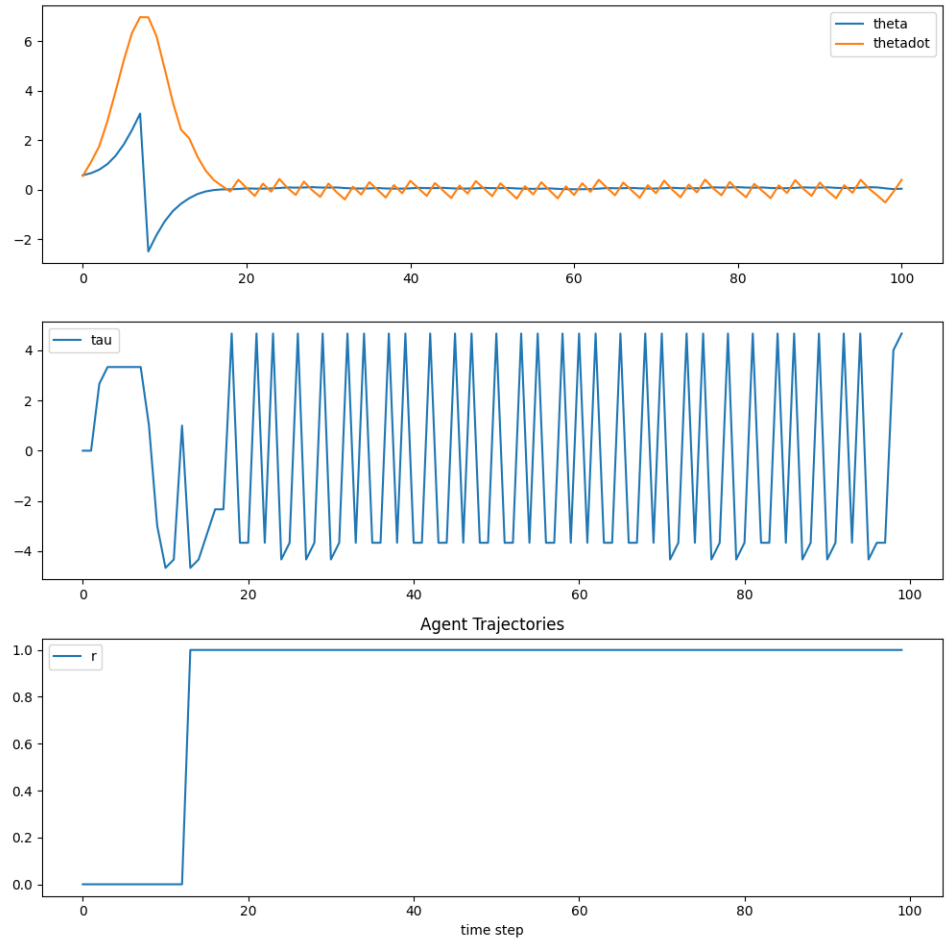

Fig. 2 Agent Sample Trajectories with replay and target Q.

Looking at figure 2, it is obvious that the pendulum has been successfully balanced. Both θ and $\dot{\theta}$ have converged to the 0 degrees, with the $\dot{\theta}$ slightly varying because of the correcting control torque. The periodic torques can be seen exerting correcting action to the pendulum after it has reached the top inverted position. Looking at the reward plot, we can conclude that the pendulum was balanced after approximately the 16^{th} time step.

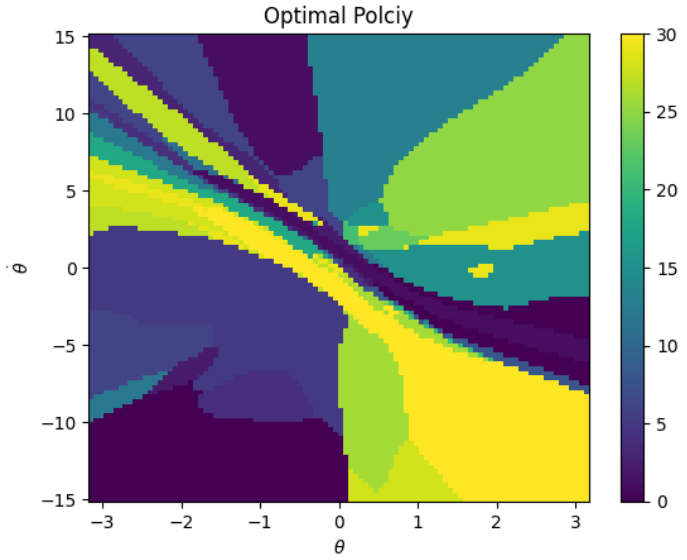

Fig. 3 Optimal policy contour with replay and target Q.

The policy contour refers to the set of control policies that can be used to stabilize a system. In figure 3, we can see that near the boundaries of $\theta \in [-\pi, \pi]$, opposite torque is applied to keep the pendulum from swinging further. This can for example be seen near $\theta \approx +2$ when the velocity is in the same direction, the policy has approached a value of 30. Similarly, we can see that near $\theta \approx -2$ when the velocity is opposing the direction of angle, there is almost no correcting policy applied. This intuitively makes sense because the pendulum only needs to exert correcting actions when the velocity is away from the unstable equilibrium point and the angle θ is not near zero degrees.

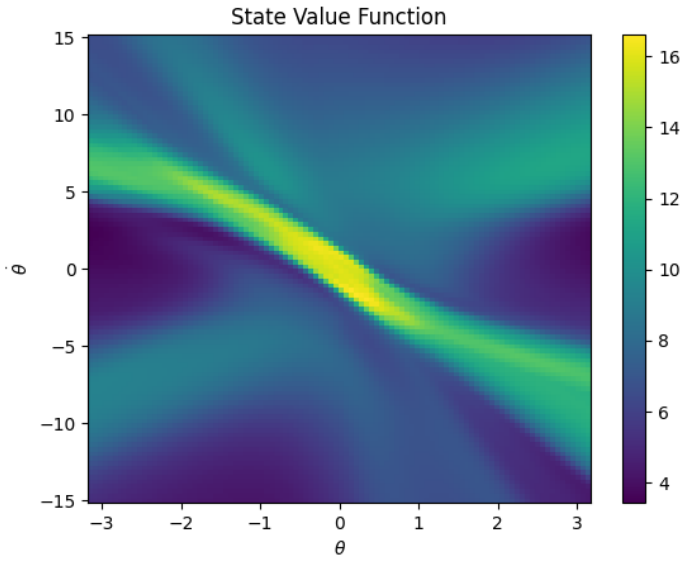

Fig. 4 State value function contour with replay and target Q.

In case presented here, the value function exhibits a characteristic shape where it is "long in the center", meaning that the values assigned to states or state-action pairs that are close to the optimal value are significantly higher than the values assigned to states or state-action pairs that are farther away from the optimal value. We can compare this to a stable manifold, where the highlighted points are where the pendulum the most stable. Alternatively, if we think of the state value function as time-to-go's, then these points represent the points from where the pendulum can be stabilized the fastest. Therefore, the value function is telling us where it is "easiest" to balance the pendulum. In

particular, it is more efficient to balance the pendulum at positive θ and negative $\dot{\theta}$, than if we actually wanted to balance a pendulum starting at a negative θ, with negative $\dot{\theta}$.

Another intuitive reason for this shape of the value function is that, in many MDPs, the optimal policy tends to involve staying in a certain "sweet spot" of states or state-action pairs, where the expected cumulative reward is relatively high and the transition probabilities to other states or state-action pairs are favorable. As a result, the value function assigns higher values to states or state-action pairs that are close to this sweet spot, while values farther away from the sweet spot gradually decrease.

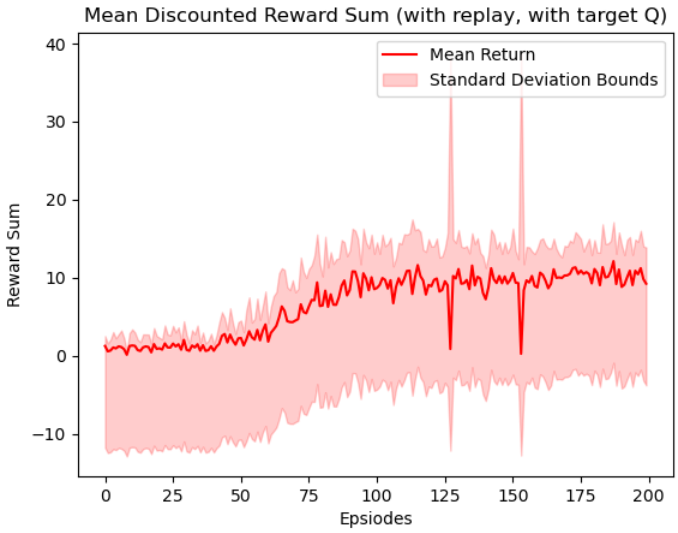

Fig. 5 Mean learning curve resulting from 20 realizations.

Considering the average learning rates of 20 realizations shown in Figure 5, we see that the average learning levels=off near the 10-reward sum, which indicates that most of the pendulum instances were actually stabilized (on average). There are two outlier points near 125 and 160 episodes which have skewed the mean slightly. Ignoring these two outliers, the standard deviation seems to indicate that there is a higher uncertainty in getting lower reward sums than the mean, than it is obtain higher values. In general, the standard deviation tends to stay relatively constant across the episodes.

B. With Experience Replay and no Target Q

In this next subsection, we will similarly begin by considering the second variation of DQN, where only the replay is carried out, with the target Q update turned off. We will look at the three plots of trajectories, policy contour, value function contour, and finally the mean return curves over 20 realizations.

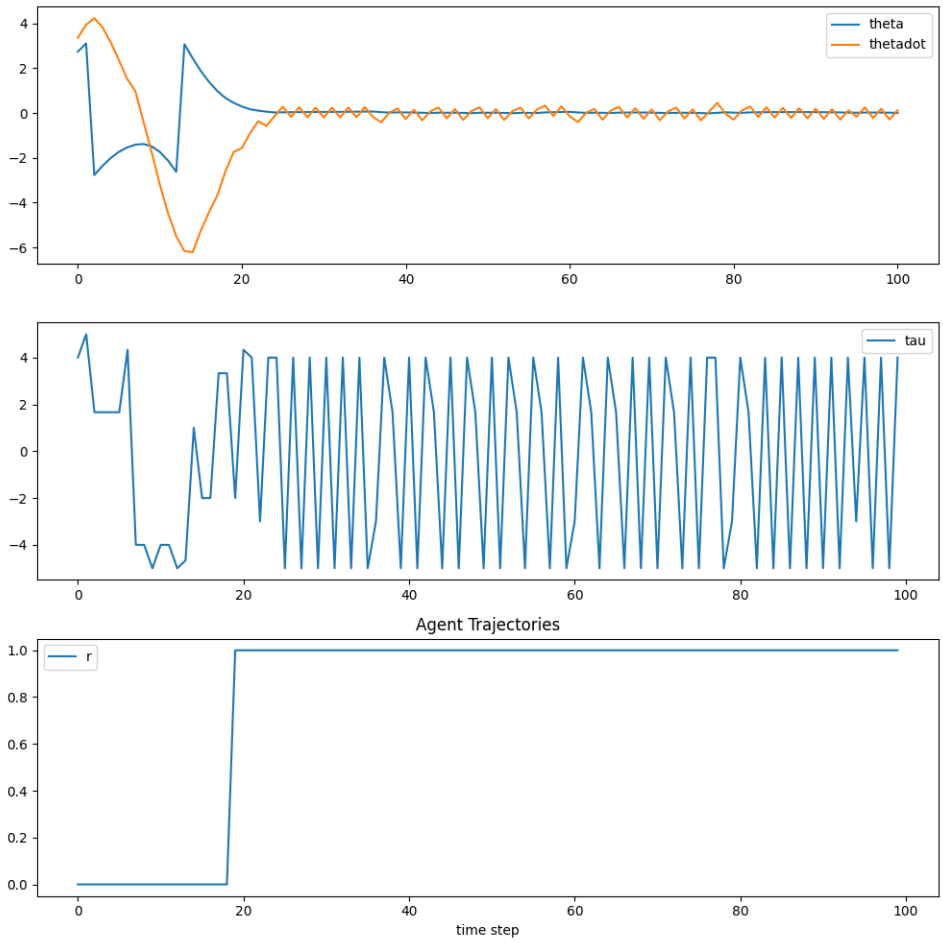

Fig. 6 Agent Sample Trajectories with replay and no target Q.

Inspecting the trajectory plots of Figure 6, we can again see that the pendulum was successfully stabilized. The periodic torques can be again seen exerting correcting action to the pendulum after it has reached the top inverted position. Looking at the reward plot in this case, we can conclude that the pendulum was balanced after approximately the 19^{th} time step. This is a slight delay compared to the previous case where the target Q network was enabled. In fact, we can see from the trajectories that the pendulum was near the $\theta = -\pi$ for a longer time.

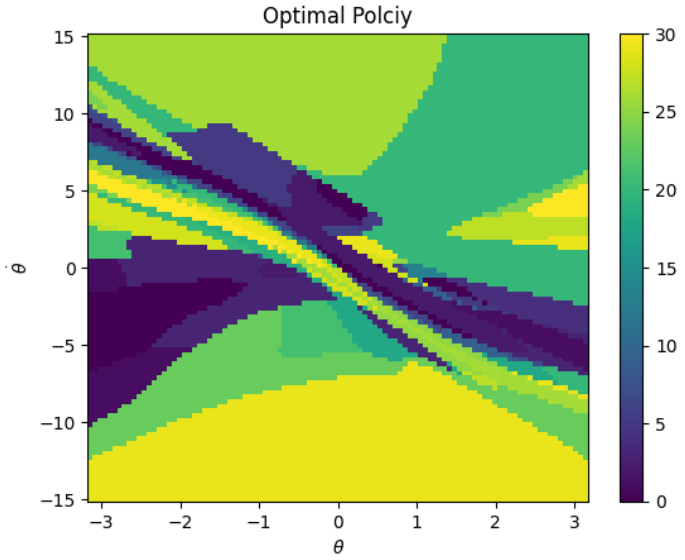

Fig. 7 Optimal policy contour with replay and no target Q.

From Figure 7, we see that the policy contour is similar to the previous case in subsection A. However, we can also see that this policy plot seems to be slightly distorted. For example, we can see that according to this policy, larger negative θ values with larger negative $\dot{\theta}$ values obtain a higher policy value. This might in fact not be needed, thus making the policy sub-optimal.

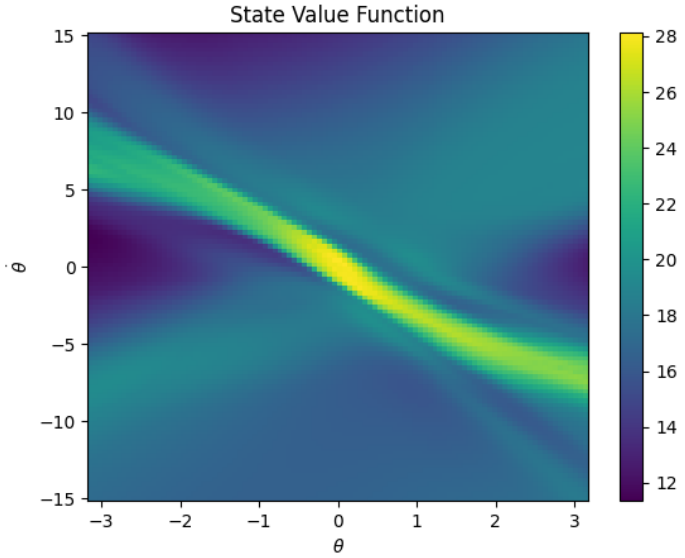

Fig. 8 State value function contour with replay and no target Q.

The “long center” characteristic of the value function can still be seen here. In fact, this can explain why the pendulum has still retained its stability properties with no target Q involved.

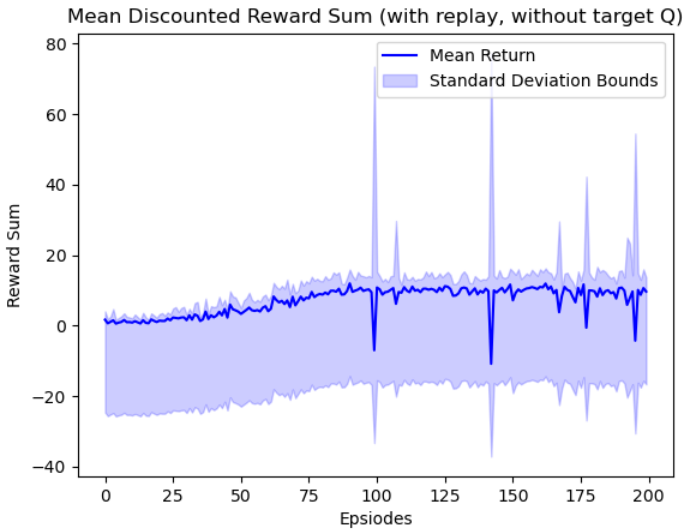

Fig. 9 Mean learning curve resulting from 20 realizations.

Inspecting the mean discounted reward sum of the 20 realizations with no target Q, we see a higher number of outliers than the previous analysis when target Q was enabled. As we saw in the policy contour, this may be due to a smaller stability package (i.e., set of states from where the pendulum can be stabilized using policy in Figure 7). Moreover, the variance tends to still be higher towards zero here, i.e. there is more uncertainty in getting low values.

C. Without Experience Replay and With Target Q

In this subsection, we will be considering the third variation of DQN, where only the target Q is carried out, with the experience replay turned off. We will look at the three plots of trajectories, policy contour, value function contour, and finally the mean return curves over 20 realizations.

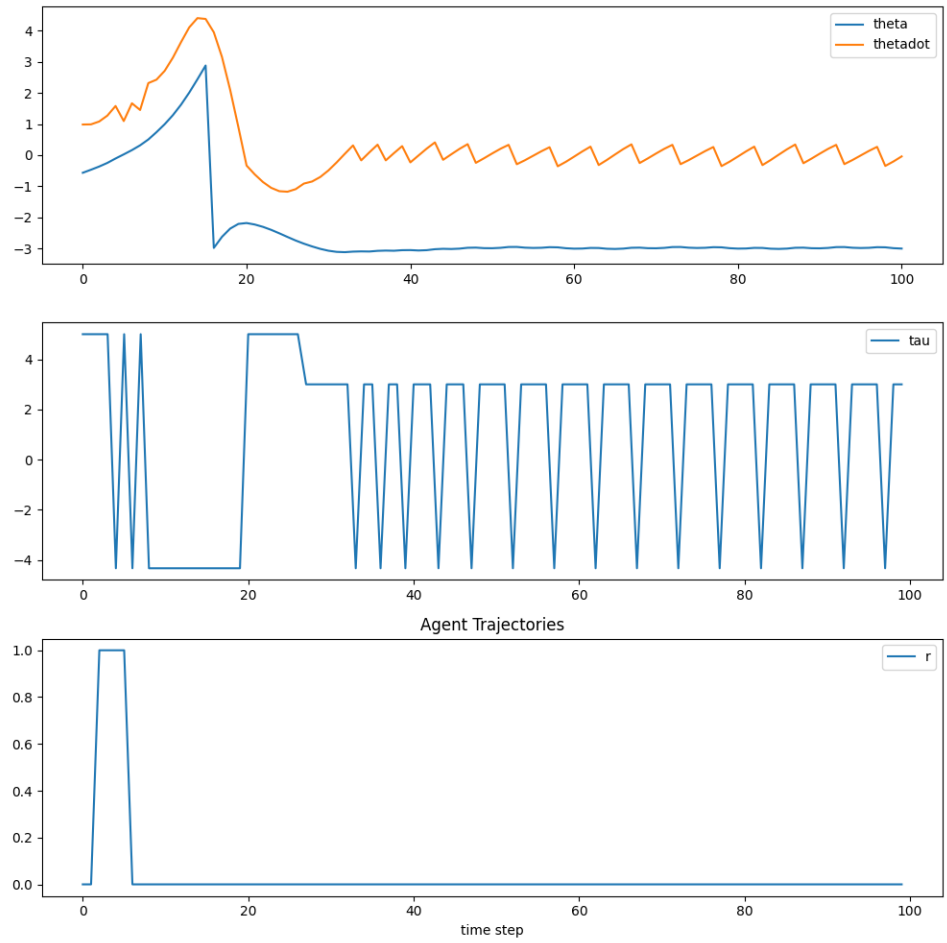

Fig. 10 Agent Sample Trajectories with target Q and no replay.

From Figure 10, we can see that the pendulum is struggling to produce a high frequency balancing torque. There is an initial attempt at balancing which seems to have failed around the 18th time step. This attempt has resulted in an initial reward of 1. After the 18th time step, the pendulum seems to have stabilized around the incorrect point of $\theta \approx -\pi$ with the $\dot{\theta}$ having converged to zero. All in all, the balancing has not been successful with the removal of the experience replay function.

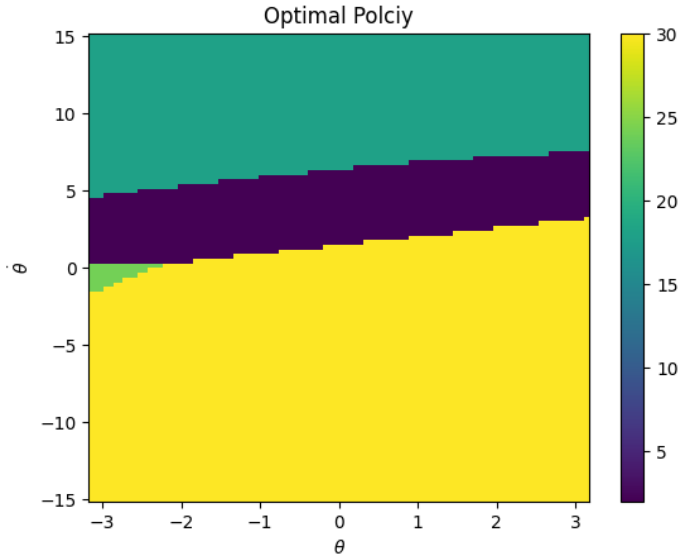

Fig. 11 Optimal policy contour with target Q and no replay.

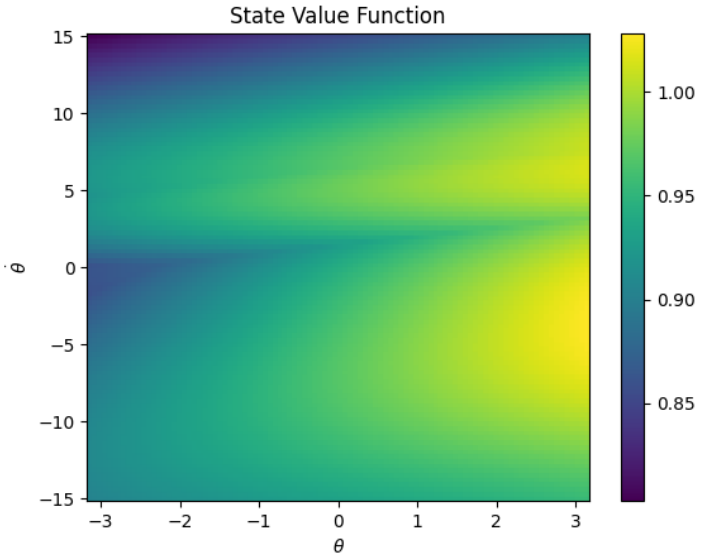

Fig. 12 State value function contour with target Q and no replay.

Both the policy and state value functions have no meaningful information and seem to have not converged to a stabilizing set of values. The optimal policy seems to be suggesting a higher policy value for negative θ, while suggesting lower values when θ is near zero. Comparing this case to the two previous cases, we observe a lack of symmetry both in the policy and the state value function. As we will discuss in the conclusion section, we believe that the poor performance here may be because a second set of hyperparameter are needed.

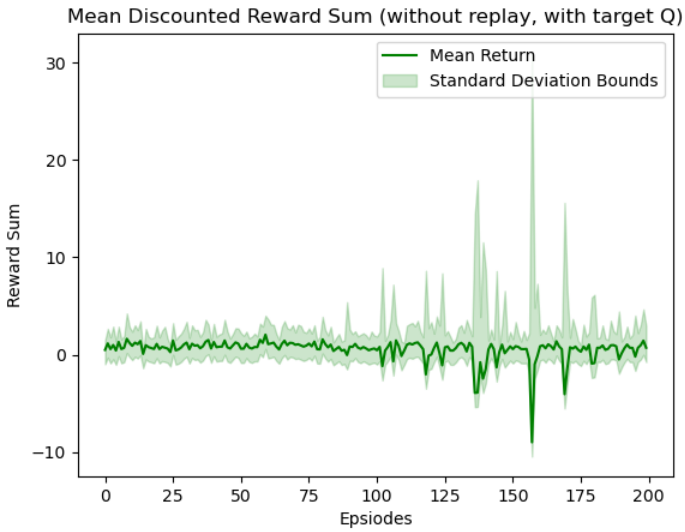

Fig. 13 Mean learning curve resulting from 20 realizations.

Looking at the mean discounted reward sum of the 20 realizations with no replay, we see a higher number of negative outliers than the previous analyses. The mean return values tend to be mostly positive, but near zero. We also cannot see a significant rise in mean return values which indicates that no significant learning was achieved on average. This in turn explains why the pendulum was not balanced. We should point out that without experience replay, DQN would rely only on the most recent experiences (i.e., the most recent s, a, r, s_1 tuple), which may not be representative of the true distribution of experiences. As a result, the algorithm may require more experiences to converge to a good policy. We should also note that the correlation between target Q and Q have increased in this case, which could have resulted in the current mean learning curve.

D. Without Experience Replay and Without Target Q

In this subsection, we examine the behavior of the pendulum and the learning algorithm when both the target Q and the experience replay are turned off. Of course, because turning off the replay in the subsection resulted in poor results, we expect similar results here.

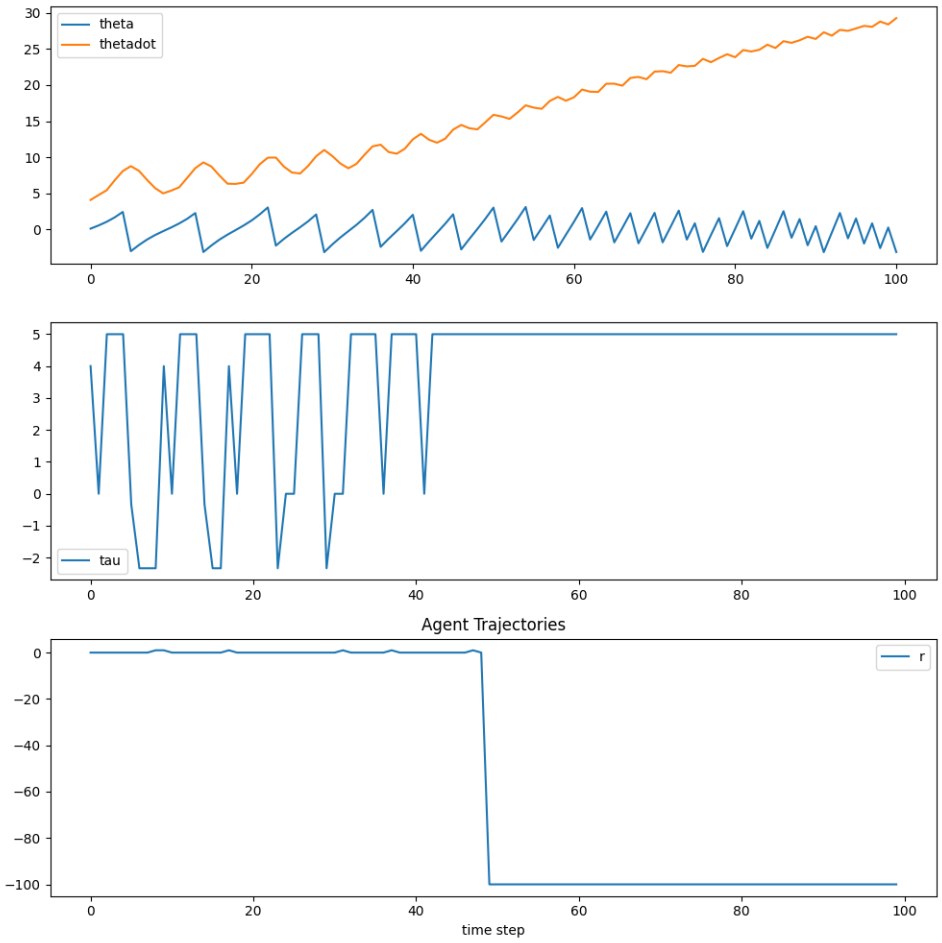

Fig. 14 Agent Sample Trajectories with no replay or target Q.

As expected, the pendulum was not able to fully converge to the inverted position, but interestingly, the velocity $\dot{\theta}$ was increasing and destabilizing the pendulum. Having both target Q and experience replay turned off has resulted in an unstable pendulum.

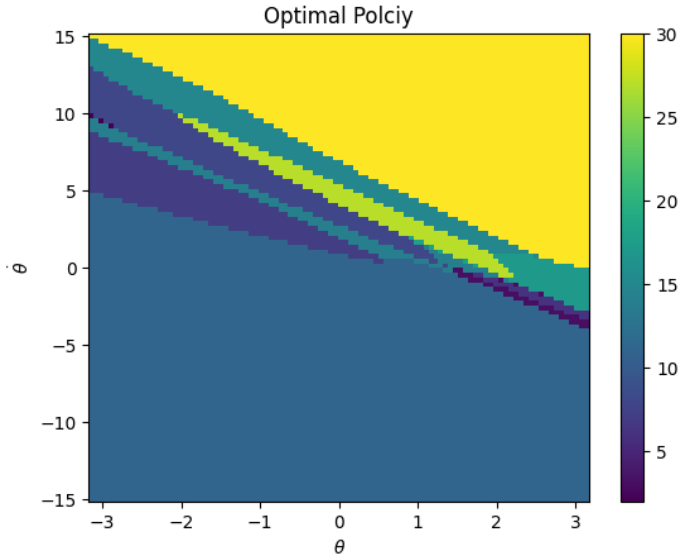

Fig. 15 Optimal policy contour with no replay or target Q.

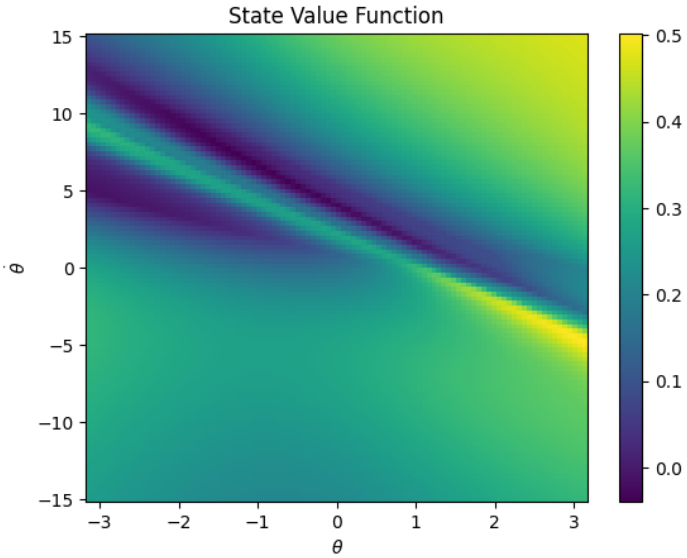

Fig. 16 State value function contour with no replay or target Q.

Although we can see a symmetry forming in Figures 15 and 16, the value function values have not defined a set of stable state values. In fact, the state value function surface seems to be very flat compared to the previous cases. This implies that the state value function may diverged due to high correlation between the Q and the target Q values. We should also keep in mind that this algorithm is not able to learn from previous experiences, thus, this has possibly caused the learning process to stagnate and prevent the algorithm from converging to an optimal policy.

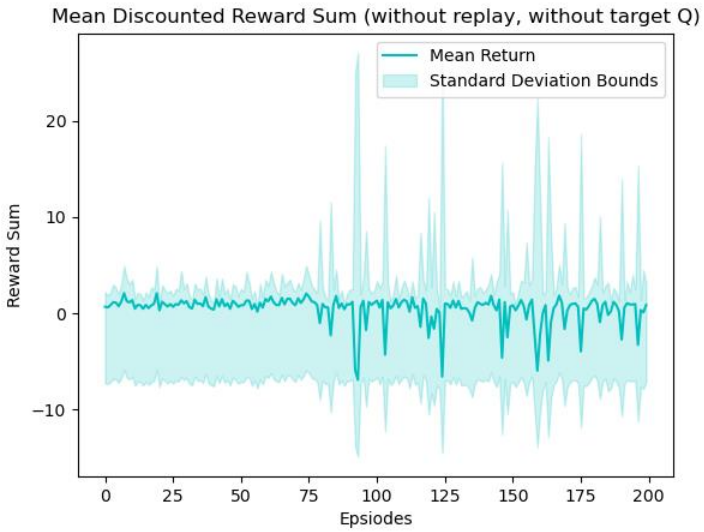

Fig. 17 Mean learning curve resulting from 20 realizations.

The mean learning curve of this case is similar to the previous case (with no replay), with the exception of having a significantly higher number of negative outliers.

E. Video Simulation

The videos (in .gif format) for the previous four cases can be found on the author's GitHub account [3]. The four videos are stored under the names *discreteaction_pendulum1*, *discreteaction_pendulum2*, *discreteaction_pendulum3*, and *discreteaction_pendulum3* in the *figures* folder.

V. Ablation Study

In this section, we present the result of our ablation study comparing the previous four DQN variations through generating mean learning curves over 20 realizations, each with 200 episodes using the parameters of Table 1.

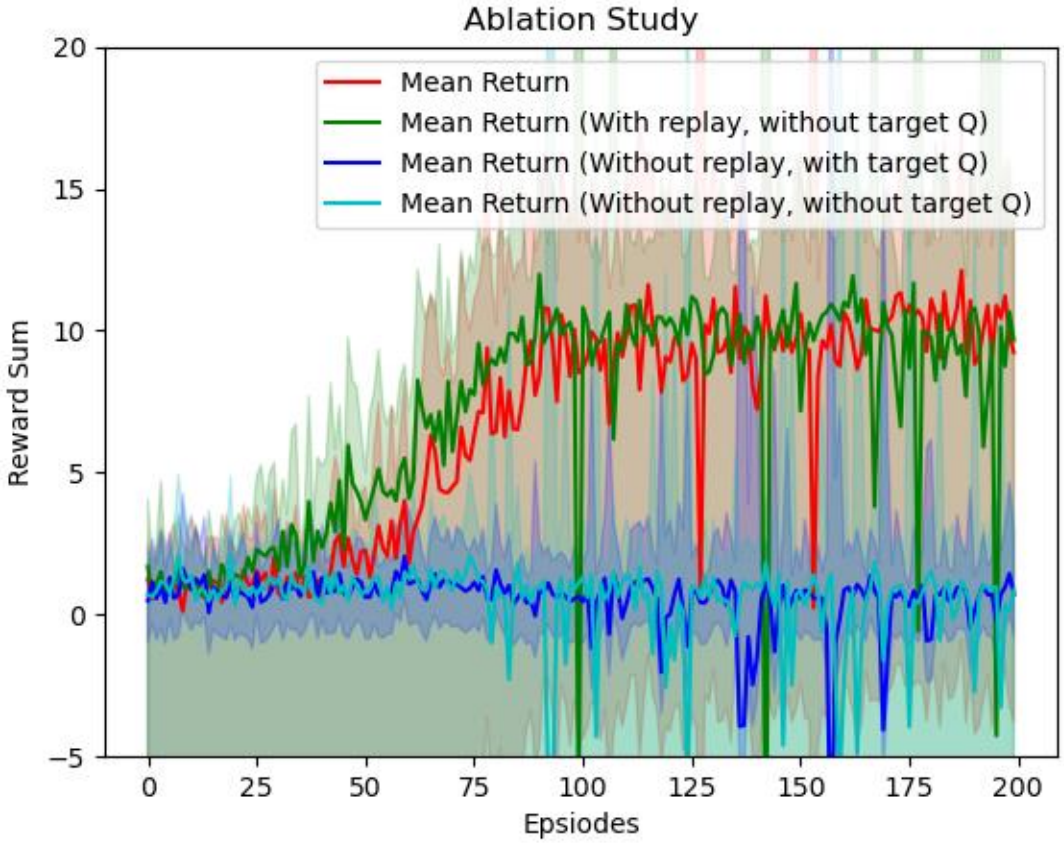

Fig. 18 DQN Ablation study with 4 algorithm variations.

From Figure 18, we observe that the DQN with experience replay, and target Q (in red) has similar performance compared to DQN variation with experience replay and no target Q. However, the latter seems to have a higher number of outliers as the number of episodes increases. It also tends to have its mean learning curve increase earlier, but this is at the expense of higher number of outliers. Therefore, the first algorithm seems to be a more robust option. The other two DQN variations without the experience replay also have a similar experience to each other, but these two variations do poorly compared to the variations that include experience replay. Thus, we can conclude that experience replay is a crucial component of the DQN algorithm. In fact, comparing our results to the results of Mnih et al. [1], Extended Data Table 3 indicates that the without replay variation do significantly worse than the other variations. For example, in the Breakout game, DQN with replay and target Q has a score of 316.8, while the without replay, the score is only 3.2. This agrees with our findings in this section.

VI. Conclusion

In this report, we explored the DQN algorithm to balance a pendulum. The approximation of the Q function was made possible by the use of neural networks. The experience replay turned out to be a crucial part of the DQN algorithm according to our results. The experience replay increases variance, reducing correlation between consecutive Q network data. Without the replay function, our results showed that on average very little learning was happening. Similarly, we experimented with the omitting the target Q functionality. In doing so, we observed that mean learning curves that lacked the target Q functionality were less robust and contained a higher number of outliers. Aside from over-estimating the Q, we observed the slow convergence of the value function to the solution. This can be seen by comparing Figures 3 and 7 above. Figure 7 having lacked the target Q functionality was missing details of policy surface function. From this we can conclude that not including the target Q component can result in convergence to low details solutions.

Working on this report, we also realized that the effectiveness of DQN can be influenced by various factors, such as the choice of hyperparameters. In fact, the code seems to be so sensitive to the choice of hyperparameters, where we were forced to choose a decaying epsilon profile to improve convergence. In the end, we found that the simulation instances that lacked the experience replay were not able to balance the pendulum on average. This result agrees with the findings of [1]. However, we believe that (as a conjecture) that there exist a set of hyperparameters that algorithm variations without replay can still balances a pendulum with. Of course, we imagine this set of hyperparameters to be smaller, hence, making balancing pendulums without the replay function significantly more difficult.

Overall, this report has shed some light on important aspects of DQN implementation and how to address them.

Acknowledgments

The author is thankful to Prof. Huy Tran and other classmates for their constructive discussions and guidance during completion of this report.

References

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

[2] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[3] Golpashin, A. (2023) https://github.com/uiuc-ae598-rl-2023-spring/hw2-dp-golpashin