AE 598 Homework 2 - DQN

1st Scott Bout
Department of Aerospace Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois
bout2@illinois.edu

***Abstract*—This document aims to provide a brief overview of the homework 2 assignment - DQN, for the AE 598 RL Spring 2023 semester.**

I. Introduction

Deep Q networks rose to fame in 2013 when researchers from DeepMind released the landmark paper "Playing Atari with Deep Reinforcement Learning" [1]. I actually remember when this came out and my dad showed me the videos DeepMind released to accompany it showing the superhuman performance on select atari games. 13/14 year old me never would have thought I would be learning the theory behind it and/or replicating the algorithm (on a smaller scale). This document aims to provide a brief discussion of the problems, methods, results, and general conclusions. The format was chosen via the IEEE two column template provided through overleaf. To begin with, the focus will be on methods and the algorithm.

A. Methods and Algorithm

This homework aimed to familiarize students with not only the theory, but the implementation behind DQN - an important distinction as when it comes to reinforcement learning, implementation matters. Deep Q networks operate around the basic principle behind Q-learning, but estimate the Q values via a deep neural network. For this assignment, the Q network consisted of two hidden layers with 64 neurons each, and hyperbolic tangent activation functions applied to said hidden layers. The target Q network is a replica of this architecture, and the network parameters are initialized to be the same as the Q network, effectively making the target Q network a copy of the Q network. The optimizer chosen was Adam, a commonly used high performing optimizer, and the loss function by which the deep neural network parameters were optimized was PyTorch's built in SmoothL1loss. This loss function operates similarly to the Huber loss with a couple notable distinctions. From the PyTorch documentation, Smooth L1 loss is equivalent to $huber(x, y)/beta$ so as beta approaches 0, Smooth L1 loss converges to L1 loss, while Huber loss converges to constant 0 loss. At beta of 0, Smooth L1 is equivalent to L1 loss. As beta approaches $+\infty$ Smooth L1 loss converges to a constant 0 loss while Huber loss converges to Mean Squared Error loss. Furthermore, the hyperparameters are summarized in the table below.

Hyperparameter	Value
α	0.001
batch size	64
starting ϵ	0.9
ending ϵ	0.05
ϵ decay	1000
γ	0.95
number of episodes	1000
Target Q update frequency	10

Fig. 1. Hyperparameter Table

These hyperparameters are chosen by the scientist/engineer and there is no predefined way of selecting them. The same can be said for the optimizer and loss criterion. These hyperparameters were chosen because the author thought they struck a good balance between what the algorithm needs and performance. From what the author understands, a learning rate that is too small would take very long to converge to a local optimum, while two large can cause divergence and missing the local optimum entirely. From a paper that looked into the effects of batch size and training behavior/performance, the authors concluded that "It has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize" [2]. These days, popular batch sizes typically range between 32 and 128. 64 was chosen as a mean between these. ϵ is an important hyperparameter that is responsible for the amount of exploration vs exploitation. In reinforcement learning practice, it is typically vital to include rather high exploration in the beginning, before gradually switching to exploitation as the model learns the value landscape and is better able to make optimal actions accordingly. As a result, the hyperparameters to choose are how much exploration in the beginning and the end. Since $0 < \epsilon < 1$, a large ϵ of 0.9 and therefore a large amount of exploration was chosen, and ϵ annealing was implemented such that throughought the training transient, exploration would slowly decrease and exploitation would slowly increase until a final value where $\epsilon = 0.05$ which has almost no chance for exploration and instead entirely relies on exploitation. The final major hyparameter to be chosen is γ. As γ approaches 1, the algorithm cares more about what happens in the future, while a γ farther from 1 results in a more myopic view of the rewards. As this environment aims to stabilize a pendulum within a reasonable time frame, the best guess is 0.95 was chosen as a reasonable value for this.

B. Problems

This brief section aims to provide a quick overview of some of the problems the author encountered while implementing DQN. The first major problem was a fundamental misunderstanding of how the Q and target Q networks are updated. This led to confusing performance whereby it would not usually converge to a near-optimal solution. It was only after doing the peer review that the author recognized his mistake. The other major problem was the initial commit did not include proper ϵ annealing which resulted in not enough exploration in the beginning and not enough exploitation near the end. Another problem was that the initial target Q network parameters were not initialized to be the same as the Q network.

C. Code

This section aims to provide some guidance on navigating and understanding the code. To start, the results are stored in the ”results” folder in the ”figures” folder. In there are 4 folder representing the results of the ablation study, each named according to what component of the ablation study is different. The contents of each are identical meaning each contains a learning curve, which is the mean sum of rewards, called return, vs the episode number, a trajectory example, a gif of the trained policy, and mesh grids representing the policy and value function. Two files were added from the initial commit. These are the DQN.py file and the train_discreteaction_pendulum.py file. The DQN file contains the actual algorithm and is what the agent is initialized through in the train_discreteaction_pendulum.py file. Each method within the DQN.py file is commented to provide a general overview of what it does, its parameters, its parameter types, and what the expected return data type is. This is all an attempt at making the code as readable, explainable, and reproducible as possible. Finally there was slight modifications made to the discreteaction_pendulum.py to make the animated gif method compatible with PyTorch.

II. Results

This next section aims to provide the results and a brief discussion of the plots and diagrams. The format will simply follow the ablation study under each circumstance. The term with or without replay refers to the capacity of the replay buffer, so with replay means the capacity of the replay buffer is larger than the batch size, while without means the size of the replay buffer is the same as the batch size. With target refers to whether or not there is a static target Q network. With target means the Target Q network gets updated every 10 or 100 or so episodes. The decision after how many episodes is a hyperparameter for the engineer to choose. Without target refers to a non static target Q network, specifically, the target Q network is reset at every single timestep.

A. With Replay and With Target

To start, the results representing the standard algorithm whereby the replay buffer size is larger than the batch size and a target Q network that gets updated in this case every 10 episodes. The learning curve is shown below.

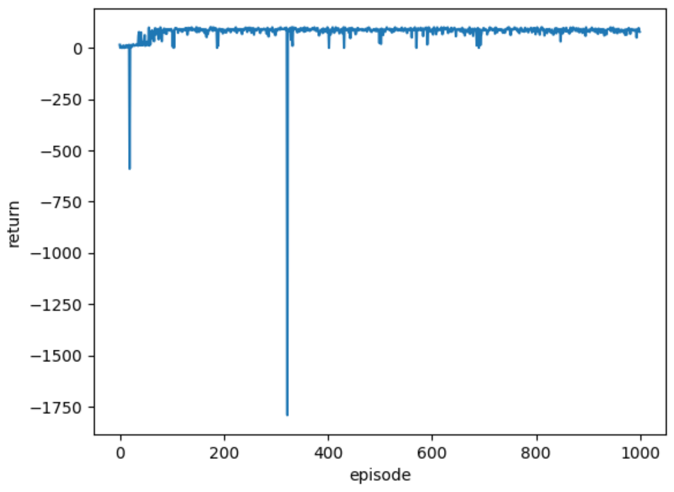

Fig. 2. Learning Curve of standard DQN algorithm

Here the return is defined as the sum of the reward per episode. While the scaling of the occasional outlier makes it a little bit difficult to very clearly see the trend, within 100 timesteps it is able to converge to a near optimal policy and attain high rewards. There is a major outlier between episode 300 and 400. The cause of such a massive negative return is a little confusing but suggests that its actions resulted in an extremely suboptimal result in that specific episode. While it may not be uncommon to see large negative rewards at the beginning fo the training transient, this outlier is suspicious and perhaps a warning that something is not operating entirely correctly. Next is an example of the trained trajectory whereby the policy aims to stabilize the system at 0 for both theta and theta dot.

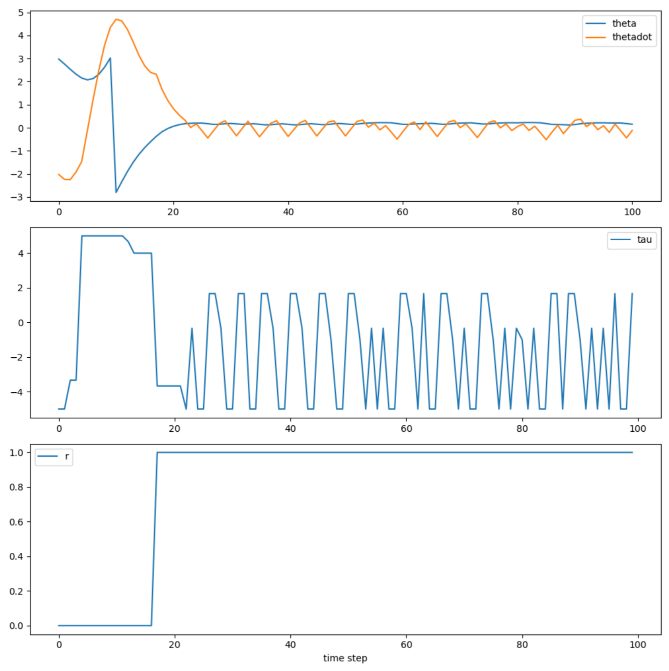

Fig. 3. Example Trajectory of Trained Policy

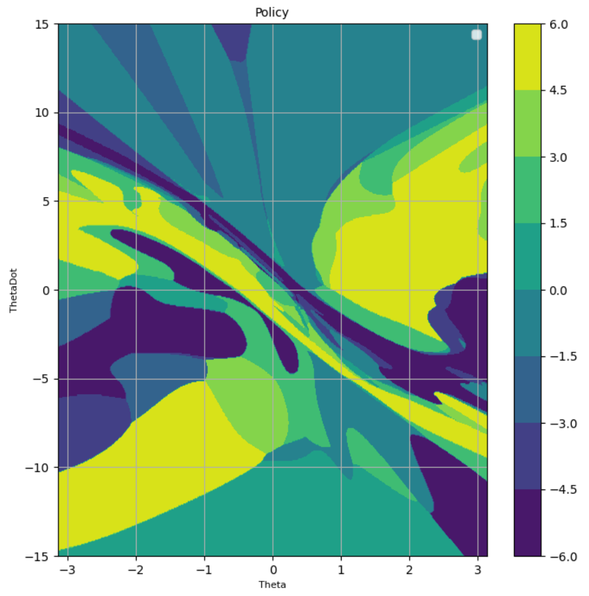

Fig. 4. Policy With Respect to State

Near optimal performance can be seen with some oscillation but especially looking at the action (or tau) graph in the center, there is a solid expected performance whereby the action oscillates between negative and positive in an attempt to stabilize. It can also be seen that the reward converges and remains at 1 within 10 timesteps. When thinking about why this system still oscillates near zero and does not remain or converge at the unstable equilibrium, this reward graph is the most obvious answer. Since reinforcement learning algorithms aim to maximize the expected return, if the reward structure allows for oscillations, there is no incentive to further optimize the policy. The author suspects that if the reward structure were refined such that the allowable error to receive the maximum reward were reduced, the resulting policy would be forced to converge to a more stable trajectory. The author is not familiar enough with the environment or DQN to complete a full analysis on this, the primary reason this may not be possible is due to the discrete action space.
Next we have a graph representing the policy.

The state of the environment can be represented using theta and thetadot. The combination of these two leads to the policy, which is the argmax of the Q values at said state. As can be seen via the bar on the right, the policy learns an action space between -6 and 6. An interesting pattern forms whereby the policy is almost diagonally symmetric. What is also interesting is the chaos as theta and thetadot converge to zero, and that as thetadot, which for a pendulum intuitively may be thought of as the angular velocity, as that angular velocity reaches really large positive values, the policy action actually approaches zero. Also seen are very large actions taken at zero-zero, as mentioned earlier when the author described chaos. This may have to do with the principle of optimality which is built into reinforcement learning, so the quickest way the pendulum can reach its maximum reward is to use the largest actions in the action space. What is also very interesting is looking at for example, a theta of 1, and thetadot being -5 or 5, it would be logical that the action taken should at least be of the opposite sign, but instead they are both largely positive. It is a thin line however on the negative side suggesting that past that boundary, if the angular velocity is large enough, it is better to flip around the axis rather than try and counter the force.
Below is a figure representing the values of each state

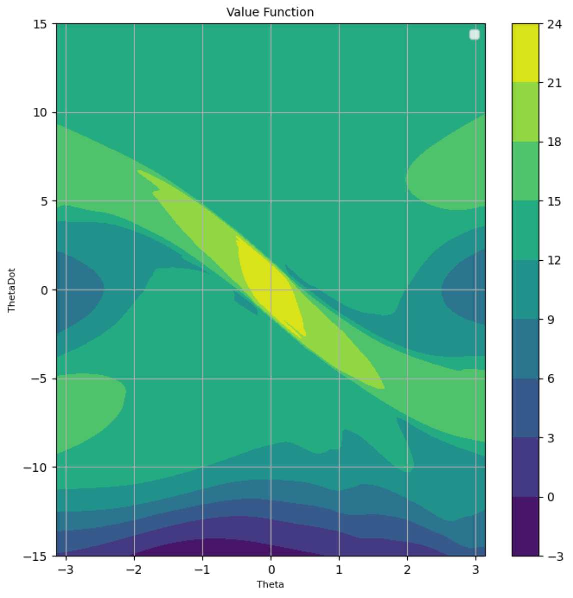

Fig. 5. Values With Respect To State

This figure is easier to parse for the reader. Unsurprisingly, it is nearly symmetrical along the diagonal. A good sanity check is that the largest values occur around the zero-zero point, defined by a theta and thetadot of zero. The mirrored tanh-like structure that can be seen for the basis of symmetry has a higher average value. This is likely because these are states that approach and converge to the target state.

B. *With Replay and Without Target*

This next section represents the results whereby a reasonably sized replay buffer was used, but the target Q network was reset at every timestep. The learning curve with respect to return over time is shown below.

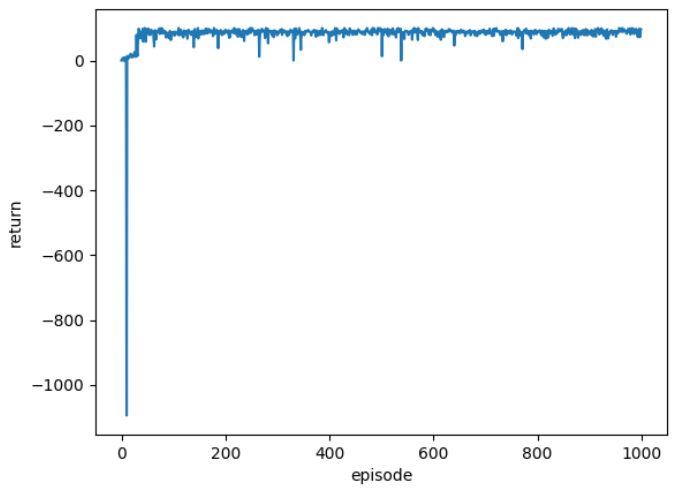

Fig. 6. DQN Learning Curve With Replay Without Target

As can be seen, resetting the target Q network at every timestep does not have a very visible effect on the return over time. In fact, one could say this loss curve is smoother and less noisy than the loss curve generated in the previous section whereby we had both replay and target. The learning curve shows a very clear trend where it makes some mistakes in the beginning but within 75 episodes it has converged to a near optimal policy, near optimal in the sense that it is maximizing the expected return. It retains this throughout and appears to slightly refine the policy at around 800 timesteps to avoid smaller magnitude outliers that were occasionally appearing. However, a closer look shows that the average return is slightly lower than with target Q. This trend follows the ablation study conducted by Minh et al [3]. Below is an example trajectory from the trained agent this loss curve represents.

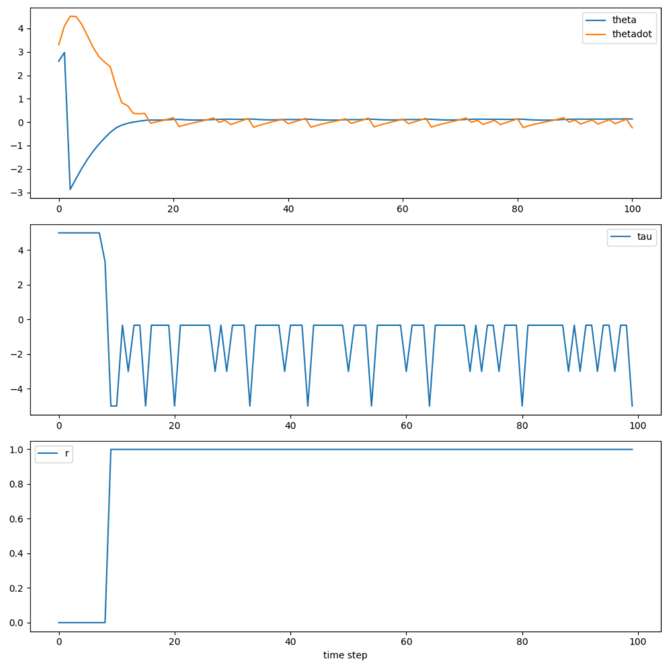

Fig. 7. DQN Trained Example Trajectory With Replay Without Target

As seen, the trajectory does converge to a near optimal solution based on the rewards. In fact, the reward graph shows a quicker convergence to the target state than in the case where there was a target Q. It appears to have found a different strategy whereby instead of oscillating around 0 as the earlier policy did, this policy performs one large push at the beginning and simply keeps applying a negative action toward the environment to ensure high reward. Intuitively, this is a less pure result. It appears to be exploiting the error term in the reward structure rather than actually trying to converge to 0 as the former policy did. This can be seen in the policy graph below.

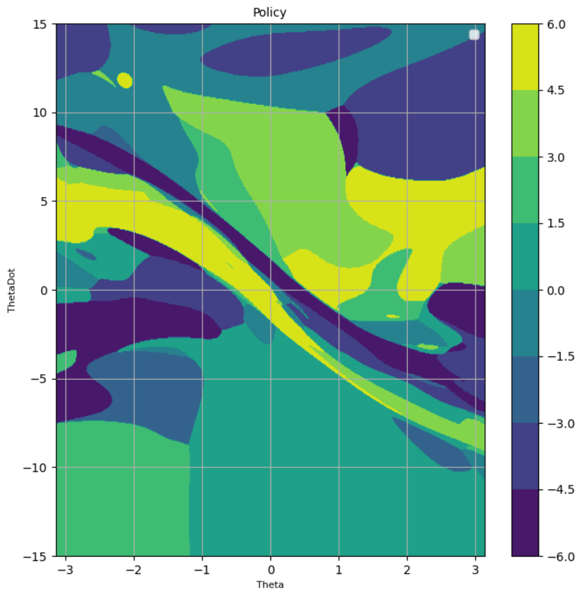

Fig. 8. DQN Policy With Respect To State With Replay Without Target

The policy graph, when compared to Fig. 4. is similar in the mirror tanh-like structure, but varies drastically outside this region. Most notably, this policy does not appear to be symmetrical outside this region, as Fig. 4. was much more. This is perhaps what was reflected in the trajectory. The way this policy graph is shown, it appears to favor spinning the pendulum in one direction every time. Checking the value function below.

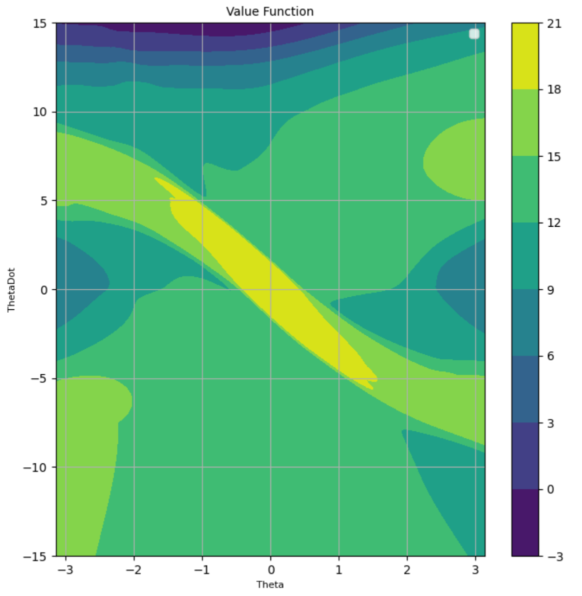

Fig. 9. DQN Values With Respect To State With Replay Without Target

Fig. 9. is very similar to Fig. 5 with two main differences. Fig. 9 shows a significantly larger region that is the maximum value, and that maximum value is not as large as the maximum value in Fig. 5. Furthermore, it can be seen that the negative values are at large positive thetadot for Fig. 9. while these negative values can be seen for large negative thetadot values in Fig. 5.

C. Without Replay and With Target

This next section takes into account DQN without replay but with a Target Q network. Without replay means the replay buffer size is the same as the batch size, effectively eliminating any replaying during the learning transient. Below is the learning curve.

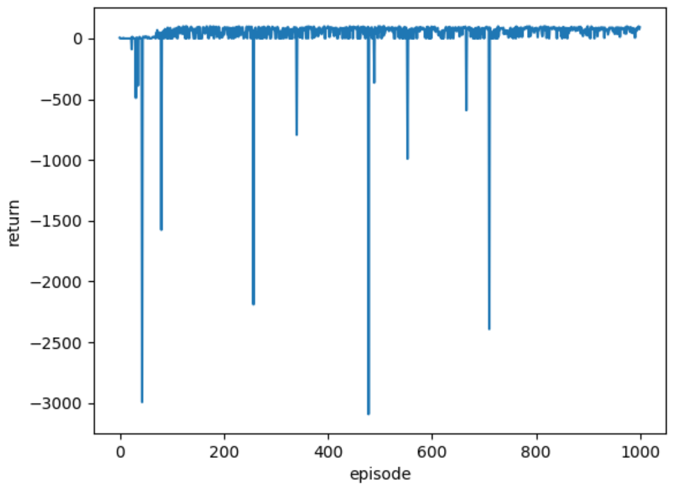

Fig. 10. DQN Learning Curve Without Replay With Target

As is immediately obvious, this learning curve is extremely volatile. Clearly, removing the replay ability has a major affect on the learning transient. Once again, this trend follows the ablation study conducted by Minh et al [3]. Despite this volatility, it can be seen that after around 800 episodes, the noise is reduced significantly. This may suggest that one of the primary detriments to removing replay is training time, and this makes intuitive sense. Without the ability to replay, it may simply need to sample that many more trajectories to reach the same conclusion. Below is an example trajectory from the agent trained from the loss curve shown.

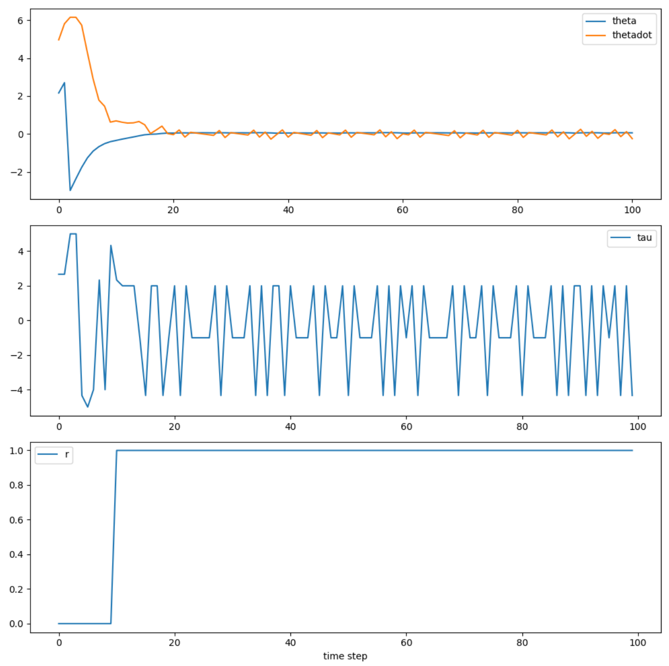

Fig. 11. DQN Example Trajectory Without Replay With Target

As seen, this is a decent trajectory and quickly converges to its target state and oscillates around 0. The trend so far looking at trajectories suggest that having a target Q is necessary to oscillate around 0 rather than converging to a policy that prefers moving in one direction only. Comparing Fig. 11 to Fig. 3, they are similar except some could argue that the policy without replay results in a better trajectory. It is unclear why exactly this may be, it is likely this is an outlier. Looking at the policy below

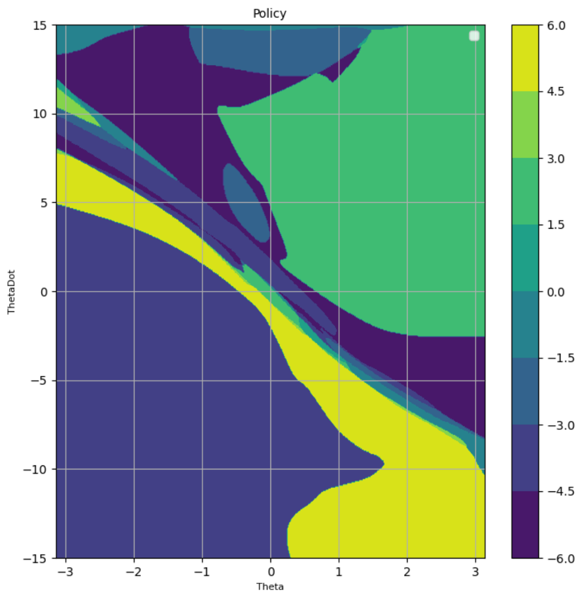

Fig. 12. DQN Policy With Respect To State Without Replay With Target

Comparing Fig. 12. to Fig. 4, we see glaringly obvious differences. Most notably, there is absolute no diagonal symmetry in this policy. This once again suggests it favors moving to one side, but the region the author described as a mirrored tanh-like region seems capable of countering these large asymmetries. Furthermore, this mirrored tanh-like region is not so clearly defined as in Fig. 4., and with the human eye, is simply appears less chaotic. This is reflected in the value graph below.

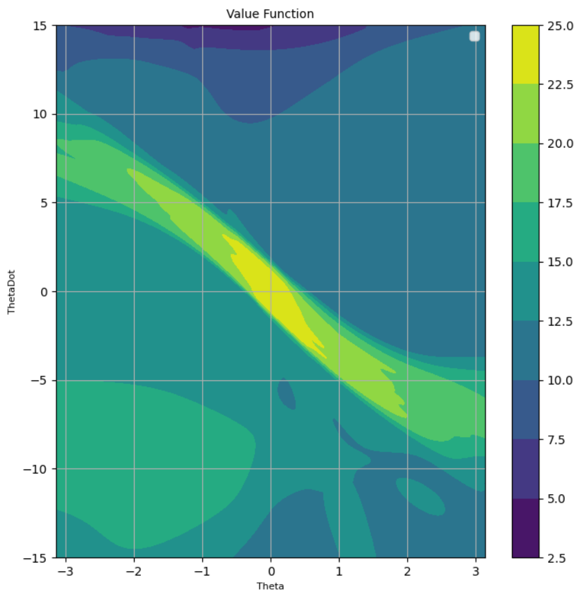

Fig. 13. DQN Values With Respect To State Without Replay With Target

Comparing Fig. 13 to Fig. 5, we seem a very similar structure but the maximum value region is stretched out diagonally so it is slightly larger, however, the value of this maximum value is larger. Once again, the author is unsure why exactly this would be the case. Perhaps this is an outlier, but perhaps the lack of replay forces the extreme convergence toward high values. Regardless, the values are almost identical. The smallest values however are not negative, another major difference between the Fig. 13 and Fig. 5.

D. *Without Replay and Without Target*

The final step for the ablation study is to determine the behaviour when there is no replay and without a target Q network. This means the replay buffer is the same size as the batch size and the target Q network is reset at every step. Below is the learning curve.

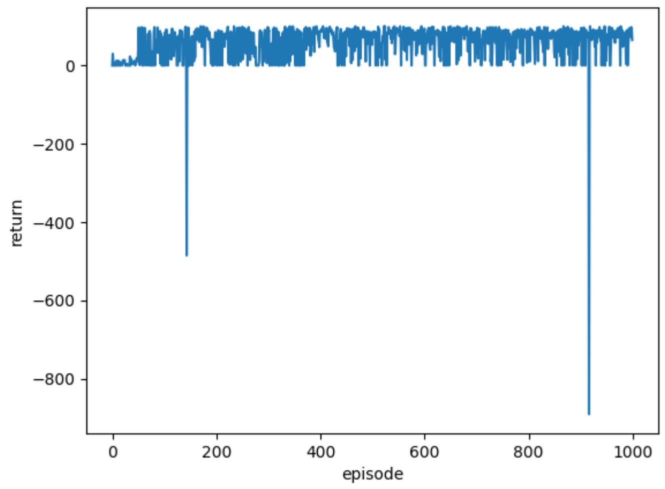

Fig. 14. DQN Learning Curve Without Replay Without Target

While this learning curve actually has the least amount of large negative outliers, it can also be seen that the noise never truly reduces, making the average return the lowest in the ablation study. As discovered through trends earlier, this is likely caused primarily by the lack of replay and exacerbated by the lack of a target Q network. Below is an example trajectory using this trained policy.

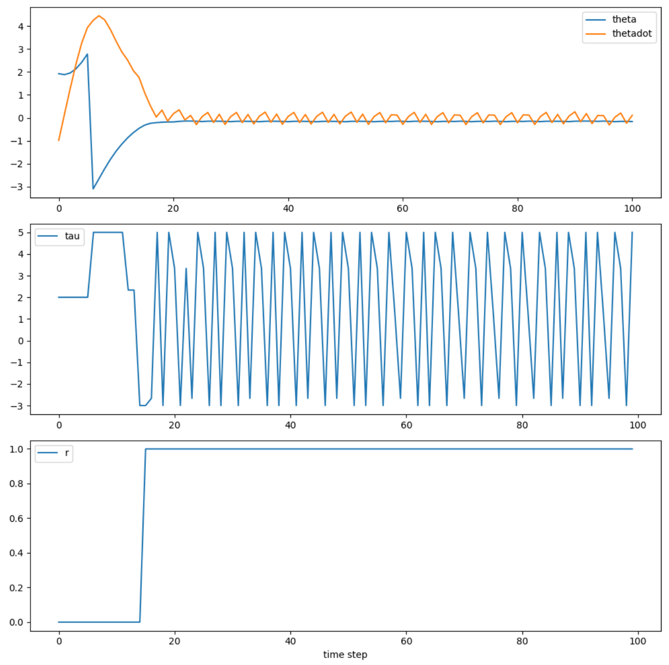

Fig. 15. DQN Example Trajectory Without Replay Without Target

Comparing Fig. 15. to Fig. 3, we see similar performance, with higher oscillation frequency and larger amplitude of oscillation. This suggest the policy learns to acts using very large actions. The policy graph below support this idea.

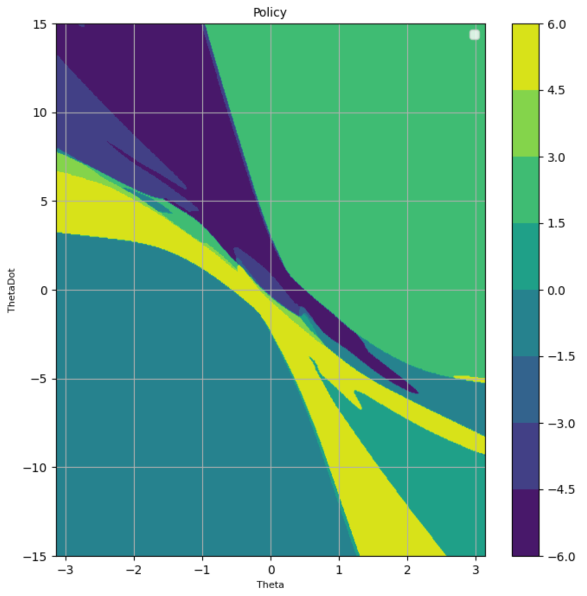

Fig. 16. DQN Policy Without Replay Without Target

This is a very interesting policy graph because it shows not much action unless it is within the diagonal hourglass region. In fact, it is really the only policy graph that is missing the more characteristic mirrored tanh-like region. It is highly non symmetrical but the region sizes are symmetrical, which once again makes it an outlier compared to the others. The policy appears to prefer large actions around zero-zero while smaller actions the farther away it is. This is again reflected in the value graph below.

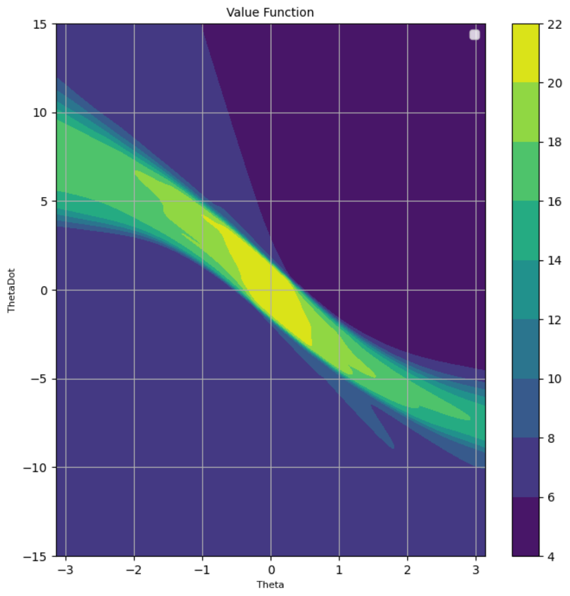

Fig. 17. DQN Values Without Replay Without Target

This is the simplest value function graph of all the agents in this ablation study. Essentially, if the agent is close to the target state and is moving toward the target state, output a large positive value, otherwise do not. The maximum value is smaller than the maximum for with replay and with target.

E. Ablation Study Results

The ablation study revealed that at least when looking at learning curves, the apparent best performance is unsurprisingly the standard DQN algorithm with replay with a target Q network. The example trajectories and value graphs revealed some interesting results that made it less obvious which algorithm is actually better. Perhaps this environment is too simply for noticeable large difference in deployment performance. All algorithms worked and allowed for a semi-stable result near the target state, and with the reward structure, the algorithm thinks that it is indeed at the target state.

III. Conclusions

This was a very interesting homework assignment into one of the ”classics” of deep reinforcement learning which I always wanted to pursue. DQN has impressive performance in past applications but it seems to be largely succeeded by policy gradient methods today. I am curious how and why so much of the focus has shifted away from Q learning to policy gradient methods, my initial guess would be because of continuous action spaces which seems like a severe limitation to Q-learning in general. I believe the algorithms are implemented correctly but am a little suspicious as to how well everything performed. As part of the results, gifs were included, and all of them appear to work great. Overall this was a great learning experience as I find implementation of these algorithms significantly increases my understanding of the algorithm itself.

Acknowledgment

The author would like to Acknowledge Professor H. Tran for the creation of the environment and programming setup, as well as for theoretical background knowledge regarding DQN. The author would further like to acknowldege PyTorch for their DQN tutorial and the work done by Dennis McCann which helped clear up what I was doing wrong.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[2] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” 2016. [Online]. Available: arXiv:1609.04836.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015) ”Human-level control through deep reinforcement learning”, Nature 518, 529–533. DOI: 10.1038/nature14236.