AE 598: RL – HW2 Report

Mahshid Mansouri (mm64)

***Abstract*—The goal of this assignment was to implement the DQN algorithm from [1] on an inverted pendulum problem with discrete action and continuous state spaces. This report introduces the problem, demonstrates the results, discusses the observations from the results, and provides some details regarding the algorithms implementations (e.g., the choice of the hyperparameters, etc.).**

I. Introduction

Q-learning is a model-free off-policy reinforcement learning (RL) algorithm which tries to learn the Q-values of different state-action pairs in an environment in a tabular setting where there's a finite number of state-action pairs [1]. DQN is a similar algorithm which combines deep neural networks with Q-learning and was first proposed by [2]. The main difference between Q-learning and DQN is that DQN uses a neural network (NN) to approximate the Q-function rather than using a lookup table. This is specifically useful in the case of systems with a continuous state space because Q-learning can only be applied to systems with a finite state space.

DQN has three main components: 1) An experience replay buffer which interacts with the environment to generate data to train the Q-Network, 2) a Q-Network which is essentially a neural network (NN) that takes the state as input and outputs the Q-values corresponding to all possible actions that could be taken from each input state, and 3) a Target network which is a copy of the Q-Network that is mainly used for stability purposes and its weights get updates every certain number of episodes during the training.

For this assignment, we implemented DQN for an inverted pendulum problem with a continuous state space but with discrete actions. Additionally, we performed an ablation study like "Extended Data Table 3" of [2] under different conditions to investigate the effect of including/excluding the experience replay buffer as well as the target network in the implementation. We will show and discuss the results in the following sections.

II. Results

- A plot of the learning curve for at least one trained agent (return versus the number of episodes or simulation steps).

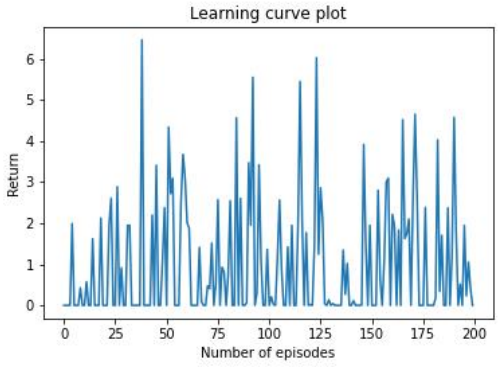

Fig. 1. Learning curve plot for case 1

- A plot of an example trajectory for at least one trained agent.
- A plot of the policy for at least one trained agent.

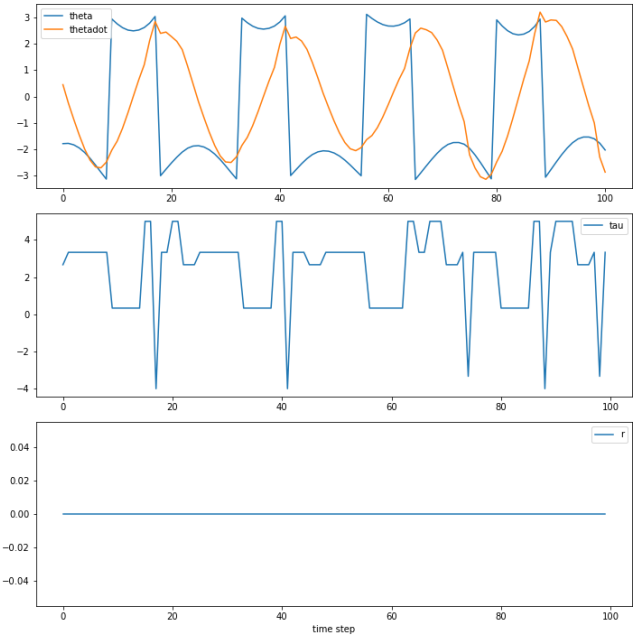

Fig. 2. Trajectory/policy plot for the trained agent for case 1

Note that for presenting the policy, we also use the trajectory plot (Fig. 2) as we believe using this plot, it's easier to understand how the pendulum states, i.e., θ and $\dot{\theta}$ as well as the action, i.e., τ should change over time to balance the pendulum.

- An animated gif of an example trajectory for at least one trained agent.

The animated gif is not shown here but has been uploaded to the "figures" folder on github.

- A plot of the state-value function for at least one trained agent.

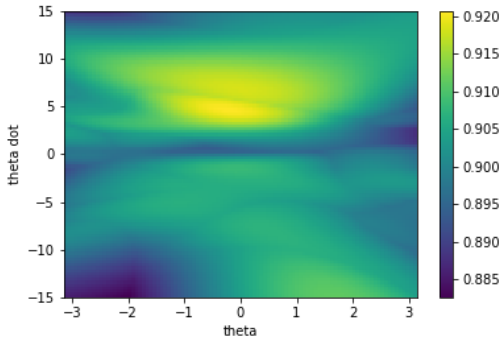

Fig. 3. State-value function for the trained agent for case 1

- Results for an ablation study, similar to what appears in "Extended Data Table 3" of Mnih et al (2015), under the following conditions. Note that all the parameters for all cases were kept the same and the only variables changed were two flags to turn on and off the replay as well as the target network.

Case 1: With replay, with target Q (i.e., the standard algorithm).

These results have already been presented (Fig. 1-3).

Case 2: With replay, without target Q (i.e., the target network is reset after each step).

- A plot of the learning curve for at least one trained agent (return versus the number of episodes or simulation steps)

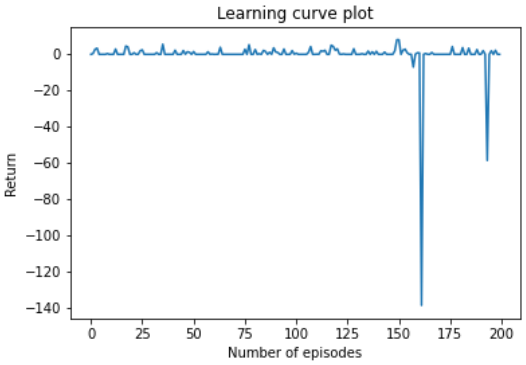

Fig. 4. Learning curve plot for case 2

- A plot of an example trajectory for at least one trained agent.
- A plot of the policy for at least one trained agent.

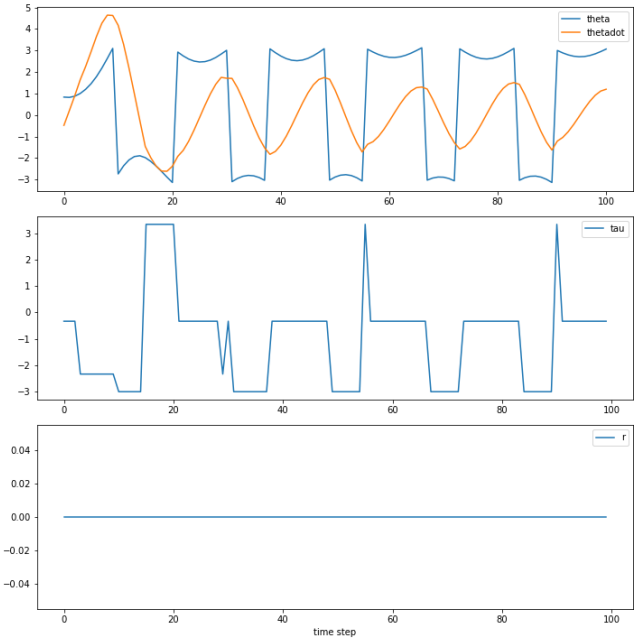

Fig. 5. Trajectory/policy plot for the trained agent for case 2

- An animated gif of an example trajectory for at least one trained agent.
See "figures" folder on github.

- A plot of the state-value function for at least one trained agent.

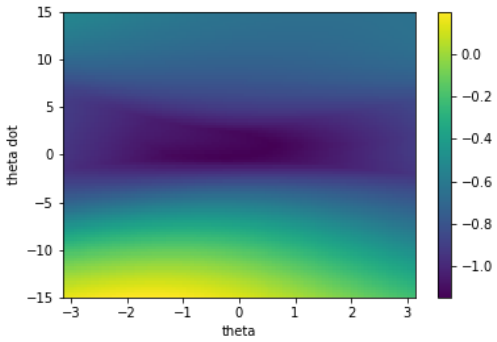

Fig. 6. State-value function for the trained agent for case 2

Case 3: Without replay, with target Q (i.e., the size of the replay memory buffer is equal to the size of each minibatch).

- A plot of the learning curve for at least one trained agent (return versus the number of episodes or simulation steps).

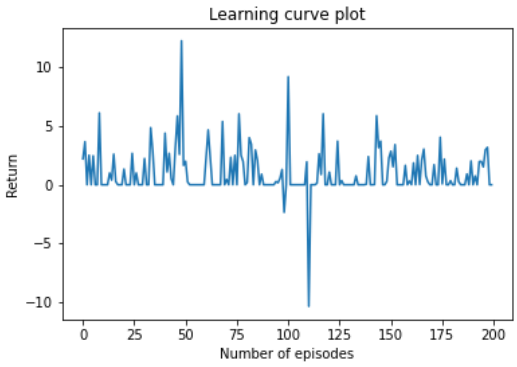

Fig. 7. Learning curve plot for case 3

- A plot of an example trajectory for at least one trained agent.
- A plot of the policy for at least one trained agent.

See Fig. 8 on the next page.

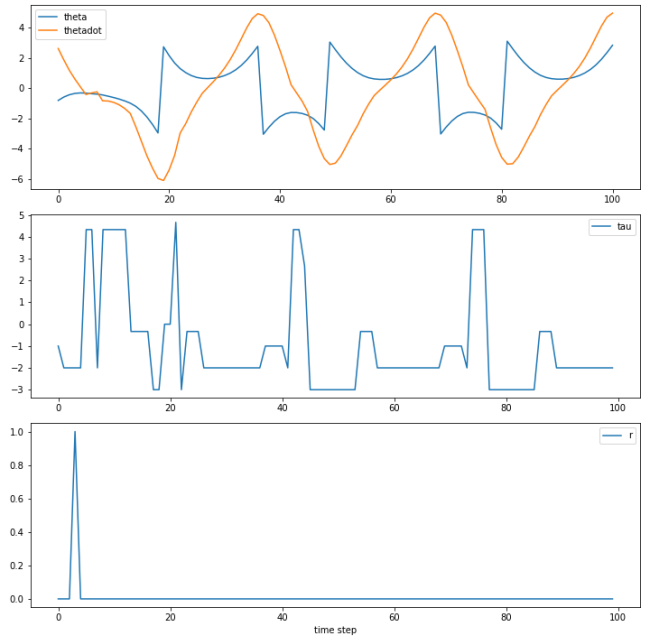

Fig. 8. Trajectory/policy plot for the trained agent for case 3

- An animated gif of an example trajectory for at least one trained agent.
  See “figures” folder on github.

- A plot of the state-value function for at least one trained agent.

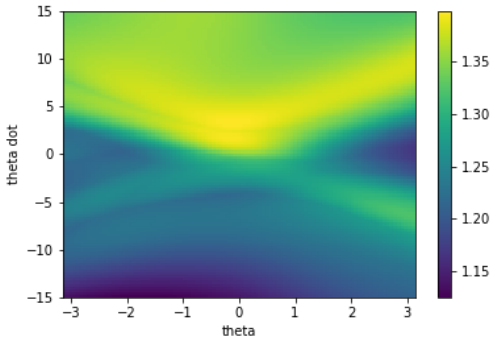

Fig. 9. State-value function for the trained agent for case 3

Case 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).

- A plot of the learning curve for at least one trained agent (return versus the number of episodes or simulation steps).

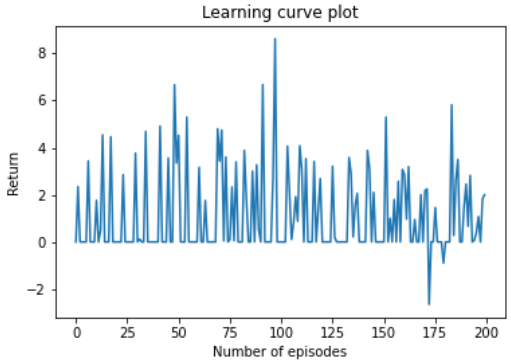

Fig. 10. Learning curve plot for case 4

- A plot of an example trajectory for at least one trained agent.
- A plot of the policy for at least one trained agent.

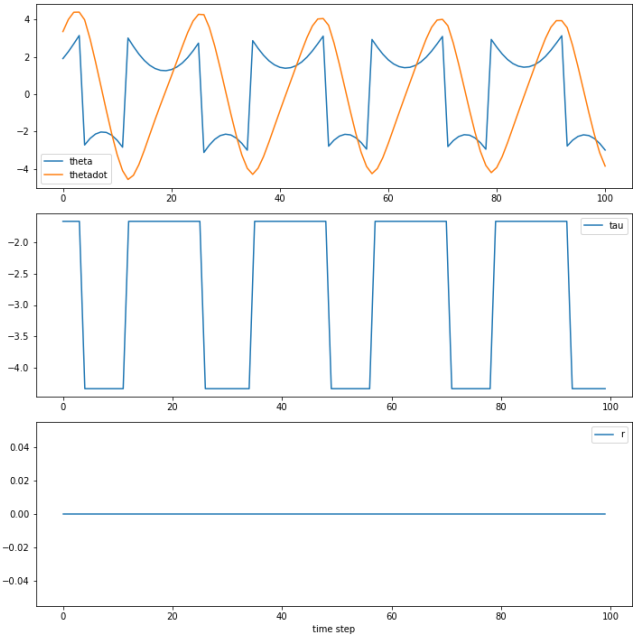

Fig. 11. Trajectory/policy plot for the trained agent for case 4

- An animated gif of an example trajectory for at least one trained agent.
  See “figures” folder on github.

- A plot of the state-value function for at least one trained agent.

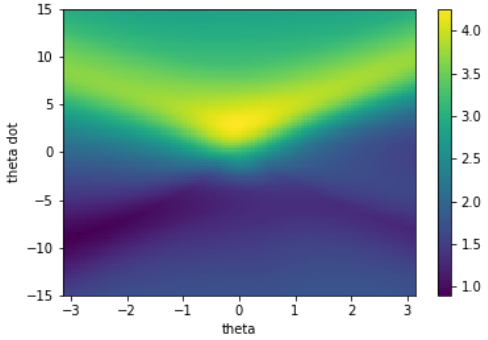

Fig. 12. State-value function for the trained agent for case 4

Plot of the learning curve for all four cases:

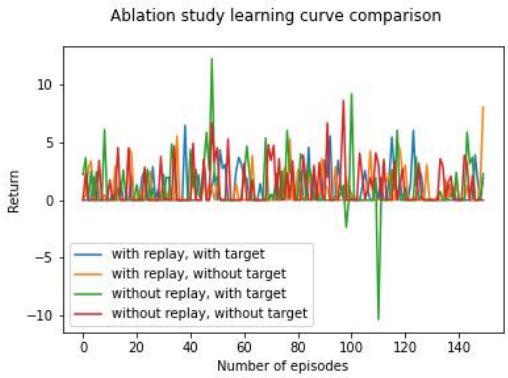

Fig. 13. Comparison of the learning curve of all four cases

III. Discussion

Case1: With replay, with target Q

This case is the standard DQN algorithm implementation. Ideally, I would have expected to see that the return in the learning curve plot starts small, but gradually increases and plateaus at almost a constant value as the network is trained well enough. However, the results that I got are different from this hypothesis and we observe that the return values never increase (Fig. 1). I tried a couple of different things that I thought might be causing this issue such as: 1) increasing the number of episodes to increase the learning time, 2) increasing the capacity of the replay buffer to generate more training data for training the neural network, 3) changing the target network update frequency to ensure that the target network weights are not getting updated too fast as if there was no target network in the implementation, etc. However, none of these methods worked out. Here are a couple of thoughts that I have that might be causing this problem

Additionally, in the ideal scenario, regarding the trajectory and policy plots for this case, I would have expected to observe that the agent tries to keep θ around zero to balance the pendulum with small oscillations of $\dot{\theta}$ around zero after a certain amount of time. Moreover, the action values or τ should have been changing from positive values to negative values to prevent the pendulum from falling. Lastly, the reward should have stayed zero for a certain time, but after enough training, it should have jumped to 1 which is the maximum reward that the agent can achieve. However, my results show that the pendulum is not balanced and keeps moving away from the equilibrium point (Fig. 2). It also never achieves a reward of 1.

Regarding the animation of an example trajectory, we should observe that the pendulum is kept upright and oscillates around the equilibrium after a certain number of episodes once the agent is well-trained, which is not seen in my current results.

Lastly, in the ideal case, the learned state-value function should have the highest value around $[\theta, \dot{\theta}]$ = [0,0] which is the pendulum equilibrium where the agent gets the highest reward. However, since my algorithm is not working properly, we do not see such a behavior, rather it seems like the value function has the highest value in a different region (Fig. 3).

Case 2: With replay, without target Q

In this case, compared to case 1, there's no target network which means that the weights of the target network get updated at every time step to the weights of the Q-Network. Ideally, we would hypothesize that excluding the target network might cause network stability issues. This is mainly because the target network is used for stability of the neural network, and its weight get updated every certain number of episodes. As we know, in Q-learning, we are trying to minimize the error between the target network and the Q-Network outputs (i.e., the TD error term). If there's no target network, meaning that we keep updating the target network weights at every step to be the same as the Q-Network, it means that both networks keep changing, or in other words, we are chasing a moving target which might cause instability in the training. However, if we use the target network and keep updating its wights every often, the network will be fixed for a certain time and only the Q-Network keeps changing. This way, the network will have higher stability.

Since I was not able to get the standard algorithm to work, the results are not satisfactory here (Fig. 4-6). There are two large drops of the return value in the learning curve plot (Fig. 4) showing that the agent is not well-trained. Additionally, the agent is not able to balance itself as observed in Fig. 5. Moreover, the learned state-value function have negative values which means that the neural network is not trained properly which has resulted in wrong network output values. It is seen that the state-value function plots has the minimum value around the equilibrium point which is contrary to what we would have expected to observe.

Case 3: Without replay, with target Q

In this case, the replay buffer is excluded which means that the capacity of the replay buffer is the same as the batch size implying that we are using all stored transitions in the buffer as the training data in the Q-Network. As we know, the basic idea behind the experience replay is to generate a set of data for training the Q-Network. Since we are using an SGD optimizer in the training phase, we must ensure that the training data is independently and identically distributed and when the agent interacts with the environment, the sequence of experience tuples can be highly corelated. To prevent the Q-Network outputs from oscillating or diverging catastrophically, we must use a large buffer of the agent's past experiences, and sample training data from that instead of using the latest experience. In the case of no replay, there might be high correlations between the sampled data which might cause the neural network not to learn well enough using the provided data and performs poorly on a new input data.

As explained before, the results are not satisfactory for this case either (Fig. 7-9). One interesting observation is that we see the reward jumping to 1 in Fig. 8 but then going to zero.

Case 4: Without replay, without target Q

This case is a combination of case 2 and 3 and which might result in some stability issues in the network as well as highly correlated data used as the training data in the network, resulting in poor performance of the network on a new set of data. Again, the results for this case are not what I would have expected to see (Fig. 10-12).

Ablation study results:

Fig 13 compares the performance of the algorithm for cases 1-4. We cannot draw any conclusions from this figure as we know that the standard algorithm (case 1) is not working properly, thus other cases. However, ideally, I would have expected to see the learning curve for case 1 to have the highest value once the network is well-trained as this case takes advantage of both the replay buffer and the target network in the implementation, which is supposed to has the best performance. By excluding the target network while keeping the replay buffer (i.e., case 2) there might be some stability issues in the neural network training phase which might affect the learning plot convergence (i.e., not converging to the expected return value, etc.). Excluding the replay buffer while keeping the target network (i.e., case 3) might result in high correlations in the training data for the neural network. Thus, I would expect to see the learning curve of case 3 to be lower compared to case 1 meaning that the final output return score might be lower because the network will not be trained properly to improve and make better predictions if it uses highly correlated training data (not enough exploration). Lastly, excluding both target network and experience replay is expected to result in the lowest final return compared to the other cases because of the issues that excluding each condition will cause.

General suggestions for improving the algorithm performance:

One idea for future debugging of the code would be to plot the loss of the Q-Network in the optimization phase over time and see how it looks. Based on the loss plot, there might be a couple of different reasons that could explain such a performance, and potential solutions could be investigated. Changing the learning rate in the optimizer might also help. More training of the network (larger number of episodes) might also be helpful.

REFERENCES

[1] “Reinforcement Learning: An Introduction” by Sutton and Barto (MIT Press, 2018

[2] 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.