AE598 HW2: Deep Q Network (DQN) for Inverted Pendulum

Niket Parikh [*]
University of Illinois Urbana-Champaign, Urbana, Illinois, 61801

This report summarizes the problem solved in the HW2, mentions the specific implementation details needed to understand the algorithm implementations and presents the results followed by a brief discussion.

I. Introduction

This report encapsulates the work done for HW2 and presents the results. The aim of the work is to implement the DQN algorithm [1] hold a simple pendulum upright. Section II presents the problem description and modeling of the environment, Section III discusses the implementation details, in particular, the non-trivial choices made in the process of writing this code. Section IV puts forth the experimental results, along with a brief discussion and section V gives concluding remarks.

II. Problem Setup

The environment is a continuous state space, discretized action space simple pendulum for which an explicit transition model is not known. The actions consist of torque values that can be applied to the pendulum mass while the state is a vector containing in each of its components, pendulum angle and angular velocity respectively. The agent is rewarded with a small positive value for staying upright between a specific threshold of pendulum angles, a high negative reward for exceeding an angular velocity threshold and zero reward otherwise.

The problem is expressed as a Markov Decision Process with infinite horizon and discount factor, $\gamma = 0.95$.

III. Deep Q Network (DQN)

This section first discusses the algorithm and then highlights the design choices (such as hyperparameter values) made during implementation. The value of discretizing parameter for the environment's action space is $n_{actions} = 31$.

A. Algorithm Details

The algorithm considered is the vanilla DQN proposed in [1]. The underlying idea is to perform Q-learning-like updates for the agent in an unknown environment and, unlike Q-learning, in a function approximation setting. This allows us to work with continuous state space, unlike tabular Q-learning, which required us to discretize the same environment in AE598 HW1. DQN utilizes feedforward neural networks to approximate the state-action value function. While the idea of using neural networks (NN) to approximate value functions existed before the introduction of DQN as well, the techniques thus far were underwhelming in terms of performance and stability. DQN, with the introduction of experience replay and target networks, managed to overcome the unique issues of non-stationarity of target and high correlation in training data, and proved to be a tractable solution for Q-learning approximated by NNs. While better techniques have emerged since then, DQN remains a capable algorithm for a variety of problems.

B. Implementation Details

The implementation is standard, with the following design choices to be noted. The deep Q network (Q-net) is initialized at the start of the experiment with random weights. The replay buffer, until it reaches the size of a training minibatch, is filled by taking steps driven by actions sampled from an epsilon-greedy policy based on this Q-net and weights are not updated during this time. This is unlike the article introducing DQN where the replay buffer, until it reaches a size much larger than the minibatch, is filled by taking steps driven by actions sampled randomly from the action space. While the latter is indeed useful in various scenarios, including when the Q-net is costly to evaluate and store, this is not an important consideration for the problem at hand. The parameter of the epsilon-greedy policy is decayed exponentially as the training progresses.

[*]PhD Student, Aerospace Engineering, University of Illinois Urbana-Champaign

IV. Results

Table 1 gives the values of algorithm parameters chosen for the experiments. To analyze the performance of the algorithm, the following plots have been recorded - learning curves (return versus number of episodes), contours for the policy and state-value function for an example trained agent, state, action and reward trajectories of an example trained agent and lastly, an ablation study where the standard DQN is compared against three other baselines, where either one or both the target Q-net and replay buffer features of the standard DQN are turned off. A visual rendering of the simple pendulum has also been recorded and it shows that the pendulum indeed stays upright, as desired.

Given the variance in results from one training run to another, the quantities presented have been averaged over several training runs, with the mean value plotted along with an error region of one standard deviation.

Table 1 Parameter values used to obtain the computational results

Batch Size	32
Replay Buffer Length	100000
Rate of Target Network Update	500
Starting Value of ϵ	0.9
Minimum Value of ϵ	0.05
\epsilon Decay Parameter	1000
Number of Episodes	400
Number of Training Runs	10
Optimizer	RMSprop
Optimizer Learning Rate	0.00025
Loss Function	MSELoss

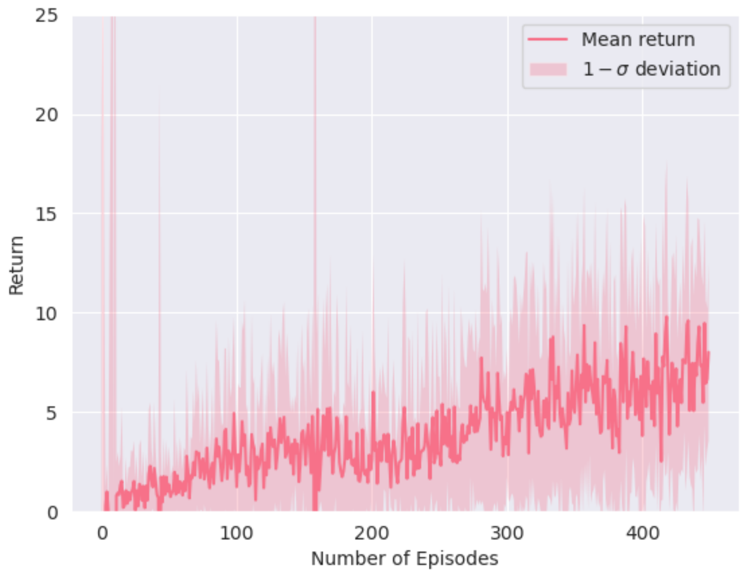

Fig. 1 Learning Curve for standard DQN

Figure 1 shows the averaged (over training runs) learning curve for the DQN agent. It is clear that the trend is upward and the agent is learning successfully.

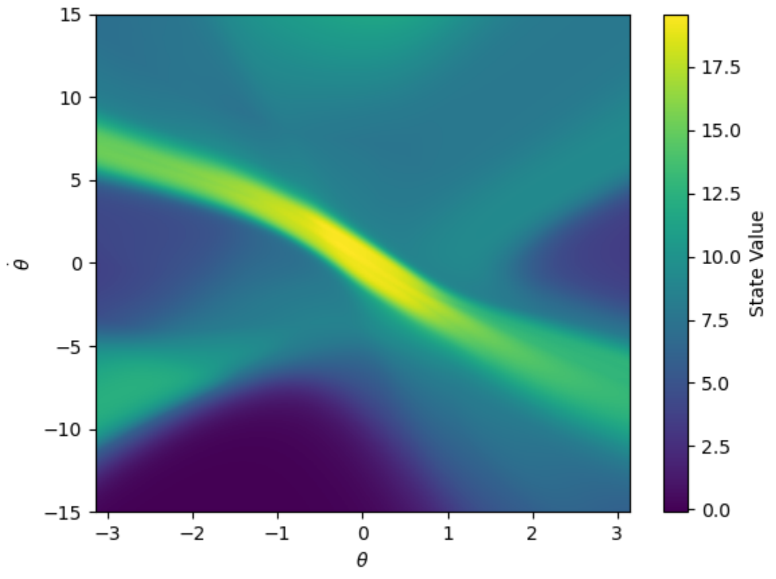

Fig. 2 Contour of the state-value function for an example trained agent

Figure 2 shows the state value contour. At a high level, the overall trend can be reasoned easily and the values do pass sanity checks. The stationary state with zero velocity and zero degrees angular position, i.e., upright, has the highest value, as desired from the rewards. The values in second and fourth quadrant (second quadrant is top left) are relatively high because they move the pendulum towards upright position while in the other two quadrants, the tendency is to move away from the upright position, thus making them less valued.

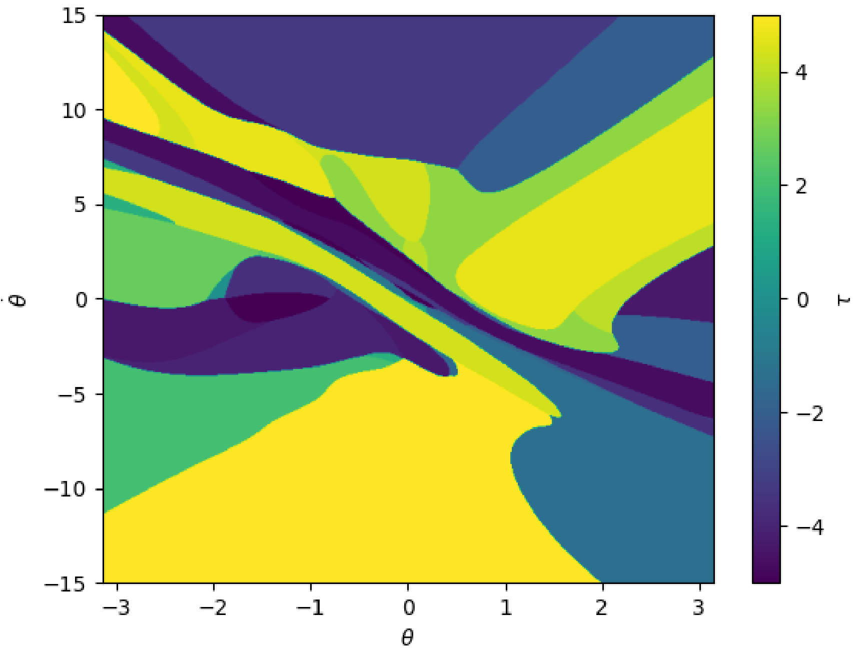

Fig. 3 Contour of the policy for an example trained agent

Figure 3 shows the policy plot, which again can be reasoned out in a similar manner as above, where the direction of torque applied and magnitude are such that the pendulum moves towards upright state.

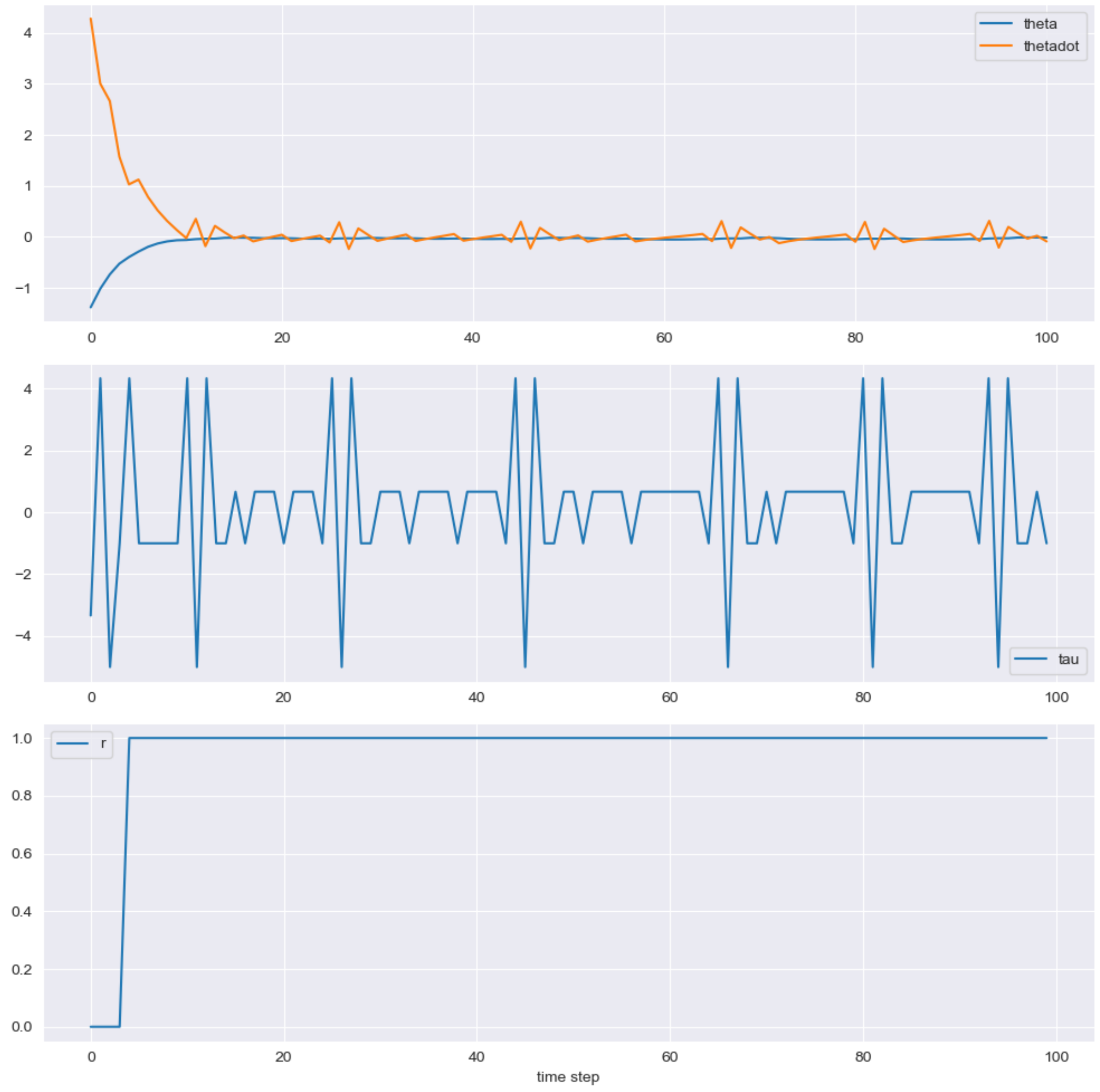

Fig. 4 Trajectories of the state, action and reward for an example trained agent

Figure 4 shows the trajectory plots wherein it is clear that the pendulum indeed seems to stay near zero degree angular position, thus meeting the problem requirements and seeing the highest possible reward for all times except an initial learning phase. Torque constraint is also not violated in this case, which is welcome because constraint satisfaction is as such, not guaranteed by DQN.

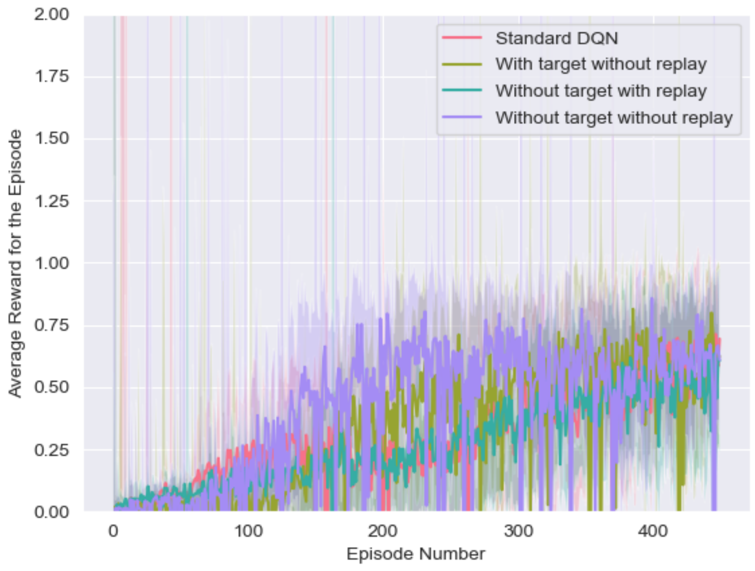

Fig. 5 Learning curves (with respect to average reward per episode) for standard DQN and other baselines

The ablation studies (figures 6 and **??**) looked at the averaged learning curves for each compared algorithm with the vertical axis containing return in one case, and average reward per episode in the other case. It is not easy to see a clear demarcation between the performance, which is unexpected, but perhaps down to either inadequate training, tuning or selection of the metric to be plotted. However, in the curves it is noticeable that the general trend for standard DQN is upward in both cases whereas for other methods the behaviour is highly varied and not as stable.

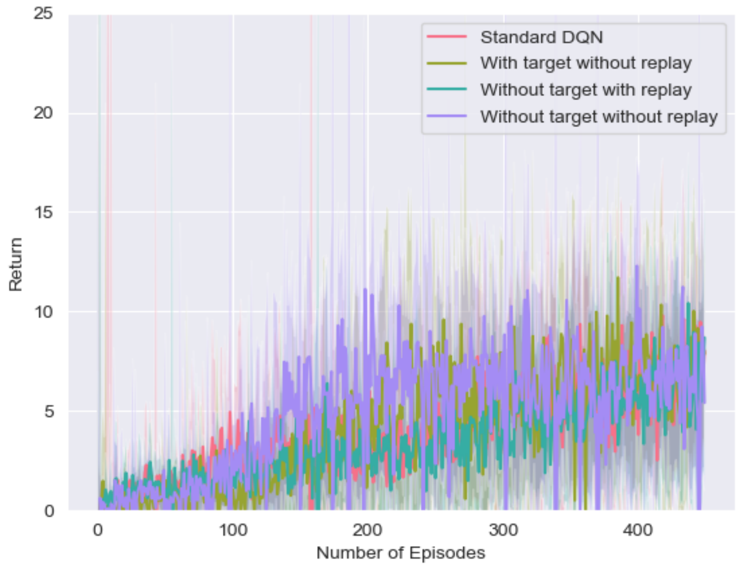

Fig. 6 Learning curves (with respect to return per episode) for standard DQN and other baselines

V. Conclusion

In conclusion, DQN has proved to be successful in this problem, as shown by the results. There is scope, however, for the results to be more refined.

Acknowledgments

The author is grateful to the course instructor, Prof. Huy Tran, for providing codes for the environment's Python classes.

References

[1] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533.