Deep Reinforcement Learning (CS 545)
Assignment Report
Assignment-2: Discretization for Deep Q-Learning

Osman Furkan Kınlı – S002969

Problem definition:

In this assignment, deep Q-learning approach is implemented in three different concepts to solve *MountainCarContinuous* (MCC) problem. The environment for MCC problem can be found on gym. Even though this environment has same constraints with earlier version of MCC, *MountainCar* (MC) which has discrete action space {-1, 0, 1}, it has continuous action space ranged between -1 and 1. The basic deep Q-networks (DQN) are not applicable to this task since the action space is not discrete, so we have to discretize the action space to feed the DQNs properly. By the help of this technique, we can use DQNs to solve MCC problem.

The main target for this task is that a car must climb a one-dimensional mountain to reach the target on the right-top of the mountain. Unlike MC problem, the engine force applied is allowed to be a continuous value. The car is pushed to the left with negative values, while it is pushed to the right with positive values. If the car reach the goal, the episode terminates. Reward for reaching the goal on the right-top of the mountain is 100. At the end of the episode, the agent should gather at least 90 reward to fulfill the solving requirement.

Solution approach:

To solve the MCC problem, we use deep Q-learning networks which give a particular number of possibility for all actions in action space. However, in this problem, the action space can have infinite number of elements in a range of -1 and 1. We consider to discretize the action space with a certain number of elements in between -1 and 1 in order to train our agents in a continuous space.

Discretization technique:

The action space is discretized in 9-step way such as {-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1}. The value of an action in a certain state is approximated to the closest element in discretized action space before acting.

Moreover, we have three different concepts of DQNs. These concepts are proposed by Google DeepMind to stabilize the learning phase so that the agents can learn better. These concepts follow:

1. Vanilla DQN
2. Experience Replay DQN
3. Experience Replay with Target Network DQN

Implementation of these networks is done with Keras in Python 3.5.X environment.

Model files:

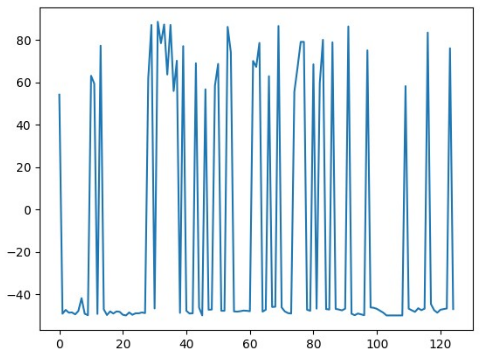
1. Vanilla DQN → “Furkan_Kinli_1.h5f”
2. ER DQN → “Furkan_Kinli_2.h5f”
3. ER with TN DQN → “Furkan_Kinli_3.h5f”

Results:

WITH A MISTAKE PART:

training phase:

Illustration 1: Vanilla DQN



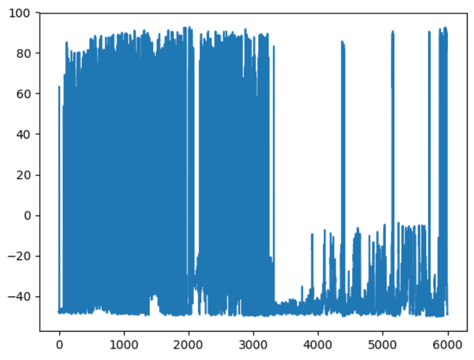


Illustration 2: Replay DQN

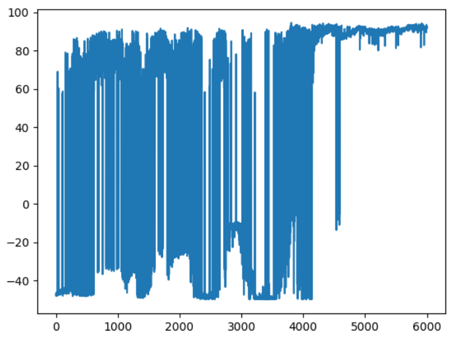


Illustration 3: Replay with Target DQN Agent

testing phase:

The results (average and maximum of total rewards in 100 episodes without training) after 10 runs in testing phase can be seen on the table below.

CONCEPT	AVG REWARD	MAX REWARD
Vanilla DQN	9 ± 18	27
ER DQN	24 ± 5	29
ER with TN DQN	32 ± 3	35

WITH STUCK PART:

training phase:

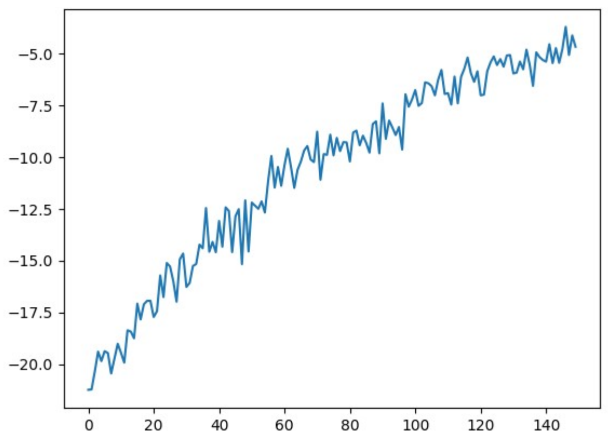


Illustration 4: Vanilla DQN

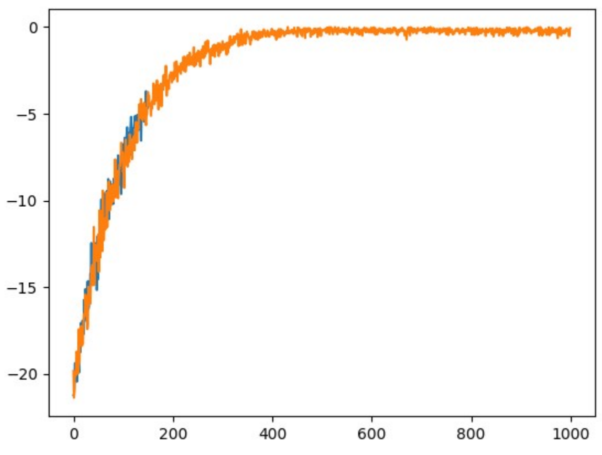


Illustration 5: Experience Replay DQN

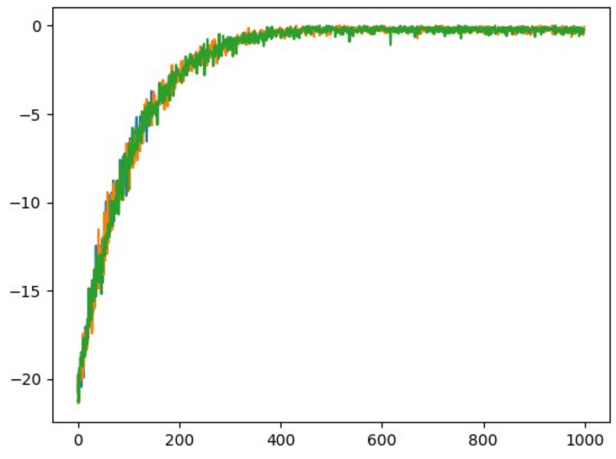


Illustration 6: Experience Replay w/ Target Network

With the first implementation trail, the agent learns how to achieve the goal in long run, and in testing, the performance is not close to training. The mistake is about the data structure of getting action. With the second implementation trial, the agent tries to learn to stop at low-ground. Therefore, **the results of 2nd trial converge to 0**, mostly in testing. As TA Emir knows, I tried my best, but I could not solve where the mistake is. That's all what I can do. Thank you.

P.S: All codes have been implemented from scratch.

Thanks to our TA Emir for giving clear intuition about discretization techniques and implementation of deep Q-learning models in his office hour.