DSAI Hw4

F74036409 林京樺

Questions:

1. What kind of RL algorithms did you use? value-based, policy-based, model-based? Why?

Mountain-car is a problem that needs to reference experience. To catch the flag, mountain-car needs to decide which action should take (to left, right, stay), and it depends on the previous observations(position and state) also the experiences of wallking. Thus I choose **Deep Q Learning** which is a **value-based** algorithm.

I've tried Q Learning at first, but there's too much possibilities in Mountain-car problem. This will make Q table become really large that will bring a massive overhead to memory. So, DQN is a solution to deal with big table size of Q learning, and the key point is to use Neural Network.

2. This algorithms is off-policy or on-policy? Why?

Deep Q Learning is an **off-policy** algorithms since it learn from previous experiences and keep update the decision with reward to meet the best solution.

3. How does your algorithm solve the correlation problem in the same MDP?

The design of two neural network in DQN could lower down the correlation problem.

Deep Q Learning:

Deep Q Learning is a great solution to Q learning. Q learning is an off-policy algorithm which depends on a Q table to record all history states. One of the most significant disadvantage is what if we got millions of states in the be-solved problem. Due to the limitation of hardware, we couldn't store such large table in any situation.

Deep Q Learning then solve this disadvantage with neural network.

Main component:

- memory: for training NN
- NN-predict: to learn and get next action
- NN-target: to learn and get more precise action

Define reward:

In reinforcement learning, reward is a key element to lead NN toward a better choice in situations. Thus, I spent a lot of time to figure out how to define a better reward that could encourage NN make a good decision in mountain-car problem.

```
# update reward

# if move forward and the velocity is the same direction will get higher reward
reward = (observation_[0]-observation[0]) * observation_[1]
```

```
# higher reward if the car is on the right of the hill => closer to the flag
if observation_[0] > -0.5:
        reward +=  3 * abs(observation_[0] - (-0.5))
# higher reward if the car is on the left of the hill => got more energy
else:
        reward += abs(observation_[0] - (-0.5))
```

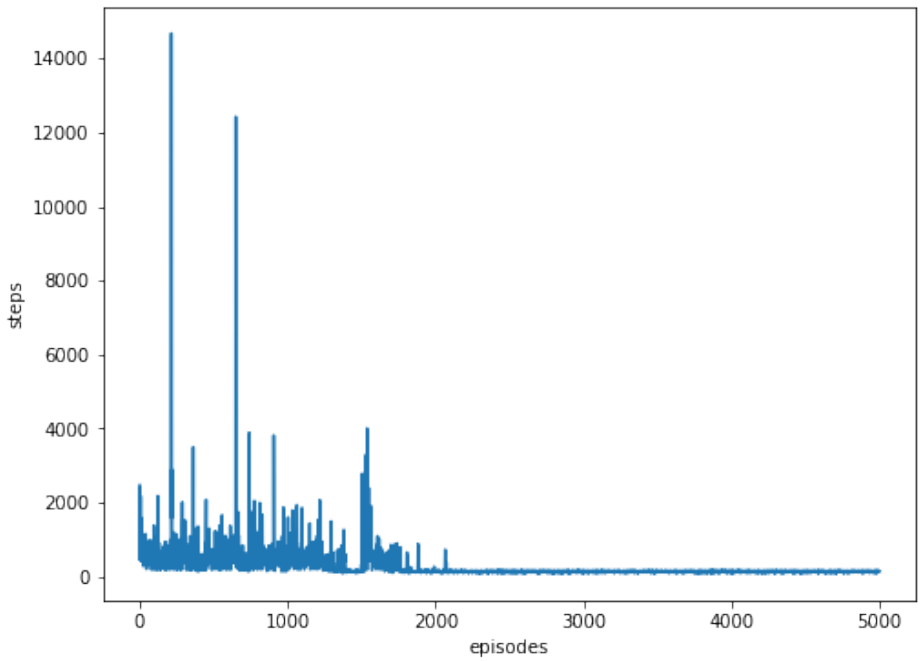
Experiment Result:

First try:

5000 episodes, the steps decreases in the iteration. After 2000 episodes rounds, the results is about to convergence:

parameters:

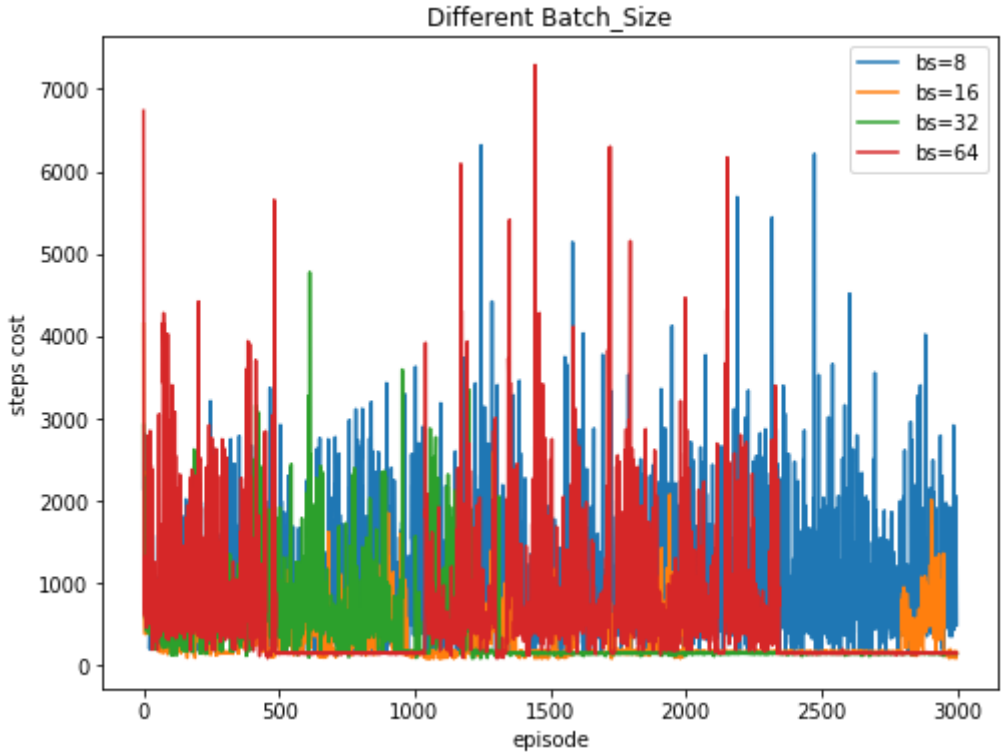
- learning_rate: 0.1
- epsilon=0.9
- batch size: 128



Batch size analysis

Below is a graph showing batch size=8, 16, 32, 64
for each batch size, retrain model for 500 times.

The result shows that batch size = 16 got the best performance



Batch size	Mean	Min steps
8	887.6	173
16	**320.4**	**84**
32	351.8	87
64	588.1	86

Learning rate analysis

for batch size = 16, I've tried different learning rate.
We can see when learning rate is 0.001, we got the best performance.