Machine Learning Report of Proj4

Haorui Dong
UBIT number: 50291149
Department of Computer Science and Engineering
University at Buffalo
haoruido@buffalo.edu

Abstract

In this project, we using reinforcement learning and deep learning to solve a problem that to find a shortest way which agent could reach terminate. We used Q-learning in this project. And we use Deep Q-Network(DQN) to set up our model. The TA offer the most part of code, and we just implemented a few important part of code.

NOTE: **report** and **writing task** are included in this document!

1 Introduction to Q-learning and DQN

1.1 Q-learning

The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. It does not require a model of the environment and can handle problems with stochastic transitions and rewards, without requiring adaptations.

Q-learning is one of reinforcement learning. Reinforcement learning involves an agent, a set of state, and a set of actions per state, the agent transitions from state to next state. And agent will execute an action in a specific state, then it will be provided a reward. The goal of the agent is to maximize its total reward. In Q-learning, it has a Q-value table, and during training, the value in the table will change, and finally it can get a table which can lead agent get the maximum rewards.

Initialize replay memory to capacity N

Initialize the environment (reset)

For episode $= 1, M$ **do** (Begin a loop of interactions between the agent and environment)

Initialize the first state $s_0 = s$

For $t = 1, T$ **do**

With probability ϵ select a random action a_t, otherwise select $a_t = argmax_a Q(s, a; \Theta)$

Execute action a_t and observe reward r_t and next state s_{t+1}

A tuple $< s_t, a_t, r_t, s_{t+1} >$ has to be stored in memory

Sample random minibatch of observations (s_t, a_t, r_t, s_{t+1}) from memory

Calculate Q-value

$$Q_t = \begin{cases} r_t, & \text{if episode terminates at step } t+1 \\ r_t + \gamma max_a Q(s_t, a_t; \Theta), & \text{otherwise} \end{cases}$$

Train a neural network on a sampled batched from the memory

End **For**

End **For**

Figure 1: Q-learning pseudocode

During Q-learning, it will update the value of Q-table, and use the new value in Q-table to decide actions in a specific state.

1.2 DQN

In some tasks, using Q-table to save Q values is not practical. Because the problem is too sophisticated, and the number of state will be vary large. If we still use table to save them, we do not have a computer which have enough memory. Also it will take lots of time when we search a value in a huge table.

Thus, we use neural network combine Q-learning to solve complex problems. We let states and actions as input of neural network, and use neural network to get a Q value.

2. Coding task

2.1 Build neural network using keras lib

I use model.add() to add layers of my neural network. Depends on description of project4, the activation function of first and second hidden lays should be 'relu'. And the dimensions of first input layer should be the size of my observation space (state_size). The number of nodes for these two layers should be 128. Finally, the output layer should use 'linear' as its activation function. And its output size should be the same as the size of the action space (action_size).

```
### START CODE HERE ### (≈ 3 lines of code)
model.add(Dense(128,input_shape=(self.state_dim,), activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(self.action_dim, activation='linear'))


### END CODE HERE ###
```

Figure 2: Code to Build Neural Network

The job of neural network is introduced before. In a nutshell, we use neural network to solve the problem that Q-table is not suitable in every situation.

2.2 Implement exponential-decay formula for epsilon

Exponential-decay formula for epsilon:

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda|S|},$$

where

$\epsilon_{min}, \epsilon_{max} \in [0, 1]$

λ - hyperparameter for epsilon

$|S|$ - total number of steps

Figure 3: Exponential-decay Formula for Epsilon

I used python to implement this formula the lambda and the |S| is available for us to use.

```
### START CODE HERE ### (≈ 1 line of code)
self.epsilon = self.min_epsilon + (self.max_epsilon - self.min_epsilon) * exp( -( self.lamb * abs(self.steps)))
### END CODE HERE ###
```

Figure 4: Code for Implement Formula for Exponential-decay of Epsilon

To finish the formula, I import math module of python to help me. There are two useful functions in math module, they are exp() and abs(). I use them to calculate $e^{-\lambda|s|}$.

This part of codes will update the value of epsilon. At beginning, the epsilon is 1.0. That means all actions are random, and the reward will not influence choosing actions. During the training, the epsilon will decay to a value that we have set. (The origin code is 0.01). The smaller epsilon means less random actions. Because at beginning, the neural network is not well trained, so the reward is not correct, we should let our model try all choices to get the best solve. If we do not have epsilon, the model will not try another available choice which maybe a better one. And if we always have a big epsilon, it will still do random actions although there is a correct reward and neural network.

2.3 Implement Q-function

$$Q_t = \begin{cases} r_t, & \text{if episode terminates at step } t+1 \\ r_t + \gamma max_a Q(s_t, a_t; \Theta), & \text{otherwise} \end{cases}$$

Figure 5: Calculate Q-value

When next step is none, target is reward. Else target is reward + gamma * maximum of next Q-value.

```
### START CODE HERE ### (≈ 4 line of code)
if st_next is None:
  t[act] = rew
else:
  t[act] = rew + self.gamma * np.max(q_vals_next[i])



### END CODE HERE ###
```

Figure 6: Code of Calculate Q-value

This part of code will change the Q-value of neural network. And it will be involved in iteration to train our neural network.

I think the neural network is complex enough to finish this task. But the decay for epsilon is too slow. We can increase the speed of this decay. I will do some test in later part.

3. result and tune parameters

3.1 result

With the code offered by TA, we can get plot of epsilon and reward. The epsilon is decayed from 1.0 to 0.01 as our setting. And the reward increased from about 0 to about 8.

```
----------
Episode 9500
Time Elapsed: 755.70s
Epsilon 0.062358721208661684
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.225720668014041
----------
Episode 9600
Time Elapsed: 763.15s
Epsilon 0.06184211191317221
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.2382907062414485
----------
Episode 9700
Time Elapsed: 770.62s
Epsilon 0.06134539556578169
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.251640454119363
----------
Episode 9800
Time Elapsed: 777.90s
Epsilon 0.06088147698589778
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.26605504587156
```

Figure 7: Iteration

It is obviously that, at last, the reward is 8, and epsilon decay to 0.06. This iteration takes about 790s.

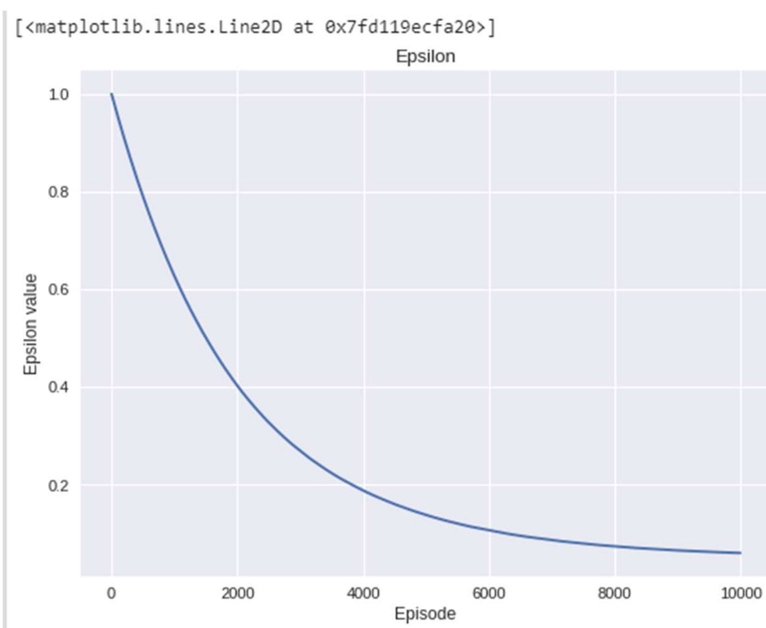

Figure 8: Curve of Epsilon

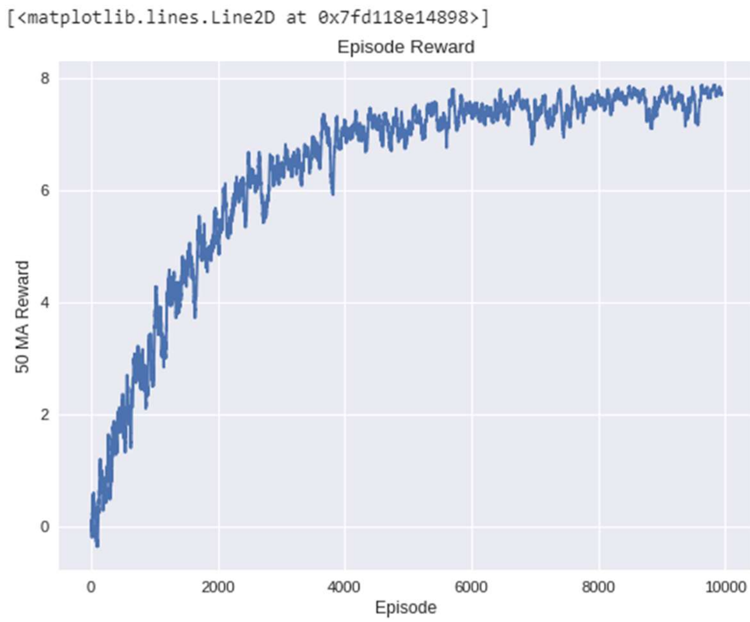

Figure 9: Curve of Reward

3.2 Tune parameters

Following the tasks from project document, I change some parameters value.

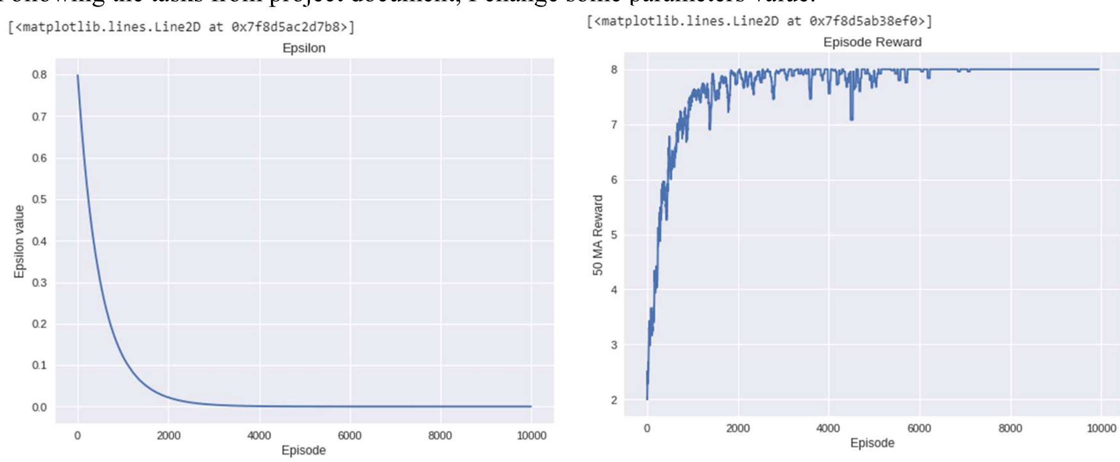

Figure 10: Curve of First Try

The result shows that at about 3000 episodes the epsilon is 0, and the reward is almost 8. After about 6000 episodes, the reward is almost always 8. Under these parameters condition, the model get a higher speed to learning. I think 3000 episodes is enough to reach the result.

I think we need higher speed of decay for epsilon and less episodes. So I change Lambda to 0.0002, and I discovered that about 3500 episodes is enough, so I do some test follow table 2.

	Max_epsilon	Min_epsilon	Lambda	Episodes
Test1	1	0	0.0002	3500
Test2	0.5	0	0.0002	3500
Test3	1	0.5	0.0002	3500

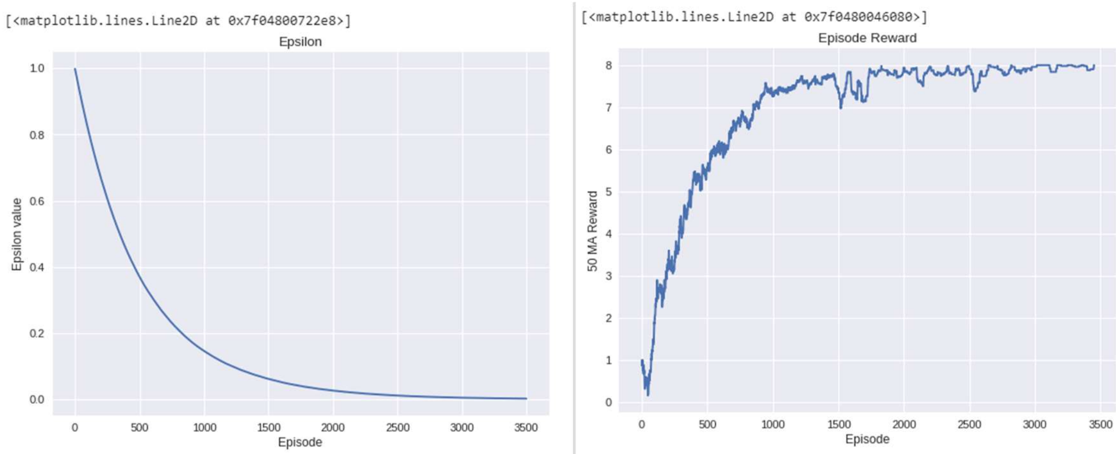

Figure 11: Curve of Test1

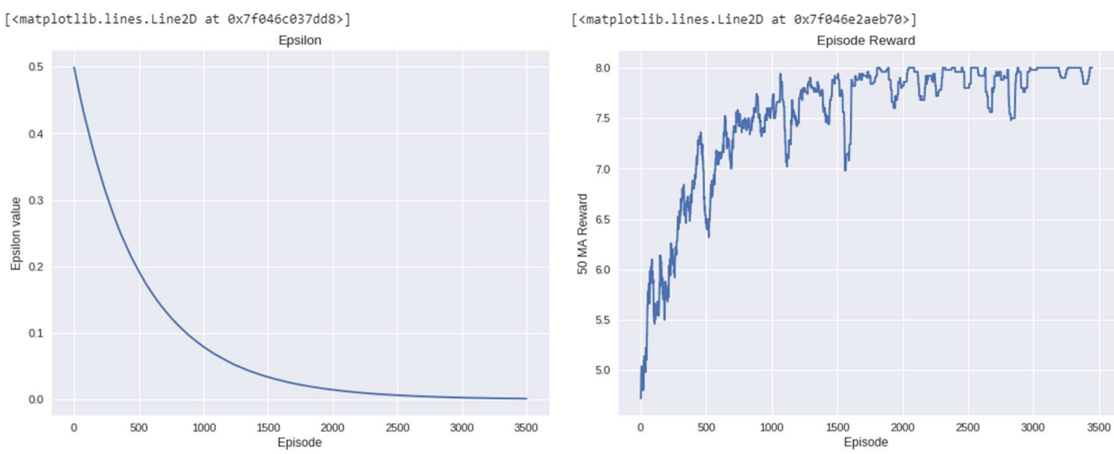

Figure 12: Curve of Test2

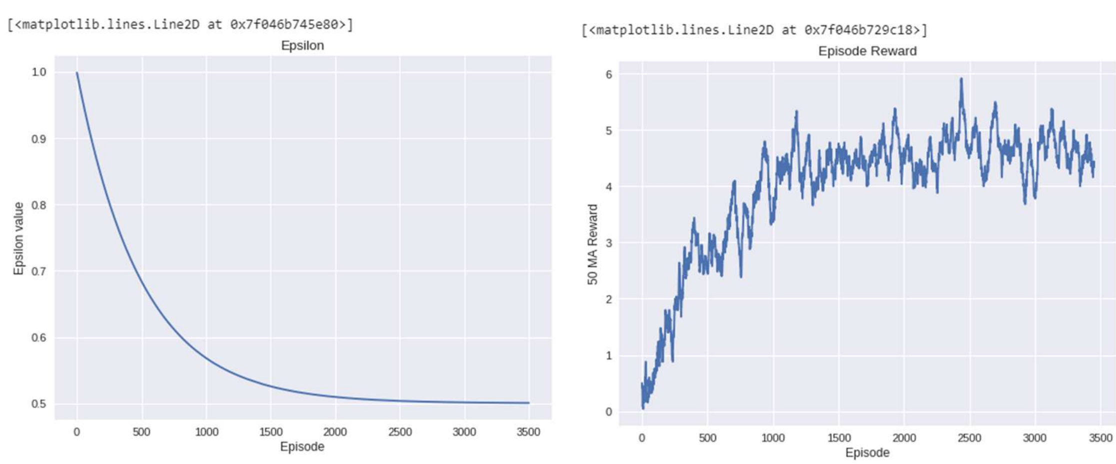

Figure 13: Curve of Test3

writing tasks

3.1 Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways to force the agent to explore.

Answer: If the agent always chooses the action that maximizes the Q-value, it will get some problem of training. When it gets its first Q-value, it will always follow the Q-value, and chooses the bigger one. Thus, it cannot try the other ways. But if we let it random chooses actions sometimes, it could try every actions and get the best Q-value. Two ways:

1) We can use “epsilon-greedy”. We can set an epsilon value which means when should the explorer choose random actions and when should the explorer choose actions depends on Q-value. In this project, we use a decay epsilon, which is 1.0 at beginning and decay to 0.01 during training.
2) Temporarily freeze the q_target parameter (cut off correlation). We can setup two neural networks, and they will have the same structure. Only difference is parameters: one of the networks has a value called target_net to predict q_target value and target_net will not be trained or update the parameter. The other network has a value called eval_net to predict q_eval, it will be trained and has the newest parameter of neural network.

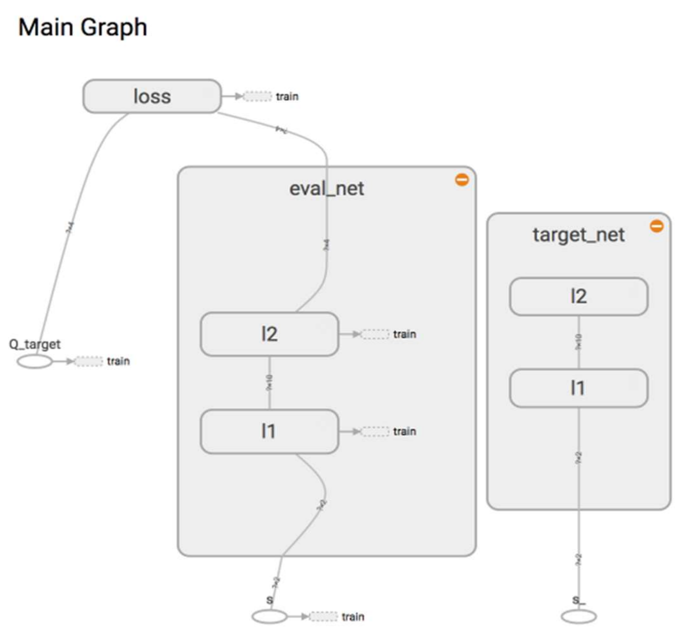

Figure 14: Two Neural Network to Freeze the q_target Parameter

3.2. Answer:

	ACTIONS			
STATE	UP	DOWN	LEFT	RIGHT
0	3.90099501	3.940399	3.90099501	3.940399
1	2.940399	2.9701	2.90099501	2.9701
2	1.940399	1.99	1.940399	1.99
3	0.9701	1	0.9701	0.99
4	0	0	0	0

Table 1: Q-table

STATE=4:
UP: Q = 0
DOWN: Q = 0
LEFT: Q = 0
RIGHT: Q = 0

STATE=3:
UP: Q = -1 + 0.99 * max(Q(3_up, action)) = -1 + 0.99 * (1 + 0.99 * 1) = 0.9701
DOWN: Q = 1 + 0.99 * max(Q(4, action)) = 1 + 0.99 * 0 = 1
LEFT: Q = -1 + 0.99 * max(Q(2, action)) = -1 + 0.99 * (1 + 0.99 * 1) = 0.9701
RIGHT: Q = 0 + 0.99 * max(Q(3, action)) = 0 + 0.99 * 1 = 0.99

STATE=2:
UP: Q = -1 + 0.99 * max(Q(1, action)) = -1 + 0.99 * (1 + 0.99 * (1 + 0.99 * 1)) = 1.940399
DOWN: Q = 1 + 0.99 * max(Q(2_down, action)) = 1 + 0.99 * 1 = 1.99
LEFT: Q = -1 + 0.99 * max(Q(2_left, action)) = -1 + 0.99 * (1 + 0.99 * (1 + 0.99 * 1)) = 1.940399
RIGHT: Q = 1 + 0.99 * max(Q(3, action)) = 1 + 0.99 * 1= 1.99

STATE=1:
UP: Q = 0 + 0.99 * max(Q(1, action)) = 0 + 0.99 * (1+0.99*(1+0.99*1)) = 2.940399
DOWN: Q = 1 + 0.99 * max(Q(2, action)) = 1 + 0.99*(1+0.99*1) = 2.9701
LEFT: Q = -1 + 0.99 * max(Q(0, action)) = -1 + 0.99*(1+0.99*(1+0.99*(1+0.99*1)))=2.90099501
RIGHT: Q = 1 + 0.99 * max(Q(1_right, action)) = 1 + 0.99*(1+0.99*1) = 2.9701

STATE=0:
UP: Q = 0 + 0.99 * max(Q(0, action)) = 0+0.99*(1+0.99*(1+0.99*(1+0.99*1))) = 3.90099501
DOWN: Q = 1 + 0.99 * max(Q(0_down, action)) = 1+0.99 * (1+0.99*(1+0.99*1)) = 3.940399
LEFT: Q = 0 + 0.99 * max(Q(0, action)) = 0+0.99*(1+0.99*(1+0.99*(1+0.99*1))) = 3.90099501
RIGHT: Q = 1 + 0.99 * max(Q(1, action)) = 1+0.99 * (1+0.99*(1+0.99*1)) = 3.940399