AI 3603 Artificial Intelligence: Principles and Techniques

By: Zhao Xiangyu: 518021910982 Zhu Fuze: 519030910066

HW#:2

October 26, 2021

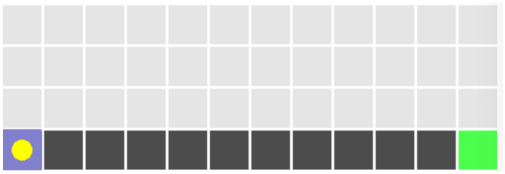
I. INTRODUCTION

In this assignment, we will implement Reinforcement Learning agents to find a safe path to the destination in a grid-shaped maze. The agent will learn by trail and error from interactions with the environment and finally acquire a policy to get as high as possible scores in the game. The assignment is divided into three parts: the first part is 50 pts, the second part is 50 pts and the bonus part is 5 pts.
About the installation:
Install anaconda
Create an environment for Gym.

II. FIRST PART: RL IN CLIFF-WALKING ENVIRONMENT

A. Introduction of the game rule

Suppose a 12*4 grid-shaped maze in the picture. We can see the purple rectangle represents the starting point and the green rectangle represents the exit. The yellow circle denotes the current position. We can move upward, downward, leftward and rightward in the game. Also, we will stay in place if we try to move outside the maze. As a result, we should reach the goal through the safe region(gray) and avoid falling into the cliff(black). Reaching the exit gives a reward +10 and falling into the cliff gives a reward -100, and both of the two cases terminate the current episode. Otherwise, a living cost (-1) is given.



(a) game1

The environment is implemented in gymgridworld.py and we won't modify it. The state space and action space are as follows:
(1)The state s_t is an integer, which represents the current coordinate (x,y) of the agent, i.e. s_t = x + 12*y.
(2)A discrete variable, where 0,1,2,3 represent move leftward, rightward, upward, downward respectively.

B. Game with Sarsa and Q-learning

We need to implement agents based on Sarsa and Q-learning algorithms, implement the agents in agent.py and complete the training process in cliffwalksarsa.py and cliffwalkqlearning.py respectively.

1. Cliffwalksarsa.py

In this section, we utilize some parameters to construct the agent, such as learning rate, reward decay γ, ϵ value, and ϵ-decay schema.

2. Agent.py

We need to implement ϵ-greedy with ϵ value decay in the choose_action function and functions given in the template need to be completed. We should keep a balance between exploration and exploitation by tuning ϵ value carefully.

C. Result Visualization and Analysis

We should visualize the process and the final result according to the following requirement:

- Plot the episode reward during the training process.
- Plot the ϵ value during the training process.
- Visualize the final paths found by the intelligent agents after training.

1. Code

Cliffwalkqlearning.py
In the cliffwalkqlearning, first, we add some instructions below:
1.numobservations = env.observationspace.n
numobservations obtain all the states from observationspace.
2.We initialize epsilon as 1 before for episode
3.We add a forloop about episode:

```
for episode in range(1000):
    episode_reward = 0;
    s = env.reset();
    env.render();
    episode_epsilons.append(epsilon)
    agent.set_epsilon(epsilon)
```

This part is added to renew epsilon after every forloop of iter.
4.We add an instruction: epsilon = epsilon*0.95;
5.We implement matplotlib.pyplot to draw the picture and the code below:

```
import matplotlib.pyplot as plt
plt.figure(1)
plt.plot(episode_rewards)
plt.xlabel('Episode')
plt.ylabel('Episode_Rewards')
plt.savefig('/Users/xiaojian_xiang/Projects/AI3606/HW2/QLearning/Cliff_reward_zero.png')
plt.show()

plt.figure(2)
plt.plot(episode_epsilons)
plt.xlabel('Episode')
plt.ylabel('Episode_Epsilons')
plt.savefig('/Users/xiaojian_xiang/Projects/AI3606/HW2/QLearning/Cliff_epsilon_zero.png')
plt.show()

print('\ntraining_over\n')

# close the render window after training.
env.close()

##### START CODING HERE #####
```

Agent.py
In this part, we added predict function, save function, restore function and setepsilon function in the class QlearningAgent
Chooseaction:

```
def choose_action(self, observation):
        """choose_action_with_epsilon-greedy_algorithm."""
        if np.random.rand() <= self.epsilon:
            # epsilon randomly choose action
            action = np.random.choice(self.all_actions)
        else:
            # greedily choose action
            action = self.predict(observation)
        return action
```

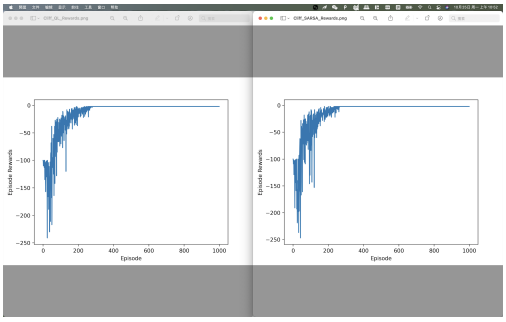
Cliffwalksarsa.py
In this part, in the forloop "for episode in range (1000)", we changed the code below:

```
for episode in range(1000):
    # record the reward in an episode
    episode_reward = 0
    # reset env
    s = env.reset()
    # render env. You can comment all render() to turn off the GUI to accelerate training.
```

```
    env.render()
    # agent interacts with the environment
    episode_epsilons.append(epsilon + 0.05)
    agent.set_epsilon(epsilon + 0.05)
    # episode_epsilons.append(epsilon)
    # agent.set_epsilon(epsilon)
    # Set epsilon = epsilon(k-1) * 0.95
    #agent.set_lr(learning_rate)
    a = agent.choose_action(s)
```

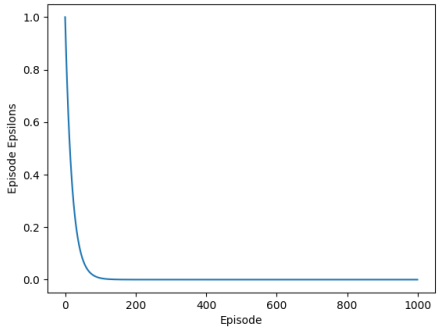
2. Result Visualization

Epsilon reward during the training process

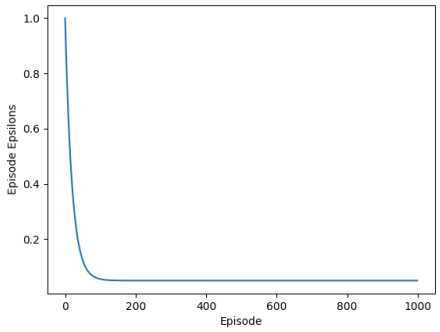


(b) Epsilonreward

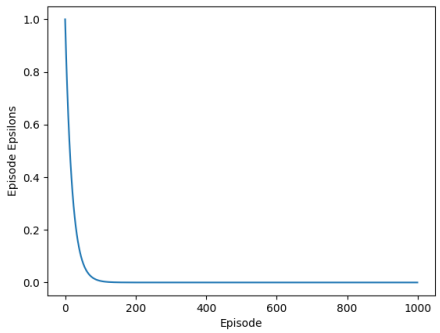
The left picture is qlearning and the right is sarsa. the *epsilon* value during the training process First, we try to make *epsilon* converge to 0 but we found that the paths we get through using qlearning and sarsa are the same. So we make *epsilon* converge to 0.1 then we found paths became different. Cause during the training process if we let epsilon converge to 0, then the path of sarsa algorithm will converge to the optimal path, the same as qlearning. Therefore, we obtain two different results and get four pictures.



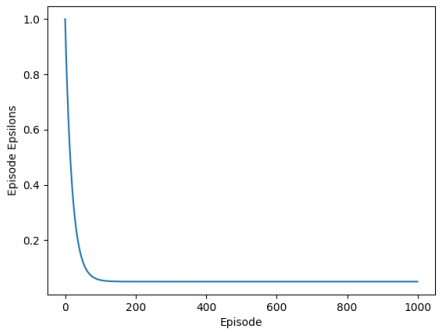
(c) Cliffepsilonzeroqlearning



(d) Cliffepsilonnozeroqlearning



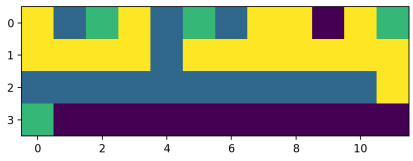
(e) Cliffepsilonzerosarsa



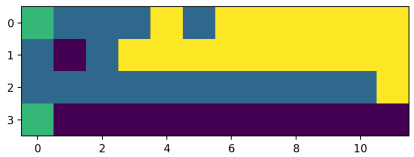
(f) Cliffepsilonnozerosarsa

Paths

First we set the rule that when we reach the purple floor then we turn left, reach the blue floor then we turn right, reach the green floor then we turn up and turn down when we reach the yellow floor.

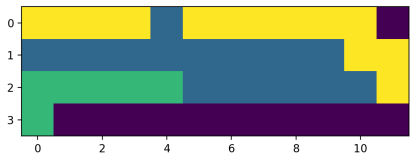


(g) qlearningway



(h) sarsazeroway

From the two pictures above, we can know that if we set the epsilon converge to 0, the paths we get are the same. However, if we set the epsilon converge to 0.1, then we get a different path below.



(i) sarsanozeroway

3. Result Analysis

Differences between Sarsa and Qlearning
The only difference between Sarsa and Qlearning is the way of updating the value of q.
For Q-learning, in the state s_t, according to the *epsilon*-greedy, perform the action A_t and reach the state S_{t+1}. And in the state S_{t+1}, choose the largest action Q(S_{t+1},a). In the state S_{t+1}, we choose the same

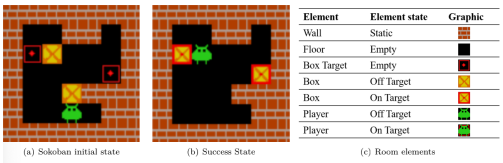
strategy to perform actions.
For Sarsa, in the state s_t, according the *epsilon*-greedy strategy, perform the action A_t and reach the state S_{t+1}, now the way of updating (S_t,a) is still the *epsilon*-greedy and really choose (S_{t+1},a).
For two algorithms, both are choosing *epsilon*-greedy strategy, the differences are the way of updating the value of Q in the Q-learning is greedy strategy-choose the largest Q(S_{t+1},a), but the way of updating the value of Q in the Sarsa is still the *epsilon*-greedy.

III. REINFORCEMENT LEARNING IN THE SOKOBAN GAME

In this part, we should implement intelligent agents to play the Sokoban game utilizing Sarsa, Q-learning and dyna-Q algorithms. Therefore, we will have a deeper understanding on the model-based RL algorithms and the explore-exploit dilemma.

A. Introduction of game rule

As shown in the picture, the game is a transportation puzzle, where the player has to push all boxes in the room on the storage targets. The possibility of making irreversible mistakes makes this game a bit challenging for RL agents.



(a) Sokoban initial state (b) Success State (c) Room elements

Element	Element state	Graphic
Wall	Static	
Floor	Empty	
Box Target	Empty	
Box	Off Target	
Box	On Target	
Player	Off Target	
Player	On Target	

(j) game2

In this task, a 7*7 room with two boxes is utilized, as illustrated in Fig.2(a). The environment utilized in this task is static, i.e. the room settings including box coordinates, target position, and agent positions are the same after each reset. In this game, we can move upward, downward, leftward and rightward. Pushing two boxes onto the targets gives a reward +10; Pushing one box on box on a target gives a reward +1, while pushing a box off the target gives -1. In addition, a reward of -0.1 is given for each step as living cost. the game terminates when two boxes are pushed onto the target, or 100 steps are exhausted before success.
State(Array): The state s_t is an array composed of 6 integers, where the first two integers represent current coordinate of the agent, and the last four integers represent current coordinates of two boxes in the room.
Action(Integer): A discrete variable and 0, 1, 2, 3 represent move upward, downward, leftward, rightward respectively. Game with RL algorithms We should solve this task utilizing Sarsa, Q-learing, and Dyna-Q algorithms repectively. We should implement the intelligent agents in agent.py and complete the training process in sokobansarsa.py, sokobanqlearning.py and sokobandynaq.py respectively. Actually, Sarsa and Q-learning agents implemented in Section First part can be utilized for this task.

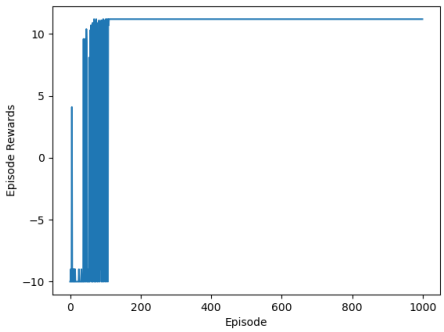
B. Result Visualization and Analysis

1. Result Visualization

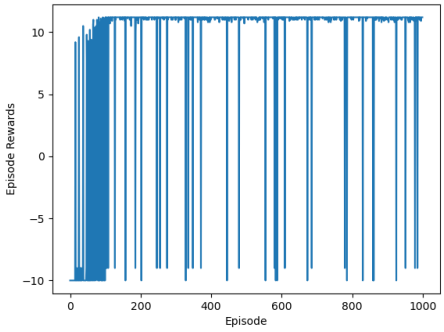
We should visualize the training process and the final result according to the following requirements:

- Plot the episode rewards during the training process.
- Plot the *epsilon* value during the training process.
- Visualize the final result of the three agents and we ought to record videos of the final result.

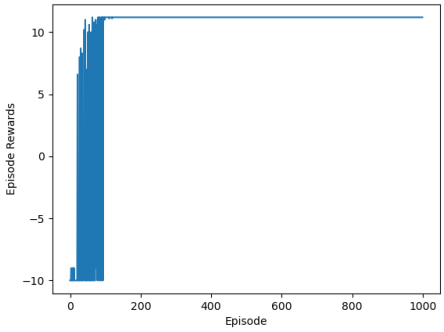
The episode reward during the training process



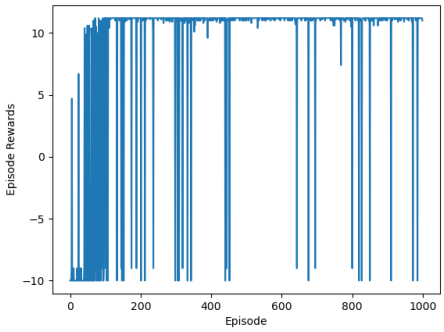
(k) SOqlrewardzero



(l) SOqlrewardnozero

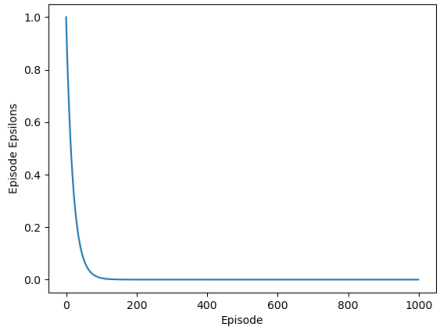


(m) SOsarewardzero

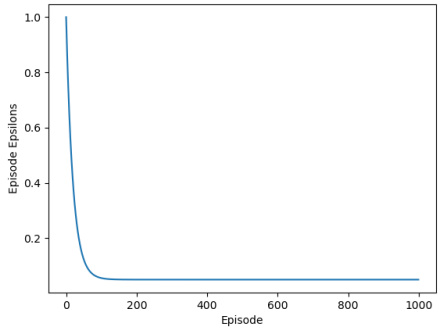


(n) SOsarewardnozero

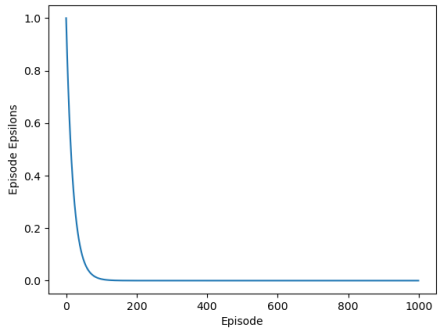
The *epsilon* value during the training process



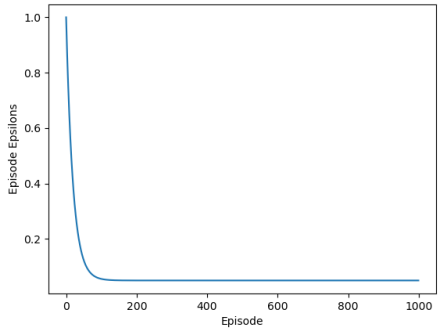
(o) SOqlepsilonzero



(p) SOqlepsilonnozero

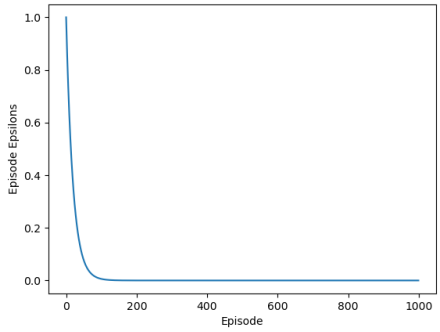


(q) SOsaepsilonzero

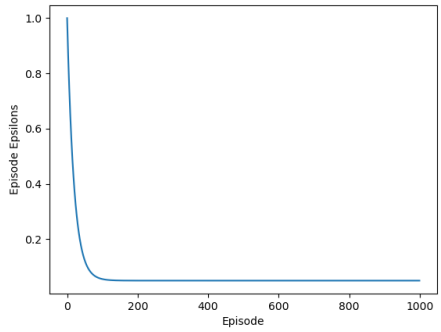


(r) SOsaepsilonnozero

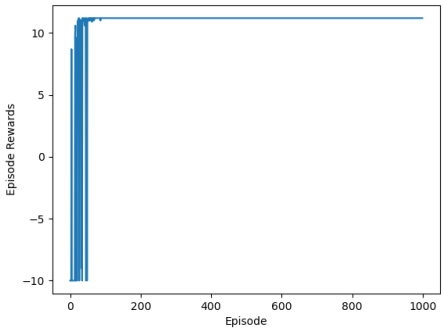
From the four pictures above, we can see that the epsilon converge to 0 and 0.1. Then we applied dyna-q algorithm and get the results below:



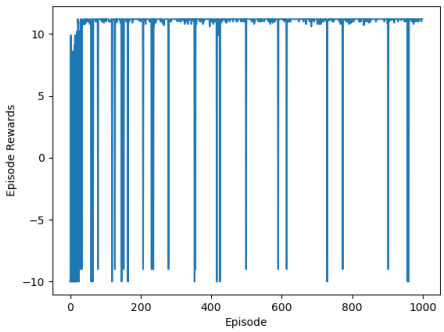
(s) SOdynaqepsilonzero



(t) SOdynaqepsilonnozero



(u) SOdynaqrewardzero



(v) SOdynaqrewardnozero

The final results of three agents The results are recorded videos in the file list. And from the videos, we can know that the paths are different through different algorithms.

IV. EXPLORE-EXPLOIT DILEMMA

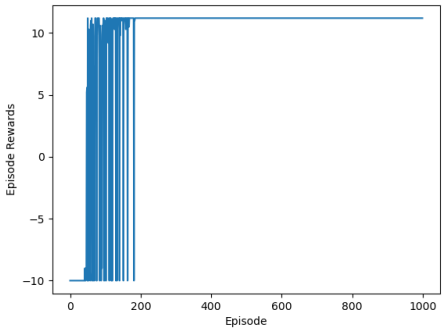
We have found the importance of exploration-exploitation dilemma in the Reinforcement Learning through previous experiments. So in this section, we should analyze the influence of different exploration schemes on the learning speed and result.

- Conduct experiments in the Sokoban environment utilizing any one RL algorithm with different *epsilon* values and *epsilon*-decay schemes. Try to summarize the relationship between different exploration schemes and learning speeds/results.
- Actually, there exists lots of other exploration strategies except *epsilon*-greedy. We can find and learn one new exploration method, such as Upper Confidence Bound(UCB)
- (Bonus)Implement the exploration strategy we have found in the Sokoban environment. We can add new agent class in the agent.py and the corresponding training process should be implemented in sokobannewexploration.py.

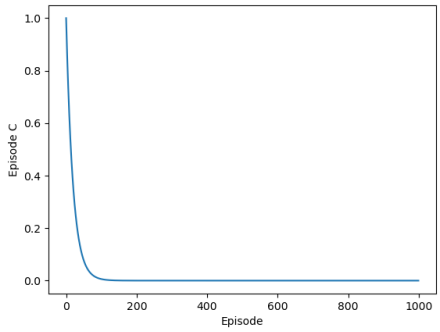
A. Result Visualization and Analysis

About this algorithm, we spent a lot of time hoping to finish it.

1. Result Visualization



(w) SOrewardzero



(x) SOepsilonzero

2. Result Analysis

There are some advantages of Upper confidence Bound over other algorithms. First, it makes a good banlance between exploitation and exploration.Second, this exploration is an optimization exploration and

the action of each exploration is certain.Third, it is proved that it can converge to the optimal strategy in finite time. In addition, it performs better than Q-learning algorithm, Sarsa algorithm and Dynaq algorithm.What's more, from the pictures above, we can see that it has a faster convergence and we have provided comparison diagram in the files.

V. DISCUSSION & CONCLUSION

A. Submission File List

- agent.py:Implementation of RL agents;
- cliffwalksarsa.py:train Sarsa in Cliffwalking environment.
- cliffwalkqlearning.py:train Q-learning in Cliff-walking environment.
- sokobanqlearning.py:train Q-Learing in Sokoban environment.
- sokobansarsa.py:train Sarsa in Sokoban environment.
- sokobandynanq.py: train Dyna-Q in Sokoban environment.
- sokobannewexploration.py:(Optional, Bonus)train RL agent with new exploration method in Sokoban environment.
- videos(floder): Demo videos recorded.
- gymsokoban(folder):Sokoban game environment.
- gymgridworld.py: Cliff-Walking game environment.
- All the pictures and videos during the experiment are listed in the files.

B. Conclusion

In this homework, we applied qlearning and sarsa algorithm to run two games: Cliffwalking and Sokoban and we succeeded to perform the algorithms. During the process, we observed the change of episode reward and *epsilon* value. At last, we applied Upper Confidence Bound(UCB) to optimize the results we got before. In conclusion, we obtain a lot of knowledge about reinforcement learning and had a keen interest in this and we hope we can learn more about reinforcement learning in the following learning process.