[OT5406] Deep Learning for Robotics
Department of Mechanical Engineering
National University of Singapore

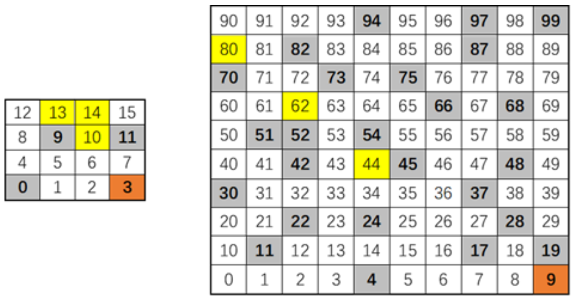
Project 1: The Froze Lake Problem and Variations

Gao Xinwei
A0243230X
e0816524@u.nus.edu

1 Performance and Result

1.1 Environment Building and Data Sampling

According to the problem statement, consider the robot-hole-frisbee as a mathematical problem in the table. Then, mode the environment like the table below. As the figure below, the gray grids represent the hole, the orange ones represent the frisbee, and all of the numbers are used to represent the $Q(s, a)$ of the algorithm.



12	13	14	15
8	**9**	10	**11**
4	5	6	7
0	1	2	**3**

90	91	92	93	**94**	95	96	**97**	98	**99**
80	81	**82**	83	84	85	86	**87**	88	89
70	71	72	**73**	74	**75**	76	77	78	79
60	61	62	63	64	65	**66**	67	**68**	69
50	**51**	**52**	53	**54**	55	56	57	58	59
40	41	**42**	43	44	**45**	46	47	**48**	49
30	31	32	33	34	35	36	**37**	38	39
20	21	**22**	23	**24**	25	26	27	**28**	29
10	**11**	12	13	14	15	16	**17**	18	**19**
0	1	2	3	**4**	5	6	7	8	**9**

Figure 1: The holes and recording states: grey grid means holes, while the yellow ones mean recording states.

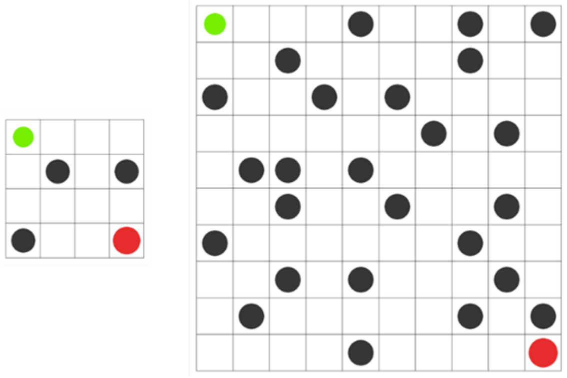


Figure 2: Build the environment in the gym.

Build the environment at the gym without reference. The state-action transition function will be regarded as $S_{t+1} = S_t + actionvalue$. The action value for up, right, down, left is equal to $+gridwidth$, +1, $-gridwidth$, -1, where the $gridwidth$ is represented 4 or 10 for task 1 and task 2 respectively.

For observing the training process and making results analyzing visual, the yellow grids are remarked as recording grid, which used to log and reveal the training process. For example, the optimal action choice for state 13 in task 1 is right, while the action choice of the left still can be a sub-optimal choice for the robot achieving the frisbee. The author believes that the Q value changing can help us understand three compared methods' policy updating process with the reward curve.

The author trained three tasks with each method in three different e-greedy values (0.05, 0.10, and 0.20) and listed the training record as Figure 3. Notice that here is the random initial beginning result. The author will explain the reason in Section 3.

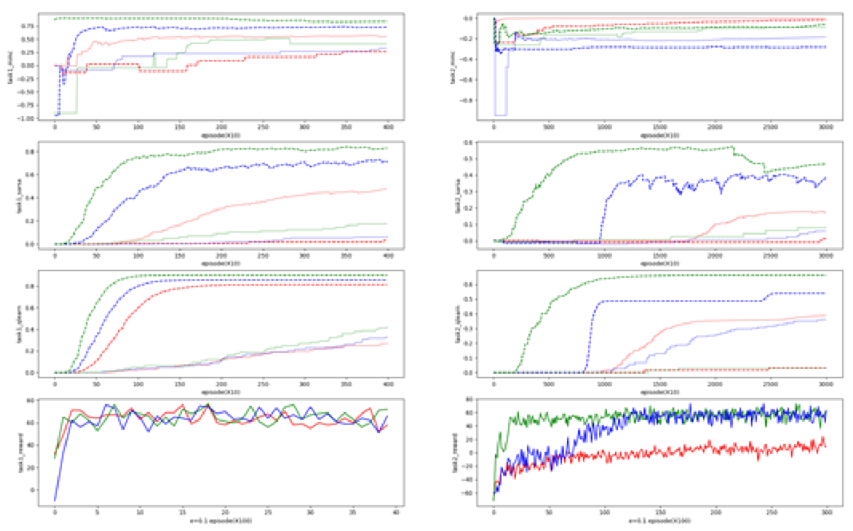


Figure 3: e-greedy = 0.05. The best result used to analysis.

In each image, the left side figures show how three different methods perform in Task 1; simultaneously, the right side figures reflect how these methods are updating policy in Task 2.

The red, blue, green lines represent Q-value at states 10, 14, 13 in task 1, and state 44, 62, 80 in task 2 on the top three lines. Moreover, these lines represent the reward of Monte Carlo Control, Q-Learning, and SARSA in the bottom reward figure.

The connecting lines and dotted lines represent the action's optimal choice and suboptimal choice at a specific state. Take the e-greedy=0.05 Task 1 Q-Learning, for example, the blue connected line shows the change of $Q(14, down)$, and the blue dotted line represents the updating of $Q(14, left)$. Once $Q(14, down) > Q(14, left)$, we can consider the algorithm out of the sub-optimal choice and begin to choose the better one. At the same time, we can observe the inner part of the algorithm and know how the decision change.

The author calculates and record the time for the three algorithms to train a specified number of sessions in two environments in the Table 1.

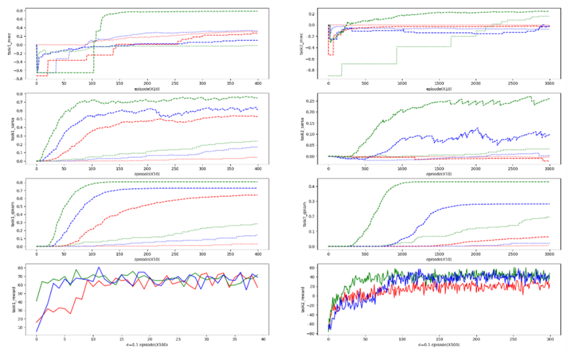


Figure 4: e-greedy = 0.10. The result used to compare.

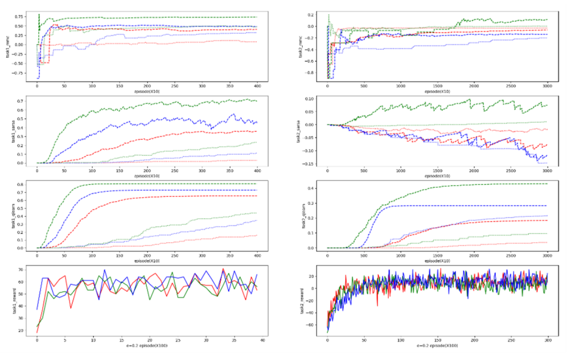


Figure 5: e-greedy = 0.20. The result used to compare.

Table 1: Calculation time of three algorithms in different tasks.

	Monte Carlo Control	SARSA	Q-Learning
Task1 (4000 episodes)	33 s	35 s	33 s
Task2 (30000 episodes)	439 s	260 s	250 s

The author expands the $Q(s, a)$ table and takes its maximum value as the value of the highest action in the current cell as the value of this state, since in SARSA the updating formula is Equation 1:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{1}$$

and the updating formula is Equation 2 in Q-Learning:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)] \tag{2}$$

By displaying the grid world on the canvas, the following patterns can be obtained for easy observation and analysis to concisely understand the action strategy of the robot under the current strategy situation.

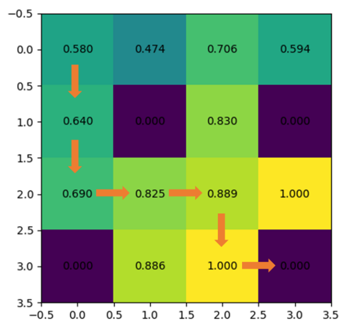


Figure 6: Task 1: State value of SARSA.

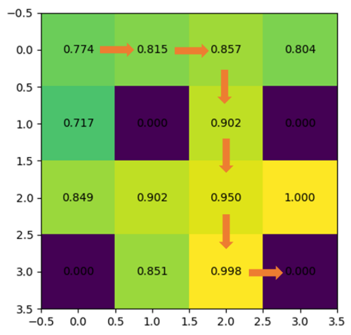


Figure 7: Task 1: State value of Q-Learning.

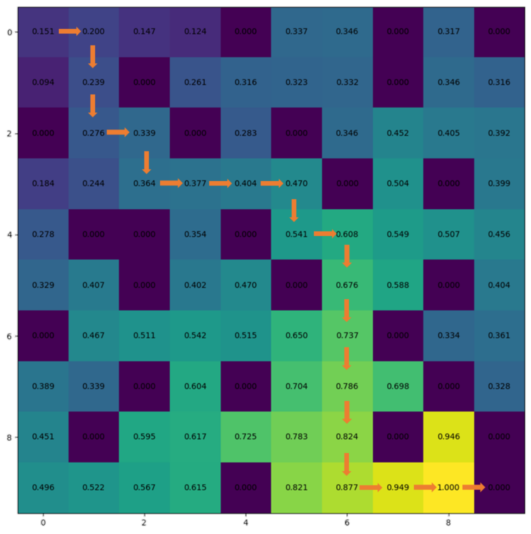


Figure 8: Task 2: State value of SARSA.

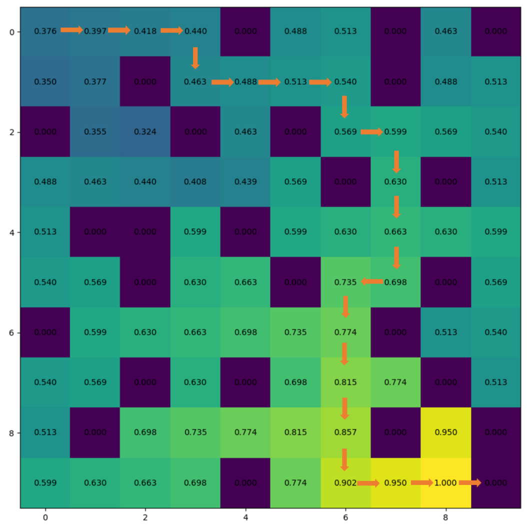


Figure 9: Task 2: State value of Q-Learning.

1.2 Result Analysis

The data shows that in task 1, three methods have almost the same performance:55, 60, 70 in 100 episodes when e-greedy is equal to 0.2, 0.1, 0.05, respectively. However, it is noticed that except the Q-Learning is caught in sub-optimal choice when e-greedy = 0.05 as the dotted lines lay above the connected ones. Moreover, the Monte Carlo Control needs around 100 episodes to optimize its policy when e-greedy is in the lower range(0.1 group and 0.05 group), slower than other methods.

In task 2, the reward curve illustrates that Q-Learning is the best method for the task. Since higher e-greedy may lead to the mistake of falling into holes during the task, walking to the frisbee is not a good choice for the robots so far away from, compared with circling. So the lower e-greedy experiments are more persuasive. The Monte Carlo Control and SARSA reward are lower than 50 in 100 rounds in e-greedy = 0.05, while that number of Q-Learning is higher than that.

2 Result Analysis

2.1 Value Update Curve - Different

The Monte Carlo Control's value updating curve always shows convulsions at the beginning of training, which also needs more time to converge its policy(connected lines and dotted lines never intersect since then).

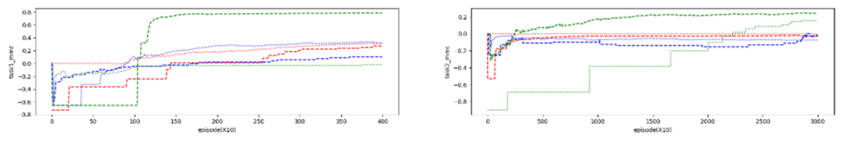


Figure 10: Monte Carlo Control's estimated value shows fluctuation at the beginning.

The change of SARSA's and the Q-Learning's value updating curve is smoother and steady at the beginning and all from the same value. Nevertheless, the value updating of SARSA shows the fluctuation of value even in the period that policy has shown converge. Moreover, this fluctuation is pronounced when we put the Q-Learning to compare with.

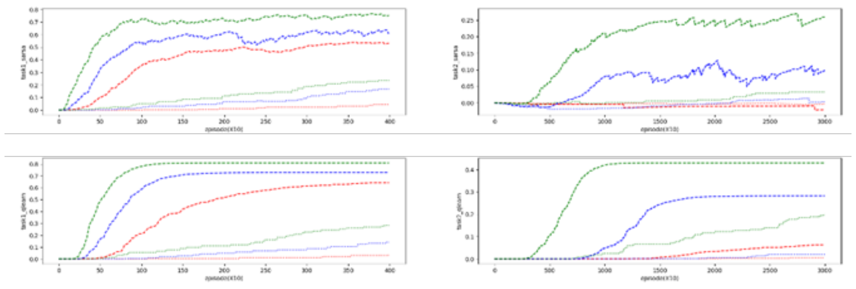


Figure 11: The value curve of SARSA and Q-Learning is more stable. The SARSA shows fluctuation after convergence.

2.2 Effect of e-greedy - Same

The three methods' final reward almost lies on the same level. The main factor that affects the reward level mostly is e-greedy in a different group of experiments. This trend is mainly because, with the increasing e-greedy, the agent has more probability of dropping into the holes. For example, the decreasing reward in task 1 and task 2 for e-greedy = 0.05, 0.1, 0.2 respectively show below.

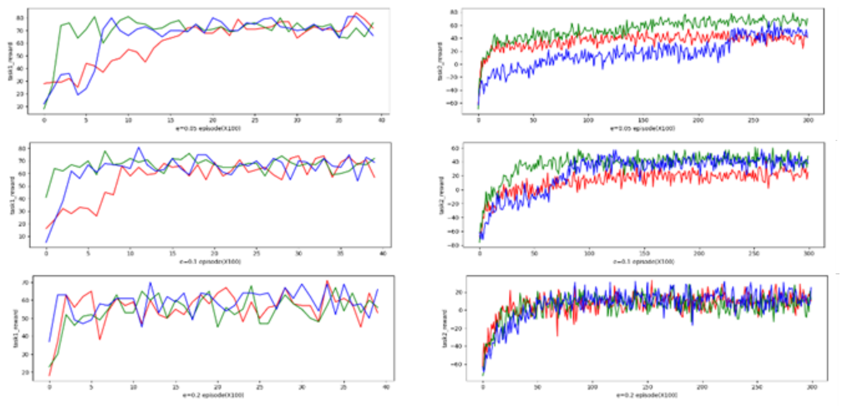


Figure 12: The reward of the three methods remains on the same level. And higher e-greedy, lower performance on tasks.

2.3 Order to explore - Same

The lines raise in GBR order for both tasks, except sometimes will show this order in inverse sequence when facing task1. This pattern reflects that the $Q(s,a)$ table is updated from the state closed to the frisbee and gradually updated the accurate estimated value to the states far away from the frisbee. Showing RBG in task 1 sometimes is caused by sub-optimal choice, since the state 10, 14 and 13 is nearby each other, causing the state transfer order like 13 to 14 to 10, which will never happen in task 2 leading by the distance between the state 44, 62 and 80.

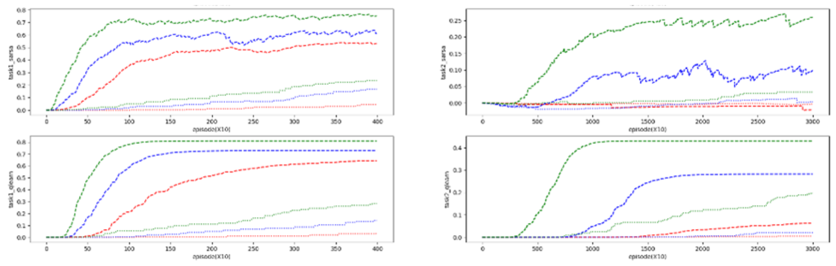


Figure 13: The lines raised in GBR order or RBG order.

3 Unexpected Result

3.1 Choice of gama

During the pre-training stage using the Monte Carlo Control method, the author found the appointed recording $Q(s,a)$ always becomes very similar, which leads hard to observe the inner change of the algorithm. Take the figure below, for example, the green connected line and dotted one which represents $Q(44,down)$ and $Q(44,left)$ respectively, intervene together and effect the research observation.

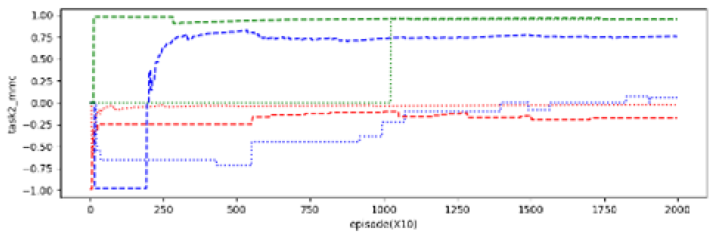


Figure 14: The connected line and the dotted one intervene together.

This Q value similarity is probably because the high gama(=0.99) leads to the slight difference between choosing down and choosing right at state 44. Specifically, only three steps can go back to state 34 after choosing the left, which is the same as choosing down. So the similar rate between the Q(44,down) and Q(44,left) is equal to $gama^3$, which is equal to 0.97 when gama=0.99.

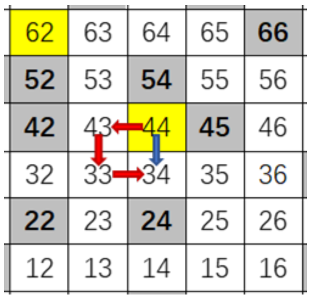


Figure 15: Sight difference between down and left action, which leads to a similar Q-value.

To prevent the difficulty of observation, the author set this gama parameter as 0.9 in later training, and believe the lower similar rate $gama^3$ (=0.729) can lead better graph for observation.

4 Difficulties and Overcome

4.1 Stuck at the Initial State

At the beginning of this program, the author found that the agent has a very high possibility of sticking around the initial point, especially in the 10*10 grid with the Monte Carlo Control algorithm, as the situation shown in Figure 16. That is probably because the period experience tells the agent that leaving the initial point will likely fall into the holes. Furthermore, the negative experience of falling into the holes is high enough, so the agent prefers to step back and not further explore.

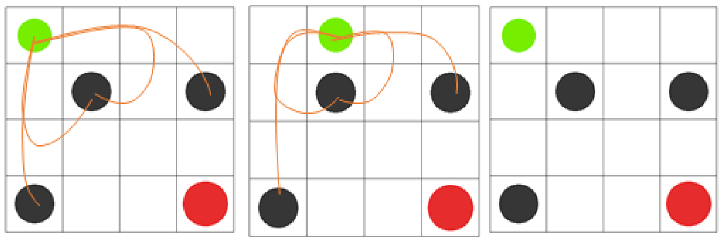


Figure 16: The robot is stuck because of the negative experience.

To conquer this not exploration problem and let the robot has a full understanding of the grid world, the author added a random function in the environment, allowing the agent to start the exploration of the world at any state and forcing the agent to update the Q table that may never update if start at the fixed point. This randomized starting method helped the robot get out from the sub-optimal choice and find the right path to the frisbee.

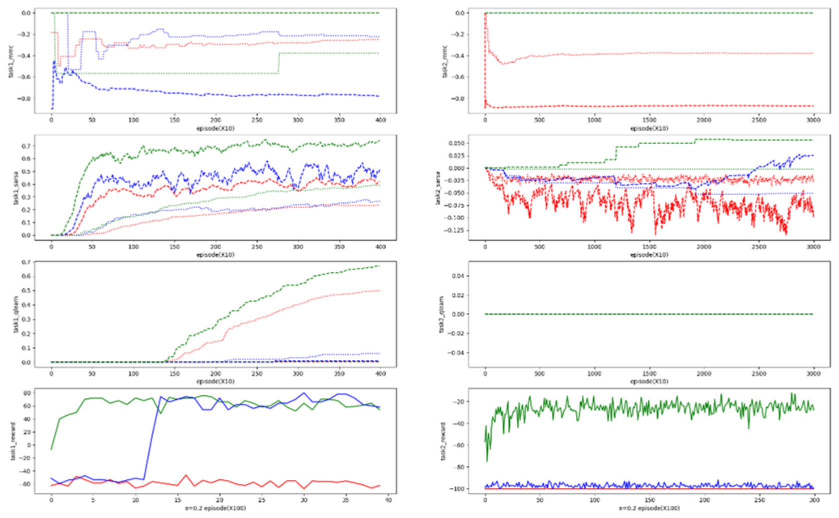


Figure 17: The training result without randomized initial starting state.

4.2 Up Action Bug in Program

However, after randomly initializing the robot's starting points, the training process shows a very weird pattern – at the beginning of the training process, the robot always chooses to walk up and fall into a locally optimal point, even if the frisbee is closer than others, as the Figure shown in 18.

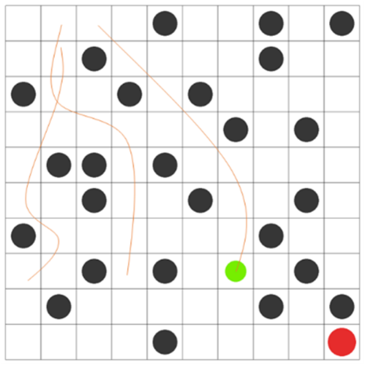
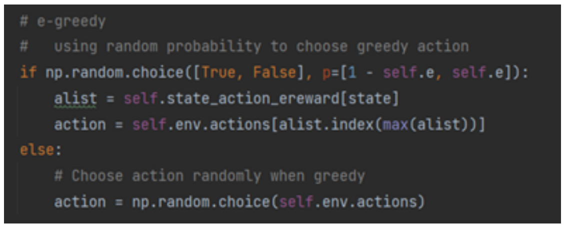


Figure 18: The agent always tends to go up at the beginning.

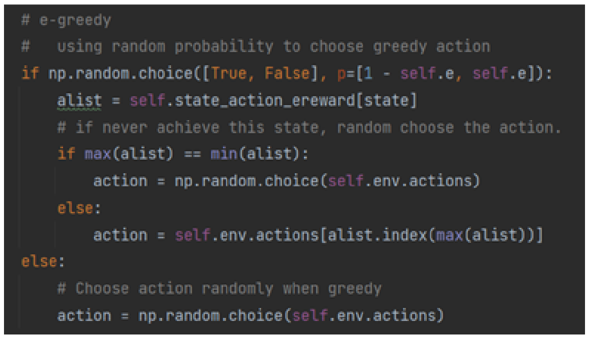
Checking back to the choosing code of action part, the author found a flaw of the code 'action = self.env.actions[alist.index(max(alist))]'. After the max() function, take out the max value in the action list that represents the $Q(s, a|s = s_t)$, the alist.index() will return the index of this maximin value. But there is a particular case that is common at the training beginning, and there will be many same maximin value in $Q(s, a|s = s_t)$. It is possible to appear that $Q(s_t, up) = Q(s_t, down) = Q(s_t, left) = Q(s_t, right)$. In that case, the index() can only return the initial maximin value's index. While the headmost corresponds to the up action, the robot will explore the upper area preferentially.



```
# e-greedy
#   using random probability to choose greedy action
if np.random.choice([True, False], p=[1 - self.e, self.e]):
    alist = self.state_action_ereward[state]
    action = self.env.actions[alist.index(max(alist))]
else:
    # Choose action randomly when greedy
    action = np.random.choice(self.env.actions)
```

Figure 19: The part of code that leads up action.

One method to fix this priority is to judge whether the maximin value and the minimum value are the same. If that is true, which means the robot has never achieved this state, choosing one randomly will avoid the up action priority, shown as Figure 20 .



```
# e-greedy
#   using random probability to choose greedy action
if np.random.choice([True, False], p=[1 - self.e, self.e]):
    alist = self.state_action_ereward[state]
    # if never achieve this state, random choose the action.
    if max(alist) == min(alist):
        action = np.random.choice(self.env.actions)
    else:
        action = self.env.actions[alist.index(max(alist))]
else:
    # Choose action randomly when greedy
    action = np.random.choice(self.env.actions)
```

Figure 20: The part of code that leads up action.

5 Initiatives to Improve

5.1 Prior Knowledge and Guidance

For now, these three algorithms are based on the policy that updates from the trial and error experience. The bad thing is that the robot needs plentiful time to get the optimal policy from scratch, learning to avoid action after dropping into the holes millions of times. The author thinks that we can add some prior knowledge to the robot, just like the human baby born with the instinct leading him to feed. During the growth of human beings, we humans also gather knowledge from books, teachers, mediums, and other ways. The robot control algorithm should also have this ability and opportunity to learn the skill and knowledge that way, not just learning from scratch.

We should add some human beings or experts action as the prior experience into the experience buffer or do some demonstration so that the agent does not learn the policy from scratch and gets out from the locally optimal choice alone. This method most likely helps the agent get out from the initial state without randomized initial state and speed up convergence speed.

5.2 More Effective Update

The trend of convergence is pronounced in task 2 with the Q-Learning method, after which the algorithm still updates its estimated value of the state-action table step-by-step, which seems not intelligent enough. However, we can add some factors to the updating process so that the algorithm has the confidence to update the policy bravely.

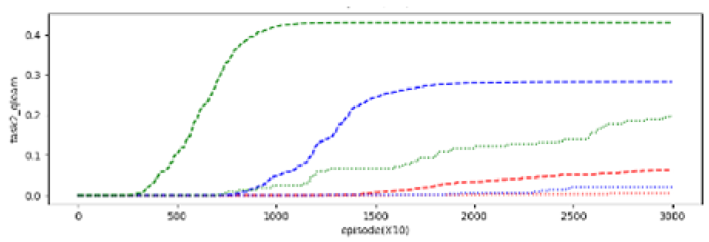


Figure 21: Steps wasted on value convergence.

Another method of accelerating the updating process is that use the experience data wisely. For the algorithm SARSA and Q-Learning, one trajectory is only used by algorithms once; then, the data will drop, which is unadvisable in the big data age. The agent should have some ability to review the experience learned and use this memory to update the policy.

5.3 More Stable Update

Due to the primary purpose of this project is to compare the difference and find the similarity between Monte Carlo Control, SARSA, and Q-Learning, the author did not use the dropping of the e-greedy and the learning rate. Unfortunately, this leads to the policy of high e-greedy(=0.2) performance terrible at task 2, and updating shows dramatic waving in the SARSA convergence part. However, the author thinks adding some technology detail like dropping e-greedy and learning rate will help learning process becomes more stable.

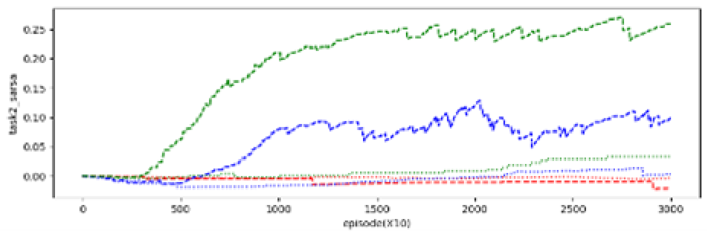


Figure 22: Dramatical waving in SARSA convergence part.

6 Present Issues

6.1 The difficulty of Application in Reality

Reinforcement learning is based on the principle of trial and error, but this mechanism may not be realistic in a real-world application. Take this robotic task as an example; we can not allow the robot to work on the frozen lake and have the chance of dropping into the holes, which will cost too much and pollute the environment. Moreover, the agents can not start the tasks randomly and are also limited by space.

6.2 Rigidly Learning Process

The mechanism of the grid world in 4*4 size and 10*10 size is exceptionally the same. The robot starts somewhere and runs to the right-bottom corner, avoiding dropping into the holes. The only difference is the size of the world. However, for these three algorithms, the agent needs to learn from the beginning in different world sizes even if it has been through the training of smaller or larger size world, which is not practical and general enough.

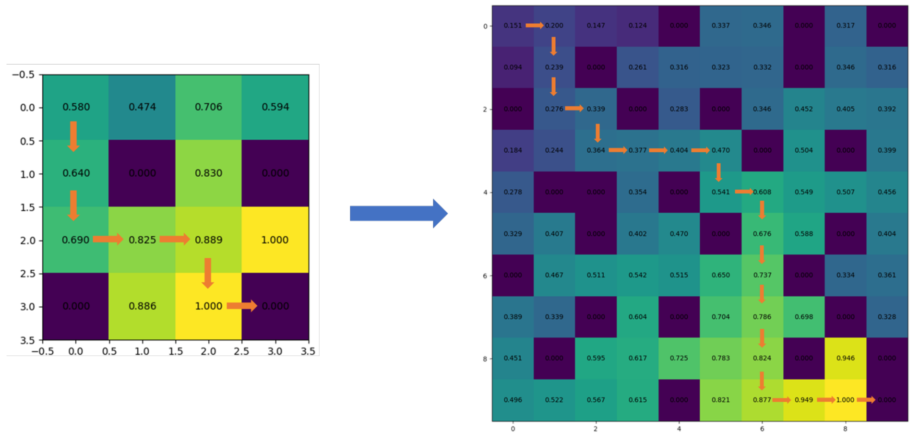


Figure 23: The policy cannot transfer to a 100 size grid world after training in 16 grid world, although the mechanism of environment is exactly the same.

The same rigid process does not just happen when the parameter of the world has changed. It also appears in the inner part of the grid world. So, for example, it is almost the same strategy for human beings running from the top-right corner or the left-bottom corner to the frisbee. However, the algorithm still needs the same and extra episodes to update its policy of both sides, which is also unadvisable.