ME5406
Deep Learning for Robotics

Project for Part I: The Froze Lake Problem and Variations

Student name: Wang Yutong

Student ID: A0225480J

Student email address:e0576114@u.nus.edu

1. Introduction

The main purpose of this report is to explain the implementation of the three model free reinforcement learning algorithms (First-visit Monte Carlo control without exploring starts，SARSA with an ϵ-greedy behavior policy，Q-learning with an ϵ-greedy behavior policy）in the frozen lake environment, compare the experimental results, and further discuss the causes of the differences and techniques to improve performance.

First, run three different algorithms in the original 4*4 and 10*10 environment respectively, and then keep the hyperparameters unchanged. Run the algorithm again in the reshaping reward environment and the valid action environment, and then get the performance of three algorithms in the three environments. Finally, the experimental results are analyzed from the path and the number of steps required to obtain the correct policies, etc. The results show that Q-learning learns a optimal policy π^*, While SARSA learns a near-optimal policy and the performance of SARSA and Q-learning is better than that of Monte Carlo, and the performance and efficiency of the algorithm can be improved by reshaping reward and taking valid action.

2. Environment Setting

I built my own environment using tkinter. It can output a gif, which is easier to understand than the original gym environment. as is illustrated in Figure 1 and Figure 2. The red circle is a robot, the green circle is a frisbee, the black grids are holes, and the other blank grids are ordinary ice surfaces. Each grid represents a different state.

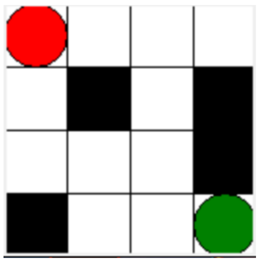


Figure 1. 4*4 environment

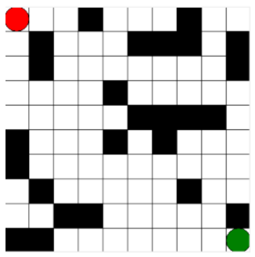


Figure 2. 10*10 environment

2.1 Original Environment

A frozen lake with holes (25% of state) covered by patches of very thin ice. A robot is to glide on the frozen surface from the top left corner to bottom right corner to pick up a frisbee.

The action space of the environment is robot can move in one of four directions, left, right, up, and down. If the robot's action moves beyond the boundary, the action will be not performed and the robot will remain in place. The observation space of the environment is the state of the robot, which is the index of the grid in which the robot is now located. The robot receives a reward of (i) +1 if it reaches the frisbee, (ii) −1 if it falls into a hole, and (iii) 0 for all other cases. An episode ends when the robot reaches the frisbee or falls into a hole.

2.2 Reshape Reward Environment

The environment is similar to the original environment, except that the reward settings are different. The robot receives a reward of (i) +10 if it reaches the frisbee, (ii) −1 if it falls into a hole, and (iii) -0.01 for all other cases.

2.3 Valid Action Environment

The environment is similar to the original environment, except that added a new function' valid action' to the environment. This new function will return the actions that can be performed in the existing state (the robot will not exceed the boundary after performing these actions). In this environment, whether greedy or not, the robot will only perform valid actions.

3.Algorithm

3.1 First-visit Monte Carlo Control Without Exploring Starts

```
Initialize:  ∘ $\pi(s) \in \mathcal{A}(s)$, arbitrarily, for all $s \in \mathcal{S}$
             ∘ $Q(s,a) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
             ∘ Returns$(s,a)$ as an empty list, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

             Loop forever (for each episode):
                 Select $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ randomly so that every $(s,a)$ pair
                     has non-zero probability of being selected
                 Generate an episode with $T$ steps starting from $(S_0, A_0)$ following $\pi$:
                     $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
                 $G \leftarrow 0$
                 Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
                     $G \leftarrow \gamma G + R_{t+1}$
                     Unless pair $(S_t, A_t)$ appears in $S_0, A_0, S_1, A_1, \ldots, S_{T-1}, A_{T-1}$:
                         Append $G$ to Return$(S_t, A_t)$
                         $Q(S_t, A_t) \leftarrow$ average(Return$(S_t, At)$)
                         $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, A_t)$
```

Figure 3. First-visit Monte Carlo control without exploring Starts

Monte Carlo is a model free reinforcement learning algorithm. It does not need complete knowledge of the environment. It requires only experience—sample sequences of states, actions, and rewards from interaction with an environment, then use the mean return calculated from experience to estimate the state or action values, from which an optimal policy can be obtained .

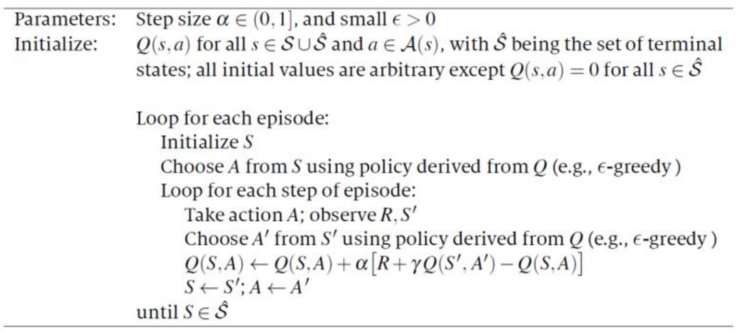
Monte Carlo methods only can be use in episodic tasks. Only on the completion of an episode are value estimates and policies changed.

The first-visit Monte Carlo method means that estimates V_π(s) or Q_π(s,a), as the average of the returns following first visits to s or s,a in an episode.

In order to better explore states without exploring starts, use ϵ-greedy policy to ensure that all actions are selected infinitely often. ϵ-greedy policy means that most of the time policy choose an action that has maximal estimated action value, but with probability ϵ they instead select an action at random, defined as

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & if\ a = A^* \\ \frac{\epsilon}{|A(s)|} & if\ a \neq A^* \end{cases} \quad (1)$$

3.2 SARSA With An ϵ-greedy Behavior Policy



```
Parameters: Step size $\alpha \in (0,1]$, and small $\epsilon > 0$
Initialize: $Q(s,a)$ for all $s \in \mathcal{S} \cup \hat{\mathcal{S}}$ and $a \in \mathcal{A}(s)$, with $\hat{\mathcal{S}}$ being the set of terminal
            states; all initial values are arbitrary except $Q(s,a) = 0$ for all $s \in \hat{\mathcal{S}}$

            Loop for each episode:
                Initialize $S$
                Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy )
                Loop for each step of episode:
                    Take action $A$; observe $R, S'$
                    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy )
                    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
                    $S \leftarrow S'; A \leftarrow A'$
            until $S \in \hat{\mathcal{S}}$
```

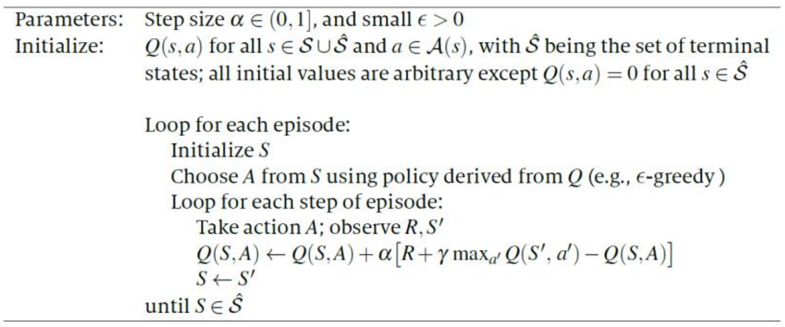
Figure 4. SARSA with an ϵ-greedy behavior policy

SARSA is a on-policy temporal-difference method. Unlike Monte Carlo methods, temporal difference methods update $V(s)$ or $Q(s,a)$ after one or a few time steps within an episode. The update rule of SARSA is

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (2)$$

To apply this rule, the action values $Q(s,a)$ are initialized to some arbitrary values for non-terminal states and to zero otherwise. Starting at a non-terminal state S_t , the agent takes an action A_t according to an ϵ-greedy policy π that enables the agent to visit every state-action pairs with a non-zero probability. After taking the action A_t the agent finds itself in state S_{t+1} and receives a reward R_{t+1}.The value of $Q(S,A_t)$is then updated according to Equation (2). Repeat the above process, until a terminal state is reached.

3.3 Q-learning With An ϵ-greedy Behavior Policy



```
Parameters:  Step size $\alpha \in (0,1]$, and small $\epsilon > 0$
Initialize:  $Q(s,a)$ for all $s \in \mathcal{S} \cup \hat{\mathcal{S}}$ and $a \in \mathcal{A}(s)$, with $\hat{\mathcal{S}}$ being the set of terminal
             states; all initial values are arbitrary except $Q(s,a) = 0$ for all $s \in \hat{\mathcal{S}}$

             Loop for each episode:
                 Initialize $S$
                 Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy )
                 Loop for each step of episode:
                     Take action $A$; observe $R, S'$
                     $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_{a'} Q(S', a') - Q(S,A)]$
                     $S \leftarrow S'$
             until $S \in \hat{\mathcal{S}}$
```

Figure 5. Q-learning with an ϵ-greedy behavior policy

Q-learning is a off-policy temporal-difference method. The update rule of Q-learning is

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)] \quad (3)$$

This form looks similar to that in SARSA; the only difference is that the term $Q(S_{t+1}, A_{t+1})$ in SARSA is replaced by the term $max_{a'} Q(S_{t+1}, a')$ in Q-learning. In this case, the learned action-value function Q, directly approximates q^* the optimal action-value function, independent of the policy being followed.

4. Experiment Results and Analysis

4.1 Path Analysis

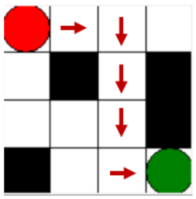


Figure 6 Monte Carlo 4*4path

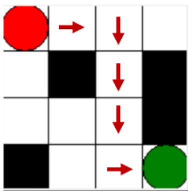


Figure 7.SARSA 4*4path

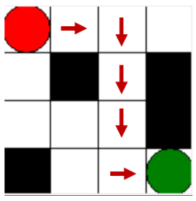
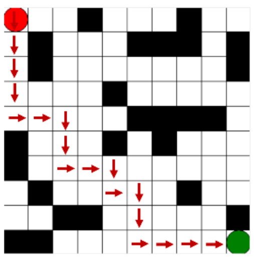


Figure 8. Q-learning 4*4path



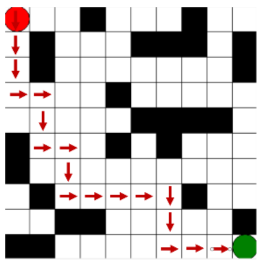


Figure 9. SARSA 10*10path Figure Figure 10. Q-learning 10*10path

The figures above are the path figure of the trained policy. The red arrow indicates the path chosen by the trained policy. As can be seen from the figure above, in the 4*4 environment, all three algorithms chose the same path, but in the 10*10 environment Q-learning learns a optimal policy π^*, which is the robot moves directly to the goal. While SARSA learns a near-optimal policy which is the robot first move away from holes, then move to goal. The cause of this difference is that the target policy of Q-learning is the deterministic greedy policy with respect to the optimal action values q^*(s,a),but the target policy of SARSA is the ϵ-greedy policy. For Q-learning, since its target policy is the greedy policy, convergence to the optimal action values of the target policy means convergence to q^*(s,a),but SARSA will converge to the optimal action values for the ϵ-greedy policy. Since an optimal ϵ-greedy policy still explores with a probability of ϵ, SARSA recognizes the risk that, if the robot move near the hole, such exploration would result in the robot fall into the hole. By keeping the robot further away from the hole, the optimal ϵ-greedy policy generated by SARSA reduces this risk.

4.2 Performance Analysis

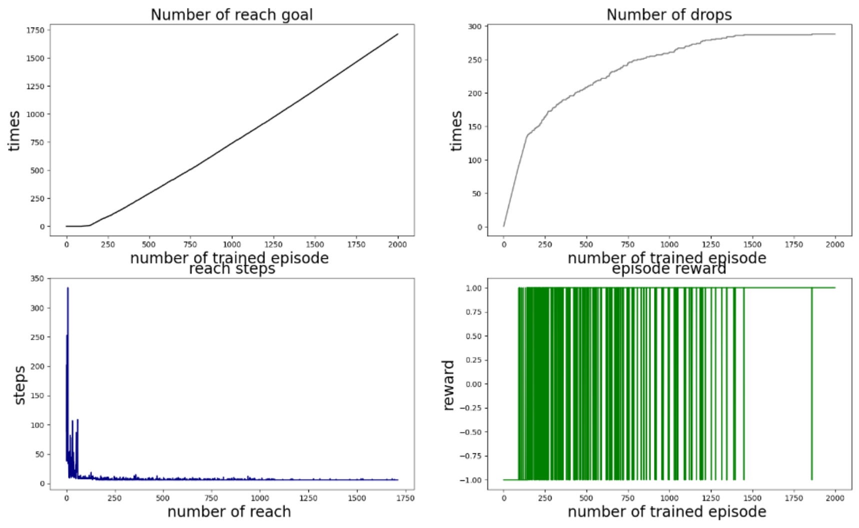


Figure 11 performance of Monte Carlo in 4*4 original environment

Figure 11 is the performance of Monte Carlo in 4*4 original environment, The first and second figure in figure 11 are the number of times the robot reached the goal and the number of times the robot fell into the hole during the training process. The third figure is the total step length required for the robot to reach the goal during the training process. The fourth figure is the cumulative rewards the robot has received in each episode. As can be seen from the figure above, since the number of falls stabilizes at about 1500 steps and the cumulative reward is always 1 with the exception of a few cases after 1500 steps (caused by exploration) , so the correct policy can be determined to be obtained around 1500, and the final policy can reach the goal in six steps.

In order to make it easier to read and save space, the performance graph and the hypermeter setting of each algorithm is no longer given here but a numerical conclusion is directly given.

The following tables are the performance of three different algorithms in three different environments, respectively. The ratio of reaching means the ratio of the robot reaching the goal during the whole training process, the higher the value, the faster the correct policy is obtained, the lower the cost during training, and the more stable the correct policy. The ratio of falling means the ratio of the robot falling into the hole during the whole training process. Its meaning is the opposite of the ratio of reaching. The episode number means the number of episodes required to get the correct policy, the higher the value, the slower the speed of getting correct policy. The episode length means the episode length needed for the final correct policy to reach the goal.

Table 1 performance of Monte Carlo in 4*4 environment

	ratio of reaching	ratio of falling	episode number	episode length
original	85.6%	14.4%	1500	6
reshape reward	87.85%	12.15%	1250	6
valid action	90.7%	9.3%	1250	6

Table 2 performance of Monte Carlo in 10*10 environment

	ratio of reaching	ratio of falling	episode number	episode length
original	0%	100%	None	None
reshape reward	83.518%	16.482%	75000	18
valid action	80.9%	19.1%	80000	18

Table 3 performance of SARSA in 4*4 environment

	ratio of reaching	ratio of falling	episode number	episode length
original	92.8%	7.2%	1000	6
reshape reward	93.45%	6.55%	1000	6
valid action	92.5%	7.5%	1250	6

Table 4 performance of SARSA in 10*10 environment

	ratio of reaching	ratio of falling	episode number	episode length
original	72.46%	27.53%	2000	18
reshape reward	89.6%	10.3%	1700	18
valid action	85.06%	14.93%	1700	18

Table 5 performance of Q-learning in 4*4 environment

	ratio of reaching	ratio of falling	episode number	episode length
original	92.15%	7.85%	1000	6
reshape reward	92.6%	7.4%	1000	6
valid action	93.75%	6.25%	900	6

Table 6 performance of Q-learning in 10*10 environment

	ratio of reaching	ratio of falling	episode number	episode length
original	48.63%	51.36%	2000	18
reshape reward	69.63%	30.36%	2000	18
valid action	75.06%	24.93%	1700	18

4.2.1 Effect of Different Algorithm

Comparing the figure above, we can see that Monte Carlo has the worst performance in a 4*4 environment, while Q-learning and SARSA perform similarly. In an environment of 10*10, SARSA performs better than Q-learning, and Monte Carlo performs worst.

- Why temporal-difference method faster than Monte Carlo? At the current time this is an open question in the sense that no one has been able to prove mathematically that one method converges faster than the other. In practice, however, TD methods have usually been found to converge faster than constant-α MC methods on stochastic tasks.

- Why SARSA performs better than Q-learning? The main different between SARSA and Q-learning is that SARSA is still exploring while learning the optimal policy, while Q-Learning directly learns the optimal policy. This makes Q-Learning possible to converge to the local optimum. This may be one of the reasons why the performance of Q-learning is worse than that of SARSA, and of course the unreasonable hyperparameter settings are also one of the reasons.

4.2.2 Effect of Different Environments

Comparing the above figure, we can see that in most cases the performance and efficiency of the algorithm can be improved by reshaping reward and using valid action. Especially, in the original 10*10 environment, Monte Carlo could not find the correct policy unless reshaping reward or executing only valid actions.

- Why reshaping reward can improve performance? This is because in the original environment, only sparse rewards can be obtained, that is, in most states, the agent will not receive any rewards and punishments, so most operations will not have an impact on the agent. The agent will not receive any positive feedback until they receive the first reward, so the model is likely to stop learning and cannot be improved. In addition to reshaping reward, this problem also can be solved by Curiosity Driven, Hindsight Experience Replay, Imitation Learning,etc.
- Why executing only valid actions can improve performance? This is because by executing only valid actions, the space of actions to be learned is reduced, thereby simplifying the problem and making it easier to learn the correct policy.

5. Difficulties and Solutions

When I first started training the algorithm, I found that it was difficult for the algorithm to find the correct policy in 10*10 environment ,so I improved the algorithm performance by using ϵ and learning rate decay.

- Exploration and exploitation dilemma is an important issues in reinforcement learning, In the early stages of learning, we hope that the policy will explore more states to find the optimal policy, but when the exploration is sufficient, we hope that the policy can be fully utilized to obtain the maximum reward. Therefore, an intuitive idea is to make the random sampling parameter ϵ of the greedy action decrease with the number of trained episodes

- The learning rate represents the update degree of the Q value. In the early stage of learning, the learning rate should be set to be relatively large to learn the correct policy faster. In the later stage, in order to stabilize the model and ensure the final convergence to the correct policy, a smaller learning rate should be used. Therefore, linear annealing is used for the learning rate, so that the learning rate decreases with the number of trained episodes.

6. list of initiatives

Random sampling parameter ϵ decay, Learning rate decay, Reshaping reward, Execute only valid action

7. Further improvement

The three algorithms all estimate value functions as a table with one entry for each state or for each state–action pair. they are limited to tasks with small numbers of states and actions and cannot handle large-scale tasks. The problem is not just the memory needed for large tables, but the time and data needed to fill them accurately. In many large-scale tasks, most states encountered will never have been experienced exactly before. The only way to learn anything is to generalize from previously experienced states to ones that have never been seen. One of the generalization methods is to use neural networks for function approximation. By combining deep learning and reinforcement learning, more powerful algorithms can be generated, such as DQN, A3C, DDPG, etc.

8.reference

[1] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[2] Chen C.Y.P.(2020). Deep Learning for Robotics .NUS press