ME5406 Deep Learning for Robotics

Project 1 – Frozen Lake Problem

Dinesh S/O Magesvaran

A0182885M

e0309680@u.nus.edu

Problem Statement

The problem statement for this project is the frozen lake problem that can be explored to learn more about the various reinforcement learning techniques. In this problem, there is a 4x4 grid that represents a frozen lake. The robot starts at the top-left most corner as shown in Figure 1, and is tasked to move through the frozen lake and reach the frisbee, which is located on the bottom-right corner of the map. However, the frozen lake consists of holes which are covered with a thin layer of ice, which the robot has to avoid falling into.

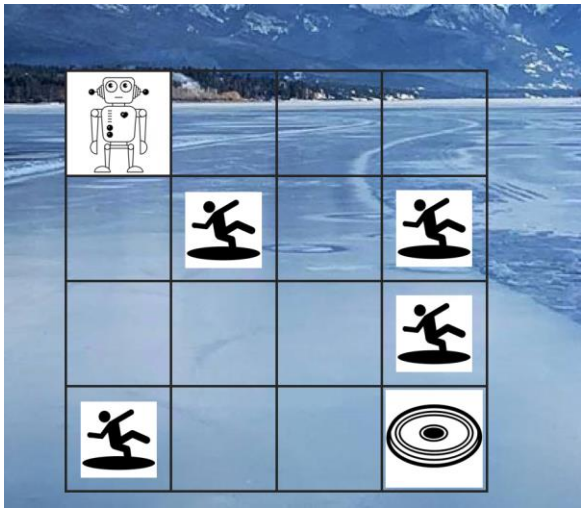

Figure 1: Illustration of Frozen Lake, taken from Project 1 requirements

States

There are 4 types of states, and 16 states in total. The types of states are as follows:

1. Starting State: where the robot is initially located
2. Goal State: which is the state with the frisbee
3. Holes: where the robot will fall into the hole
4. Frozen Surface: where it is safe for the robot to move onto

If the robot is at the goal state, or the hole state, the episode ends.

Actions

At any state, the robot is able to take 4 actions. The possible actions are as follows:

1. Up (Represented by number 0 in code implementation)
2. Down (Represented by number 1 in code implementation)
3. Left (Represented by number 2 in code implementation)

4. Right (Represented by number 3 in code implementation)

The robot is to be confined within the 4x4 grid. Therefore, if the robot takes an action that would bring it outside the grid, it goes back to the same state that it took the action from. This can be visualised as there being walls surrounding the grid which restricts robot movement to the grid.

Rewards

There are 3 rewards that the robot can receive, determined by the outcome of the state-action pairs of the robot. The possible outcomes are as follows:

1. The robot reaches the goal (Reward: +1)
2. The robot falls into a hole (Reward: -1)
3. The robot moves onto a frozen surface (Reward: 0)

Techniques

There are three techniques that will be explored and implemented in this project. They are as follows:

1. First-visit Monte Carlo control without exploring starts
2. SARSA with an ϵ-greedy behaviour policy
3. Q-learning with an ϵ-greedy behaviour policy

The 3 techniques will be referred to as Monte Carlo, SARSA and Q-learning respectively from here onwards.

Implementation

The entire implementation for all techniques was done on 1 python file 'main.py' to make submission of the code easier. The code is split into 4 parts.

1. Environment, implemented as a class
2. Monte Carlo, implemented as a function
3. SARSA/Q-learning, implemented as separate functions that both call the function TemporalDifference()
4. Main Code

Environment

The environment for the frozen lake was coded from scratch as a class, with similar functions to the OpenAI Gym, such as render(), step() and reset(). Additional functions that will be

useful for the technique implementations were also added. The environment contains 2 frozen maps that can be selected from. The 4x4 map will be used for the basic implementation, and the 10x10 map will be used in the extended implementation. The render() function prints the frozen lake, showing both the current state and the past states. This helps to easily determine the path that the robot took if it did not revisit the same state twice. The function is useful for clearly displaying the optimal policy after the learning has ended.

Technique 1: Monte Carlo

Monte Carlo was implemented as a function MonteCarlo() with 3 parts. The first part contains the function variables and some sub-functions to help compute certain values. This was done to make the technique implementation code simpler to read. The second part contains the learning code that determines an optimal policy based on the Monte Carlo technique. The third part contains the final policy code that obtains the optimal policy and calls the render() function of the environment at the end to show the policy in a pictorial view.

Technique 2: SARSA

SARSA was implemented using 2 functions. The first function is the TemporalDifference() function that contains the entire implementation of SARSA. This function was implemented in 3 parts similar to Monte Carlo. The second function is SARSA() which calls the first function to implement the SARSA technique. This was done so that the code in TemporalDifference() can be reused for Q-learning, which has an almost identical implementation.

Technique 3: Q-learning

Q-learning was implemented in a similar way to SARSA, using the same TemporalDifference() function and the QLearning() function. The QLearning() function calls the TemporalDifference() function to implement the Q-learning technique instead of the SARSA technique, which involves changes to only 1 or 2 lines of code being carried out differently.

Main Code

The main code consists of 6 parts, which correspond to the 3 techniques being done in both basic and extended implementation. Each part calls the respective functions MonteCarlo(), SARSA() or QLearning(), and passes the desired parameters and map size accordingly. Figure 2 below summarises the code implementation.

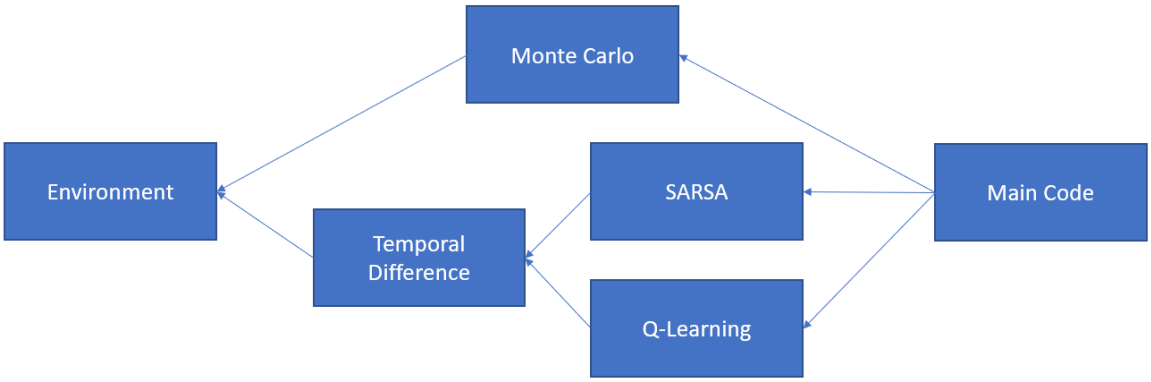

Figure 2: Summary of code implementation

Evaluation

As the task of the robot is to reach the frisbee, the three techniques will be evaluated based on the success rate of the robot reaching the frisbee in a finite number of steps. As the task only focuses on completion, the number of steps taken by the policy to reach the frisbee will not be taken into account for evaluating the three techniques.

Performance Evaluation Scheme

The robot will be repeatedly given the 4x4 map size and asked to find the solution by using each technique 100 times. Subsequently, the same would be repeated for the 10x10 map size. For each iteration, if the optimal policy of the robot after learning gets the robot to the frisbee, it is considered to be a successful iteration. Out of the 100 iterations done per technique per map size, the number of successful iterations will be counted and displayed as a percentage success rate. 0% would correspond to the robot reaching the frisbee in 0 out of 100 iterations, while 100% would correspond to the robot reaching the frisbee in 100 out of 100 iterations.

Performance

Table 1 below shows the parameters used for the evaluation of the three techniques. Table 2 below shows the results of the evaluation when the number of episodes is set to 10,000. Table 3 and table 4 show the results for the number of episodes set to 1,000 and 100 respectively.

Table 1: Parameters used for the evaluation of the respective techniques

Reinforcement Learning Technique	ϵ (for ϵ-greedy policy)	Discount Value (γ)	Learning Rate (α)	Number of episodes
Monte Carlo	0.1	1		100/1000/10000
SARSA	0.1	1	0.1	
Q-learning	0.1	1	0.1	

Table 2: Success rate of respective techniques for number of episodes of 10,000

Reinforcement Learning Technique	Success Rate (%)	
	4x4 map	10x10 map
Monte Carlo	99	0
SARSA	85	72
Q-learning	**100**	**100**

Table 3: Success rate of respective techniques for number of episodes of 1,000

Reinforcement Learning Technique	Success Rate (%)	
	4x4 map	10x10 map
Monte Carlo	92	0
SARSA	88	**93**
Q-learning	**100**	32

Table 4: Success rate of respective techniques for number of episodes of 100

Reinforcement Learning Technique	Success Rate (%)	
	4x4 map	10x10 map
Monte Carlo	59	**0**
SARSA	**100**	**0**
Q-learning	**100**	**0**

Discussion

1. Compare and contrast the performance of the three reinforcement learning techniques and the results that they have generated in your implementations.

Comparing the three techniques for the 4x4 map, Monte Carlo's best success rate is 99% at 10,000 episodes, while SARSA's best success rate is 100% for 100 episodes. Q-learning achieves a success rate of 100% at all three episode numbers. As number of episodes is not a criteria for evaluation, SARSA and Q-learning are both equally good at solving the 4x4 problem, while Monte Carlo is slightly worser than the two.

Comparing the three techniques for the 10x10 map, Monte Carlo's achieves 0% for all 3 episode numbers. SARSA is able to reach 93% at 1,000 episodes, while Q-learning is able to reach 100% at 10,000 episodes. Hence, Q-learning performs the best, while SARSA is second best and Monte Carlo performs the worst for the 10x10 problem.

The number of episodes is a factor in determining how well the technique performs. For Monte Carlo and Q-learning, more episodes generally leads to better performance. However, SARSA performs better at lesser episodes and worse at more episodes.

The map size is also a factor in determining how well the technique. All 3 techniques perform better the smaller the map is. In particular, Monte Carlo is unable to solve the larger 10x10 problem.

2. Provide explanation for any similarities or differences observed.

Monte Carlo

Overall, Monte Carlo performs the worst in solving the two 4x4 and 10x10 problems. This is due to the way the q-values are computed. When Monte Carlo fails to find a way to the frisbee, it usually results in an optimal policy that choses an action at a state that attempts to go out of the 4x4 or 10x10 grid. Hence, this would result in the robot being stuck at a 'wall'. This happens by chance. When the robot moves around and runs into holes, it will start getting q-values of -1 for all four of its actions due to a discount value of 1. When all four actions have the same q-value at a state, the robot has to take actions at random, as there is no information to exploit. When the robot happens to reach the goal by chance, the q-values will start to change as the returns which previously only consisted of '-1' now has a '+1'. However, due to the random nature in which the state-action pairs were selected, some state-action pairs were visited on more episodes and have more '-1' returns than others.

When the first '+1' is introduced, there are 2 possible cases for each state. The first case is that at a state, only 1 state-action pair has been selected before on the episode that reached the goal first. When this happens, the action that was chosen on this state that led to the goal will have a '+1' added to its returns, which will cause the q-value for the particular state-action pair to become less negative. As the other actions at that state still have q-values of '-1', the robot will now carry out exploitation most of the time and choose the same state-action pair that led it to the goal. This leads to successfully finding a solution to the problem.

The second case occurs when at a state, more than 1 state-action pair was chosen before on the episode that reached the goal first. When the q-values are updated for this case, more than 1 state-action pair's q-values are updated to become less negative, with state-action pairs that have accumulated lesser '-1' returns in the past giving the least negative value. This least negative value could be any of the state-action pairs that were updated, as previously, actions were taken at random and '-1' returns were also accumulated at random. If the highest q-value happens to be one where the robot attempts to go out of the grid, it will keep exploiting that knowledge most of the time, resulting in the robot getting stuck at the wall.

For the 10x10 problem, Monte Carlo is unable to solve the problem all the time as its relies on chance to reach the goal. As the state-action space is much larger, the robot never reaches the goal, resulting in the problem remaining unsolved as all state-action pairs have '-1' q-values. The diagram below summarises the explanation for Monte Carlo's performance.

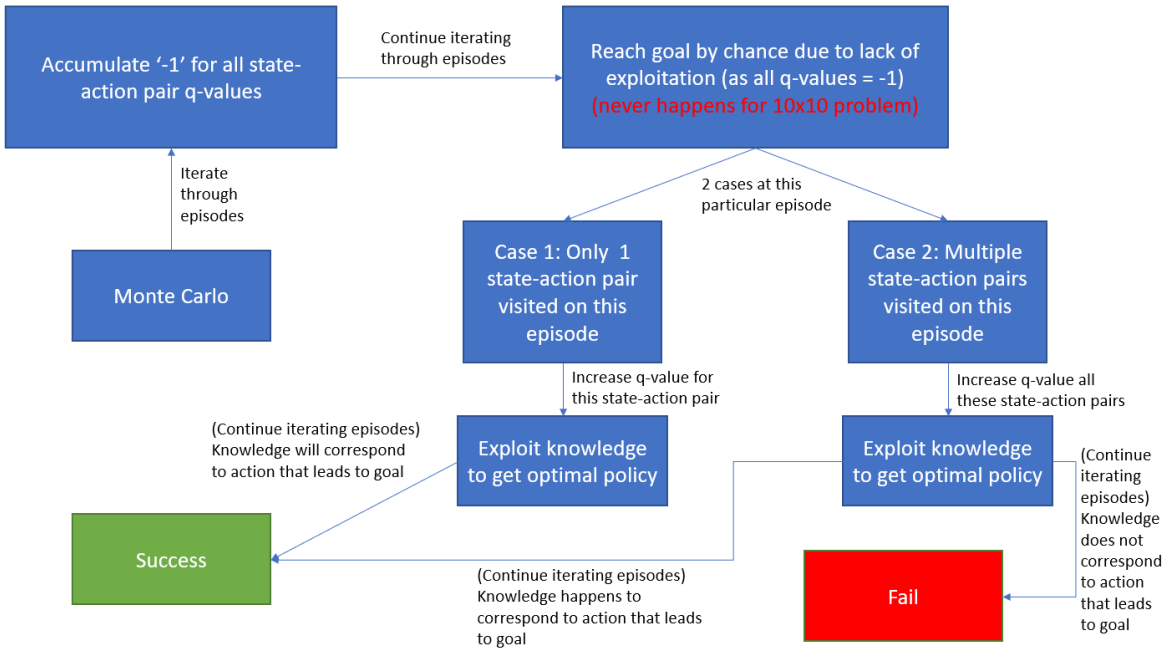

Figure 3: Summary of Monte Carlo performance

The number of episodes can help increase the chances of Monte Carlo finding a solution. This is due to the exploration element. Even though some cases lead to exploiting an action at a state that does not lead to the goal, the robot may explore and take another action, increasing its chances of reaching the goal again after the first time. This exploration may correct the q-values in the right direction so that the robot improves its optimal policy. However, as exploration occurs much less frequently than exploitation, this happening is entirely by chance, with the chances increasing as the episodes increase.

Monte Carlo vs Temporal Difference methods

In Temporal Difference methods, such as SARSA and Q-learning, q-values are updated differently. As q-values for state-action pairs are only updated based on the q-values on the next state-action pairs and rewards for reaching the next state, when the robot runs into a hole, only the state-action pair that leads to the robot moving into the hole will have its q-value updated to a negative number. This means that the robot will not be discouraged to take the same path, but it will only be discouraged to take the last step that leads it to the hole, effectively navigating the robot away from the hole. As this continues, the robot will eventually reach the goal. In Monte Carlo, the robot is discouraged to take the entire path just because of the mistake at the last step, affecting all q-values to quickly become -1 and losing the ability to exploit. Therefore, Temporal Difference methods are generally better than Monte Carlo in solving the given problem.

SARSA vs Q-Learning

Despite being largely similar in terms of implementation, both SARSA and Q-learning produces very different results. SARSA performs better with less episodes, and gets worse in the long run as the number of episodes increases. Q-learning performs better with more episodes, and gets better in the long run. There are 2 reasons that explain the behaviour of SARSA and Q-learning.

SARSA is an on-policy technique that determines its q-values based on the actions that it will take. Q-learning is an off-policy technique that determines its q-values based on the greedy action. Due to the difference in the action used in computing q-values and the action that the robot actually takes during learning, Q-learning takes a longer time to converge at the optimal policy. Similarly, as SARSA computes q-values with the same action that the robot takes during learning, it converges faster at the optimal policy. This explains why at lower number of episodes, SARSA performs better than Q-learning.

Once SARSA has learned an optimal policy that solves the problem, continued exploration is still able to change the q-values, which may move the q-values away from the desired optimal policy. This does not happen in Q-learning, as q-values are calculated based on the greedy action. Therefore, once optimal policy is able to solve the problem, Q-learning's policy is unaffected by the increase in episodes, while SARSA's policy is affected by increase in episodes.

Both SARSA and Q-learning achieved 0% for 100 episodes at the 10x10 problem. This is due to the fact that the policies have yet to converge at 100 episodes.

Q-learning was able to reach 100% for both 4x4 and 10x10 problems, while SARSA's best was 100% for 4x4 and 93% for 10x10. For SARSA, too little episodes is bad as the policy has yet to converge, while too many episodes is bad as the policy becomes worse after reaching the desired optimal policy that solves the problem. Hence, there may be a possibility where SARSA is also able to achieve a success rate of 100% in the 10x10 problem if the number of episodes is set correctly. However, after trial and error of changing the number of episodes from 1000 to various values near 1000(eg: 500 to 2000), it was observed that the success rate was never 100%. Therefore, Q-learning has an overall better success rate than SARSA.

Q-learning has better success than SARSA due to the optimal policies learned by the 2 techniques. Q-learning learns an optimal policy, while SARSA learns a near-optimal policy. SARSA's policy is not optimal due to the element of exploration being able to affect the q-values, which is not the case for Q-learning.

The general observation was that smaller problems were easier to find solutions to than the larger problems. This is due to the number of state-action pairs being much higher for the larger problem, resulting in state-action pairs being visited less frequently. The best optimal policies are found when state-action pairs are visited infinitely often (or sufficiently many times). This means larger problems should be solved by using a larger number of episodes.

3. Discuss any simulation outcomes that appear unexpected and provide plausible explanations.

For SARSA and Q-learning, the epsilon value was taken to be 0.1. As the epsilon value determines whether the actions taken lead to exploration or exploitation, epsilon is usually taken to be a non-zero value between 0 and 1, due to the importance of both exploration and exploitation. However, when epsilon value is 0 for this problem, SARSA and Q-learning are able to find a solution to the problem. In fact, it is much better than an epsilon value of 0.1. In the evaluation section, both SARSA and Q-learning have a success rate of 0% at 100 episodes. When epsilon is replaced with a value of 0, the success rates become around 98% for both.

Taking the epsilon value as 0 is an interesting case, as SARSA and Q-learning essentially become the exact same algorithm. Q-learning behaves like it is on-policy as the action taken is always greedy. Similarly, SARSA behaves as if it is using Q-learning's update rule for q-values, as the next action taken is always greedy. From the simulations, it shows that 0 exploration and full exploitation is best. This is due to the reward system defined by the problem. As the holes give a reward of '-1' it effectively cancels out the state-action pair that leads to the hole from ever being chosen again, unless all 4 actions at a state reach the same q-values, which is unlikely. This effectively steers the robot away from holes while trying to find the goal, greatly expediting the process of reaching the goal. An epsilon value of zero would not work if there were other sources of positive rewards besides the goal. This is because once it got the positive reward, it will always undergo exploitation to get that same positive reward that is not the goal. With the combination of undesirable terminal states giving negative rewards and the goal being the only source of positive rewards, an epsilon value of 0 gives the best outcomes for both SARSA and Q-learning.

Epsilon value of zero was not used in evaluation as it would not provide any insight to the differences between SARSA and Q-learning.

4. Highlight the difficulties encountered and describe how they were overcome during the project.

One of the earlier difficulties encountered was knowing whether the robot is not finding its way to the goal due to bugs in the code implementation, or due to the limitations of the technique used. At the earlier stages, there were a number of bugs fixed before the robot found its way to the goal. For Monte Carlo extended implementation on the 10x10 grid, the robot was unable to find its way to the goal, leading to the same question. The answer was found by tracing the return values and q-values, which lead to the discovery of the limitations of Monte Carlo.

Another difficulty was related to the environment. As the frozen lake environment was coded by taking inspiration from the frozen lake environment in the OpenAI Gym, the map of the lake rendered only showed the current position of the robot. This made it harder to trace the

path of the robot through the map. Hence, previous states visited were also highlighted in the rendering to show the path taken by the robot.

During evaluation, the techniques had to be executed numerous times. Initially the techniques were planned to be executed 1000 times per technique. However, this ended up being too time consuming. Hence, the number of times executed was reduced to 100 instead. The limitation of such a solution is that the evaluation results may be less accurate. This limitation was acceptable as the purpose of the evaluation was simply to compare the three techniques, and slightly less accurate values are still useful for comparison if they are not too close to each other.

5. Elaborate on your initiatives (if any) to investigate and improve the efficiency of these techniques in solving the given problem.

Efficiency can be seen as trying to balance getting the best solution and spending the least time learning. As epsilon value of 0 for SARSA and Q-learning was the best value for this problem, it is used. Two parameters were investigated to increase efficiency, learning rate and number of episodes. Increasing the learning rate did not really improve the efficiency as the success rate was the same and the time taken to execute the code did not change much. Decreasing the number of episodes was able to get the code to execute much faster. In particular, at 100 episodes, the code executed very fast while still giving success rates above 95%.

For Monte Carlo, the number of episodes and the discount rate were investigated for the 4x4 problem. Changing the discount rate to a value less than 1 resulted in worser success rates. Reducing the number of episodes also reduced the success rate by a lot. Hence, Monte Carlo proved to be challenging to improve its efficiency.

6. Present any other issues that you think are significant (and explain why) concerning your implementation of the three reinforcement learning techniques.

In my implementation of the three techniques, I used sub-functions to help compute certain values for ease of readability and implementation. However, function calls require time. In reinforcement learning, as the functions are called numerous times, this could potentially lead to slower execution times. Therefore, it may have been better to implement the three techniques without the use of sub-functions.

There is a possibility that the 4x4 and 10x10 grid used for the implementation may have an effect on the performance of the three techniques. This could be due to the specific behaviours of the techniques, such as the risk-adverse nature of SARSA or the risky behaviour of Q-learning, leading to different outcomes. However, only one 4x4 and one 10x10 grid was used as expected by the project. Trying out different grids besides the two may or may not give more insight to the varying performances of the three techniques. Probabilistic transitions may also may similarly produce different results to the deterministic case.