CS 4641 PROJECT 4

Name: CHANG Yingshan
GT Account: ychang363
GT ID: 903457645

Problem Description

1. Reinforcement learning

Reinforcement learning is a type of Machine Learning, which allows agents to automatically determine the ideal behavior within a specific context, in order to maximize its "reward". It is characterized by a process of agents observing the environment output consisting of a reward and the next state, and then acting upon that.

2. Markov Decision Process

Markov Decision Process formally describe an environment for reinforcement learning, where the environment should be fully observable. In any MDP problem, we assume the Markov Property: the effects of an action taken in a state depend only on the current state and not on the prior history. The following items are given in a typical MDP model:

1) State Space S: a set of possible world states.
2) Action Space A: a set of possible actions
3) A real value reward function R.
4) Transition Model T: a description of each action's effects on each state.

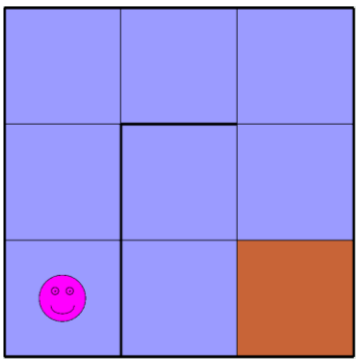
3. Grid World Problem

For this project, I have explored the Grid World Problem by multiple approaches. The Grid World problem is a classic problem for MDP because it models the exploration process of practical problems. In the grid world, the agent sets out from a starting point and must traverse a grid-like square to reach a goal. The State Space is all the possible squares that an agent can stay in. In any state, an agent can choose an action from going up, going down, going left and going right. Each action taken causes a constant "cost", which urges the agent to find the goal as soon as possible instead of moving around in the grid world for unnecessarily long times. There is no reward given for reaching the goal state, nor is there a "cost" for it.

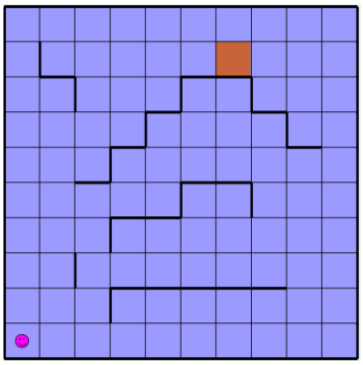
Moreover, there are "walls" on the four sides of the maze, as well as some particular places in the middle of the maze. Any action taken towards a wall will result in the agent staying at its original state. Additionally, "bumping into a wall" will result in a negative penalty of 50.

The transition model for this problem is a non-deterministic one. With probability 1-PJOG, (0<PJOG<1), the agent will move towards the target direction, while with probability PJOG/3, the agent will move towards each of the remaining directions.

I defined two Gird World Problems using a simple maze with only 9 states and a complex maze with much more states and walls. The size different would help us understand the challenges we can expect to encounter as the problem scales up.



Simple Maze



Complex Maze

The starting point for both mazes is the bottom-left corner and the goal is highlighted in orange color. The walls are highlighted by bold black lines. The target of this paper is to solve the two Grid World Problems using different reinforcement learning techniques, and compare the algorithms' performances.

Why interesting?

The Grid World Problems are interesting because they represent simplified versions of real-world environments in which a robot could navigate and seek goals.

On one hand, the problems well simulate realistic scenarios:

1) The goodness of taking a certain action or being in a certain state is not known beforehand, the environment must be explored to generate estimates, after which an agent can greedily seek to maximize reward by exploiting its understanding of the environment.
2) Sometimes things do not turn out as planned. The agent does not follow its target direction by some small probability. This adds randomness and noise to the problem, which is similar to the realistic world.

On the other hand, the problems are simple enough for implementation and management:

1) The sizes of state space and action space are reasonable.
2) When we evaluate the solutions, the goodness of a policy can be judged intuitively.
3) As the algorithm runs, we can control the whole picture

Value Iteration & Policy Iteration

1. Overview

Value Iteration

Value iteration is implemented by dynamic programming, which means at each time step, the utility of each state is computed from the values in the last time step, according to the Bellman equation.

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

The utility of a state is defined as "expected long-term sum of discounted reward by executing the optimal policy". Each iteration is also known as a Bellman Update, which is a contraction function. Therefore, Value Iteration is guaranteed to converge to the optimal policy if we repeat the update again and again. Practically, we can stop iteration if the utility difference of any state before and after an update is smaller than e (threshold). After getting the true utility, we can derive the optimal

policy of each state.

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) U(s')$$

It is noteworthy that in our problems, the values are considered as "penalty", so a negative sign should be added to the value of each state.

Policy Iteration

Policy iteration manipulates policy directly, rather than finding the optimal policy indirectly via the optimal value function. PI algorithm alternates the following two steps, beginning from some initial policy π0:

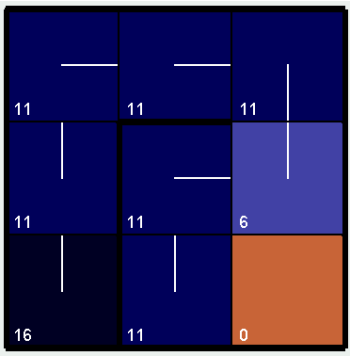
1) Policy evaluation: given policy πi, calculate the utility of each state if πi were to be executed.
2) Policy improvement: calculate a new policy πi+1, using on-step look-ahead based on the utility calculated in step 1.

The algorithm should terminate when the policy improvement yields no change in the utilities. At this point, the utilities much be the fixed point of the Bellman Update, and the policy should be the optimal policy. Because there are finitely many policies for a finite state space, and each iteration can be shown to yield a better policy, so PI is guaranteed to explore every possible combination of policies and converge to the best one.

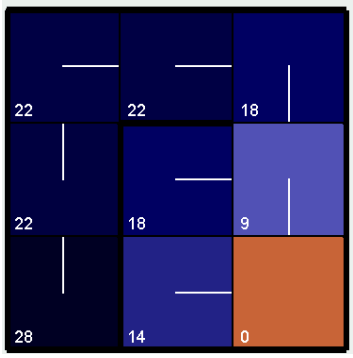
repeat
  $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
  $unchanged? \leftarrow$ true
  for each state s **in** S **do**
    if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
      $\pi[s] \leftarrow \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a)\ U[s']$
      $unchanged? \leftarrow$ false
until $unchanged?$
return π

2. Results

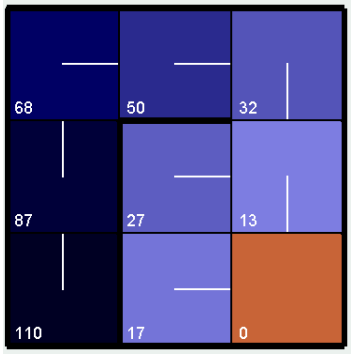
For the simple maze, VI converges after 34 steps (PJOG = 0.3). However, when I run the algorithm one iteration at a time, I noticed that actually the correct policy has already been found after 2 steps. The results of VI on the simple maze are shown as follows.



VI after 1 step

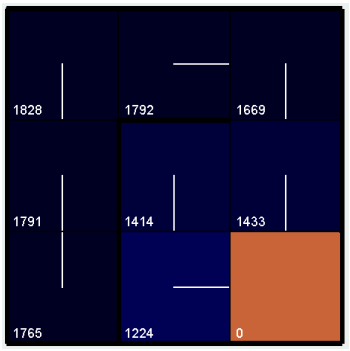


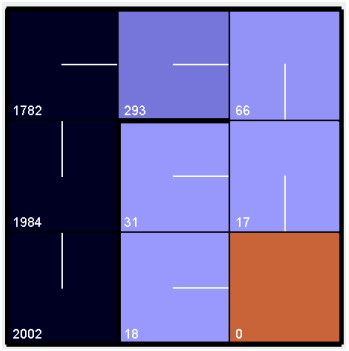
VI after 2 steps

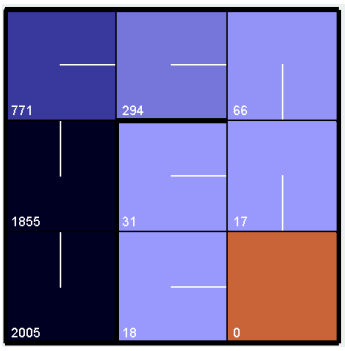


VI convergence result

PI needs much less steps to converge. It converges only after 3 steps on the simple maze (PJOG = 0.3) the results of PI on the simple maze are shown as follows.

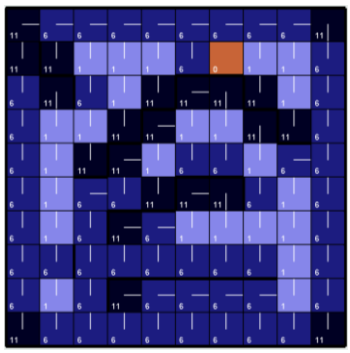


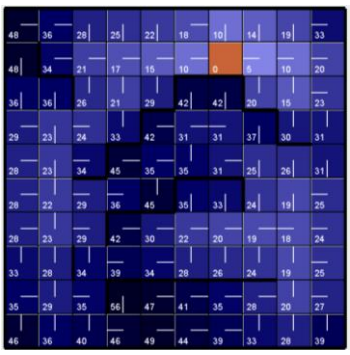


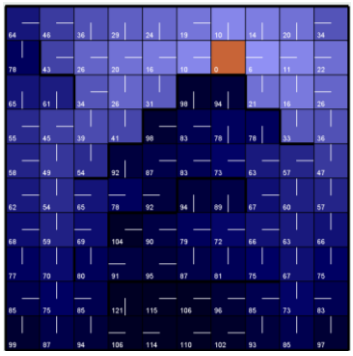


PI 1 steps PI 2 steps PI convergence result

For the complex maze, there is a noticeable growth in the number of steps needed for VI to converge. Choosing PJOG to be 0.3, VI converges after 76 steps. Again, it is notable that the VI has "almost" found the optimal policy only after 10 steps. The rest iterations only fine tune the policy and update the utilities. The results of VI are shown as follows.

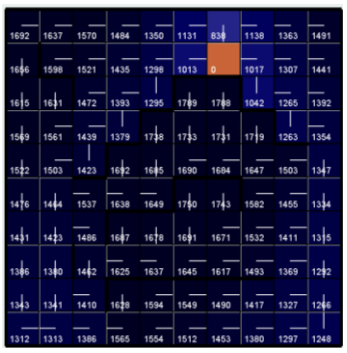


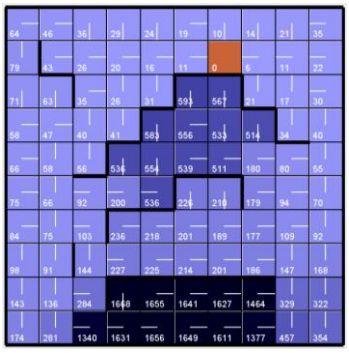


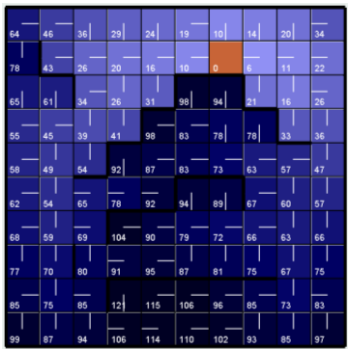


VI 1 step VI 10 steps VI convergence result

With a much larger state space, the number of steps to convergence also increases for PI. Moreover, PI's advantage over VI in speed is more remarkable on the complex maze. Choosing PJOG to be 0.3, it uses 6 steps to find the policy. Here is the result for PI on the complex maze.







PI 1 step PI 3 steps PI convergence result

3. Convergence analysis

Simple Maze: PJOG	0.1	0.3	0.5	0.7
VI steps to convergence	16	34	80	354
PI steps to convergence	3	3	2	2

Complex Maze: PJOG	0.1	0.3	0.5	0.7

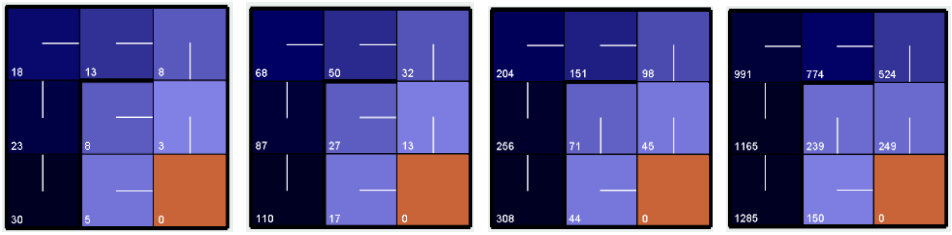
VI steps to convergence	36	76	184	1650
PI steps to convergence	8	6	5	7

It is clear that PI is significantly faster than VI in terms of number of iterations. This is because the policy usually stops changing long before the utility of each state stops changing. As a matter of fact, PI's execution time on each iteration is also usually less than VI's, because when evaluating the utility of each state, PI follows a fixed policy while VI has to loop over all possible actions that could be taken.

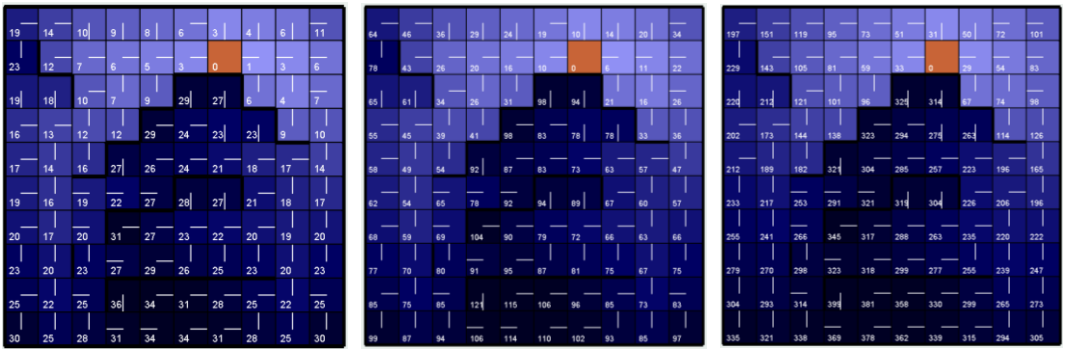
When comparing the convergence speed of VI and PI with different PJOG, it is clear that VI's number of steps to converge rockets up dramatically as PJOG goes up. This is because high PJOG value adds much noise to the environment, and this strongly impact the convergence speed of VI. By contrast, PI's number of steps to converge almost remains the same. This could be attributed to the fact that PI actually run the "simplified Bellman Update" multiple iterations before the policy for the next time step is evaluated. Thus, the convergence speed of PI seems "immune" to this kind of noise.

4. Do VI and PI converge to the same result?

I applied both VI and PI to the two mazes with different values for PJOG. VI and PI output the same result for both optimal policy and true utility all the time. Here are the final results.



PJOG = 0.1 PJOG = 0.3 PJOG = 0.5 PJOG = 0.7



PJOG = 0.1 PJOG = 0.3 PJOG = 0.5

It is interesting to analyze the ways in which the optimal policies and true utility are different when using different PJOG. Firstly, the optimal policy reflects the priorities of reaching the goal and avoiding walls to a certain extent. When PJOG is small, the agent will follow its target direction in most times. In order to avoid walls, the optimal policy let it head away from the wall. For example, in many grids on the very left side, the optimal policy is "going right". When PJOG is large, there is more uncertainty in the agent's behavior. The optimal policy focuses more on reaching the goal instead of trying to avoid walls, since there is a large probability that the agent won't do what it is supposed to do. In this case, reaching the goal leads to termination of the game, which is the best way to permanently get rid of the increment of negative reinforcement.

Secondly, the values of states are increasing significantly as PJOG grows, which means the expected sum of long-term utility is more negative when PJOG is larger. This can be explained by the fact that the uncertainty of the agent's behavior results in more accidental collision with the wall. In addition, when PJOG is larger, it also usually takes the agent more time to reach the goal since it does not always follow the correct policy to head toward the goal, which incurs more negative path cost. Moreover, it is more obvious from values in complex maze that, when PJOG is larger, the increase in values is particularly evident for states near walls. This is because, in those states, it is expected that the "disobedient" agent has larger risks of hitting the wall.

Q-Learning

1. Overview

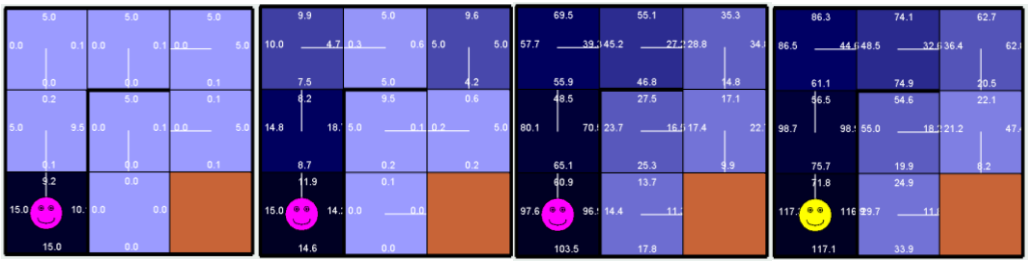
One of the biggest drawbacks of VI and PI is that they require the model (the transition model and reward function) to be known, which may not be the case in many practical problems. Q-learning is a model free algorithm that finds the optimal policy directly without the domain knowledge to calculate the transition and reward function. The agent learns how to behave by moving in the environment following certain rules. When it takes action a in state s, it senses the environment and retrieve the following information: "I'm in state s' after taking action a in state s"; "I receive a reward r". Based on that information, the agent keeps on calculating and updating the Q values of "taking a certain action in a certain state" (Q(s,a)). As the agent explore the environment over and over again, Q values will converge to its true values for each state. Q values are directly related to utility values as follows: $U(s) = \max_a Q(s,a)$. Thus, Q values may seem like another way of storing the utilities. For each explorative step (s, a, s') the agent takes in the environment, it updates the Q values using the following equation.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

The optimal policy can also be derived from Q values: $\pi^*(s) = \arg\max_a Q^*(s,a)$. As I have mentioned before, since the values shown in the maze represent "penalty", so the optimal policy will point to the direction that minimizes values shown in the maze.

2. Results

Simple maze

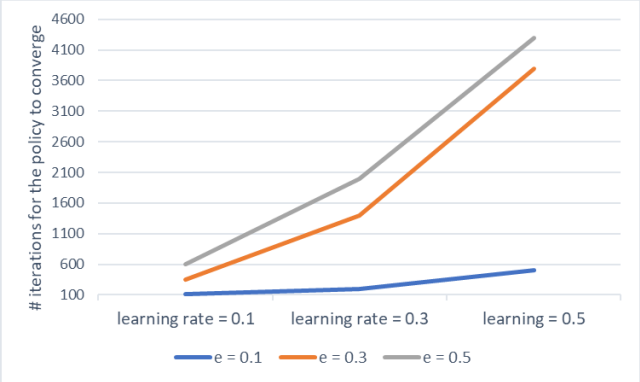


1 episode | 2 episodes | ~120 episodes (policy has converged) | ~200,000 episodes (values almost converge)

For the simple maze, Q-learning runs much faster than both VI and PI, since it only takes constant time for each transition. However, it takes much more iterations to converge. The policy produced by Q-learning usually converges after ~120 episodes, while the Q values are still fluctuating within a small range. I tried three ways to manage the convergence speed on the simple maze.

1) Increase learning rate
   Learning rate determines by how much the Q values are updated for each transition. Small learning rates make the learning very slow. However, we also need to keep in mind that, high learning rates also make it hard for the system to stabilize, especially in noisy environments.
2) Increase epsilon
   For Q-learning to converge, each state-action pair needs to be explored a considerable number of times. Low exploration-rate makes it easy for the agent to get stuck in local optima. Although part of the exploration problem can be solved by the non-determinism of each transition (PJOG>0), we can still force more exploration by increasing epsilon.
3) Increase the number of episodes
   This is the surest way to make the algorithm converge, because, ideally, we want infinitely many episodes.

Here is the summary of the convergence speed on the simple maze, with different learning rates and epsilons.

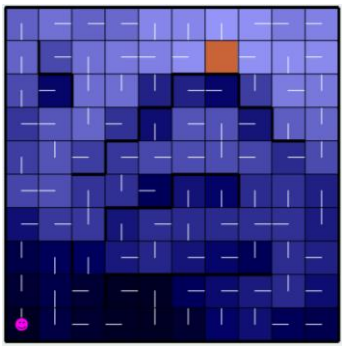


It can be concluded from the graph that both high learning rate and high epsilon makes the system unstable and hard to converge. Especially when the environment is already noisy (epsilon = 0.3 or 0.5), increasing learning rate makes the matter worse.

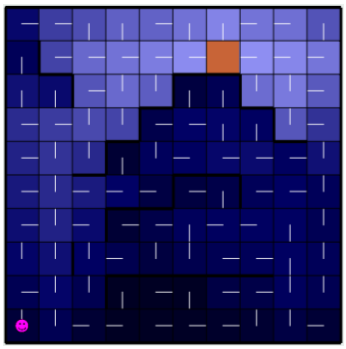
What's more, we do not see any positive effect of encouraging more exploration. There are two possible reasons. For one thing, e=0.1 may have already provided a good balance between exploration and exploitation. Increasing epsilon's value beyond 0.1 may simply add more uncertainty and noise. For another thing, the state space for the simple maze is small and easy to be fully explored. There is also no tricky local optimum, making it unnecessary to explicitly "force" exploration.

Complex maze

Here it the result for the complex maze, with PJOG=0.3, epsilon=0.1, learning rate = 0.1

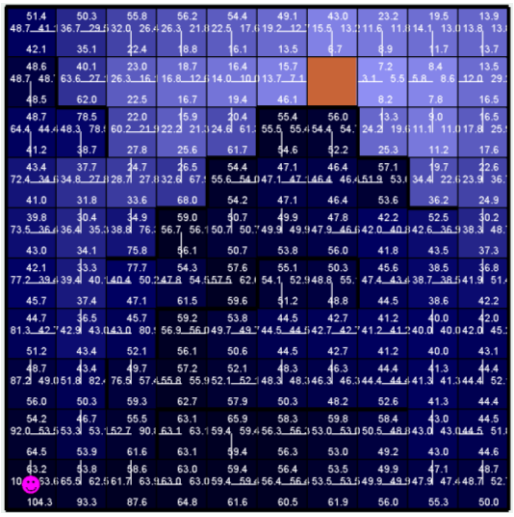


50 episodes

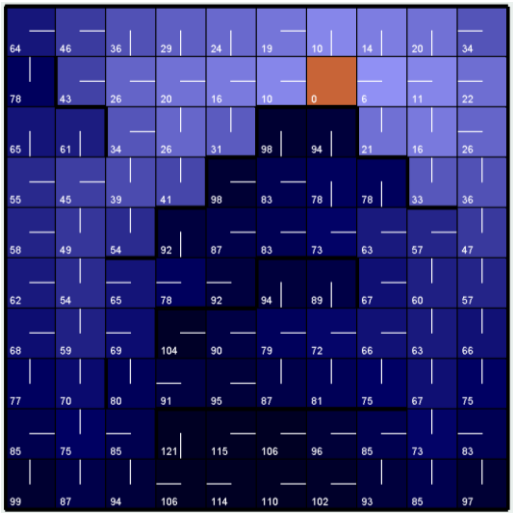


10,000 episodes

For the large maze, Q-learning is able to learn some important features of the grid world even after few episodes, such as “moving towards the goal is optimal”, “moving towards a wall is non-optimal”, “moving towards a corner that is surrounded by walls is non-optimal”. The plus side of Q -learning is that the update of Q values for states which will be frequently visited is very fast. After 10,000 episodes, we can see that Q-learning has find the optimal policy for most states, except for states near walls or corners.



Q-learning 1,000,000 episodes



Optimal policy found by VI

However, it is extremely hard for it to find the optimal policy for the “least frequently visited states”, which is the downside of Q-learning. Even after 1 million episodes (at that time the values are only fluctuating within a tiny range), it still produces non-optimal policy for a few states with low exploration rate.

To sum up, it is remarkable how Q-learning well performed within a short period of running time and how quickly it finds the optimal policy for most of states. Those states that are more likely to be visited by a Q-learning agent are usually the states that we really care about in realistic problems. Therefore, Q-learning is consistent with the problem-solving process of many practical problems: devote more time to efficiently solving “common problems” and less time to “rare problems”. This characteristic makes Q-learning generalize well to problems with large state space (the size of state space can even reach infinity), because in a large state space, only a finite number of states carry the most significance and we do not require or even care about convergence for the rest of states.

3. Exploration V.S. Exploitation

When the agent revisited the explored states, it uses the “epsilon-greedy” policy to choose its action.

Which means it has probability 1-epsilon to choose the action with maximum Q value, and has probability epsilon to ignore Q value and pick an action randomly (0<epsilon<1).

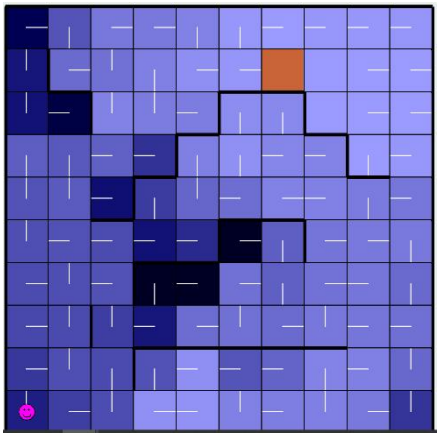
The choice of epsilon is vitally important because it balances exploration and exploitation. If the agent picks the action that maximizes Q value every time (epsilon ~ 0), there is no exploration because the agent will be completely influenced by its previous experience, making no effort to explore new states. If the agent picks a random action every time (epsilon ~ 1), there is no exploitation since the agent's behavior will become a random walk, without taking advantage of its previous experience to seek the goal. We need to balance exploration and exploitation. On one hand, we what to get the optimal policy for every state, which means the agent needs to visit every state for multiple times. On the other hand, we still hope that the agent can reach the goal with as little penalty as possible, which means it needs to move toward the direction with lower expected long-term penalty.

Actually, there are two ways to enhance exploration. One is choosing large epsilon, the other is choosing large PJOG (because an "disobedient" agent will accidently fall into unexpected states). In our own case, I fixed PJOG to be 0.3 and only modified epsilon to balance exploration and exploitation.

I tried to find the effects of different epsilon by recording the number of episodes needed until the agent has visited every state at least one time. (The agent executes an entire episode by starting from the start state and keeping transitioning from one state to another, updating the corresponding Q values until it reaches the goal). Since there is some inherent randomness in the algorithm, the number of episodes needed are not always the same. So, I repeated this procedure (i.e. let the agent execute episode one by one until it has visited every state at least once) 10 times and recorded all 10 results. In the following graph, the y-axis represents the number of episodes needed until the agent has visited every state at least once.

As can be seen, the orange line lies above the blue line in most times, indicating that Q-learning with small epsilon generally spends more steps to fully explore the grid world.

It is also noteworthy that there are obvious spikes on the orange line, indicating that once a Q-learning agent with small epsilon "overlooks" a state, it usually takes a long time for the agent to finally reach that state. A state can be easily overlooked if it lies in a corner or in the end of a narrow path, where it can be reached from only one single neighbor. If, unfortunately, the current policy of its single neighbor is heading away from it, the probability of transiting from its neighbor to itself is very small.



The picture on the right shows an example of this situation. It can be seen that there are two unexplored states on the right side of the goal. They are "kind of" located in a corner, and the policy of their right neighbors are "heading away from them". Thus, these two states are less likely to be visited and the update of their Q values are extremely low. In addition to the fact the optimal policy of these two states will not be accurate, it is even harder for the algorithm to find the accurate utility of these two states, because the utilities need to be updated multiple times before convergence.

In a word, if we hope to find a policy for every single state, moderate exploration is essential for convergence. On the other hand, in realistic situations where the state space is too large (large Q-tables cannot be hold in memory), we should reduce the proportion of exploration, primarily aiming at learning about small proportion of the state space and seek the goal efficiently.

Reference

1. https://www.geeksforgeeks.org/what-is-reinforcement-learning/
2. https://www.geeksforgeeks.org/markov-decision-process/
3. https://www.cs.rice.edu/~vardi/dag01/givan1.pdf
4. https://github.com/iRapha/CS4641
5. https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa
6. https://medium.com/@jonathan_hui/rl-reinforcement-learning-algorithms-quick-overview-6bf69736694d
7. https://towardsdatascience.com/reinforcement-learning-demystified-exploration-vs-exploitation-in-multi-armed-bandit-setting-be950d2ee9f6
8. https://towardsdatascience.com/reinforcement-learning-from-grid-world-to-self-driving-cars-52bd3e647bc4