Markov Decision Processes

Yichao Zhang
yzhang3414
CS7641 - Machine Learning

Apr 12, 2020

Code Repository:
`https://github.com/yichigo/Markov-Decision-Processes`

0. Introduction

In this report, we applied value iteration, policy iteration, and Q-Learning on two interesting Markov Decision Processes(MDPs): frozen lake problem and forest management problem.

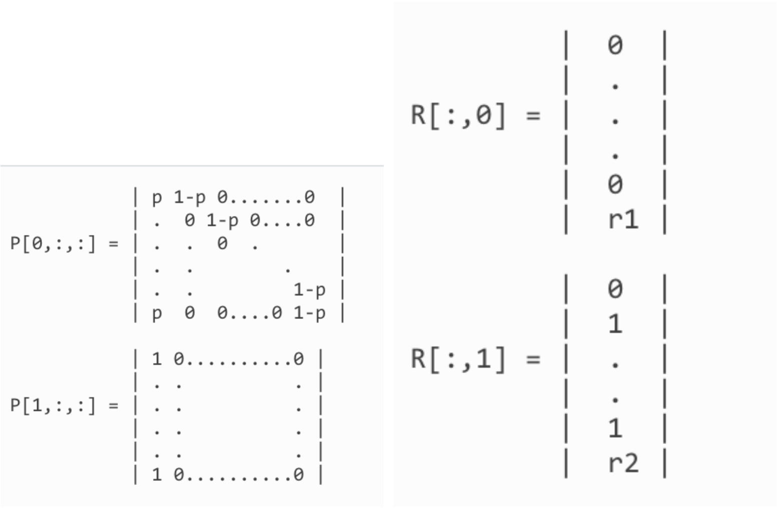
Description of 2 MDP Problems

Frozen Lake Problem has been described in `https://gym.openai.com/envs/FrozenLake-v0/`:
Winter is here. You and your friends were tossing around a frisbee at the park when you made a wild throw that left the frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you'll fall into the freezing water. At this time, there's an international frisbee shortage, so it's absolutely imperative that you navigate across the lake and retrieve the disc. However, the ice is slippery, so you won't always move in the direction you intend.
The surface is described using a grid like the following:
$SFFF \quad (S : startingpoint, safe)$
$FHFH \quad (F : frozensurface, safe)$
$FFFH \quad (H : hole, falltoyourdoom)$
$HFFG \quad (G : goal, wherethefrisbeeislocated)$
The episode ends when you reach the goal or fall in a hole. You receive a reward of 1 if you reach the goal, and zero otherwise. It is an interesting

problem because the solution of this problem can be applied in many area like traffic navigation of computer game AI.

Forest Management Problem has been described in `https://pymdptoolbox.readthedocs.io/en/latest/api/example.html`:

A forest is managed by two actions: ‘Wait’ and ‘Cut’. An action is decided each year with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. Each year there is a probability p that a fire burns the forest. The transition matrix P of the problem and the reward matrix R can be defined as follows (Figure.1):



```
              | p 1-p 0.......0  |
              | .  0 1-p 0....0  |
P[0,:,:] =    | .  .  0  .       |
              | .  .        .    |
              | .  .         1-p |
              | p  0  0....0 1-p |

              | 1 0..........0 |
              | . .          . |
P[1,:,:] =    | . .          . |
              | . .          . |
              | . .          . |
              | 1 0..........0 |
```

```
           |  0  |
           |  .  |
R[:,0] =   |  .  |
           |  .  |
           |  0  |
           |  r1 |

           |  0  |
           |  1  |
R[:,1] =   |  .  |
           |  .  |
           |  1  |
           |  r2 |
```

Figure 1: Transition matrix(left) and reward matrix(right) of forest management problem

It is interesting because the it let us make the decision based on the balance of benefit and risk. The solution might be used in some area like wealth management.

Description of 3 Methods

Value iteration builds a value table to show the score of every states. In each iteration, we update the score of each state from the discount reward of the next states and the reward of action to go to those states (times the transition rate). After that, the optimized policy can be generated from the value table by choosing the nearest neighbor with the highest score.

Policy iteration finds the best policy in each iteration from the value table of the last iteration. Then it update the value table of this iteration from the updated policy.

Q-Learning is a reinforcement learning method. It builds a Q table to evaluate the score of each action on each states. Q learner does not know the whole model of the world, it just plays the game many times and update the Q table from the rewards. After that, we can get the optimized policy from Q table.

1. Frozen Lake Problem

We generate a 6x6 frozen lake as shown below:

S	F	F	F	F	F
H	H	F	F	F	H
F	F	F	H	F	F
F	F	F	F	F	F
F	H	F	F	F	F
F	F	F	F	F	G

The number of the states is 6x6 = 36, and there are 4 actions:
left:0, down:1, right:2, and up:3 We set the discount factor 0.95 to multiply the reward each step later.

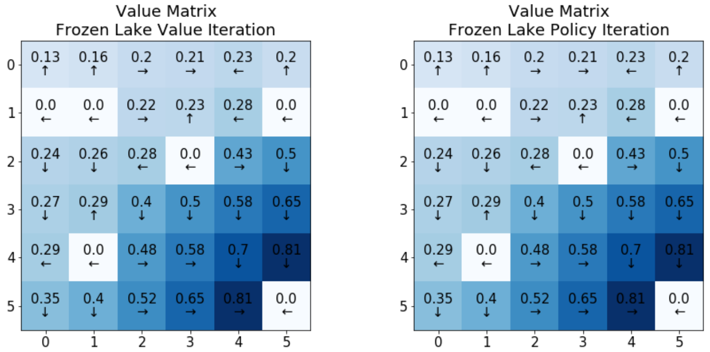
Value Iteration and Policy Iteration

Value iteration converged at iteration 393 and the running time is 1.34 seconds.

Policy iteration converged at iteration 8 and the running time is 0.28 seconds.

Policy iteration used less number of iterations and ran faster than value iteration.

Both of them returns the same optimized value function and the same optimized policy, as shown in Figure.2



Value Matrix
Frozen Lake Value Iteration

	0	1	2	3	4	5
0	0.13 ↑	0.16 ↑	0.2 →	0.21 →	0.23 ←	0.2 ↑
1	0.0 ←	0.0 ←	0.22 →	0.23 ↑	0.28 ←	0.0 ←
2	0.24 ↓	0.26 ↓	0.28 ←	0.0 ←	0.43 →	0.5 ↓
3	0.27 ↓	0.29 ↑	0.4 ↓	0.5 ↓	0.58 ↓	0.65 ↓
4	0.29 ←	0.0 ←	0.48 →	0.58 →	0.7 ↓	0.81 ↓
5	0.35 ↓	0.4 ↓	0.52 →	0.65 →	0.81 →	0.0 ←

Value Matrix
Frozen Lake Policy Iteration

	0	1	2	3	4	5
0	0.13 ↑	0.16 ↑	0.2 →	0.21 →	0.23 ←	0.2 ↑
1	0.0 ←	0.0 ←	0.22 →	0.23 ↑	0.28 ←	0.0 ←
2	0.24 ↓	0.26 ↓	0.28 ←	0.0 ←	0.43 →	0.5 ↓
3	0.27 ↓	0.29 ↑	0.4 ↓	0.5 ↓	0.58 ↓	0.65 ↓
4	0.29 ←	0.0 ←	0.48 →	0.58 →	0.7 ↓	0.81 ↓
5	0.35 ↓	0.4 ↓	0.52 →	0.65 →	0.81 →	0.0 ←

Figure 2: Optimized result: Value Iteration(left) and Policy Iteration(right). We can find that the scores on the holes are all zero, and the optimized policy go towards the targets and avoid the holes.

Q-Learning

The Q-Learner is slower than the previous 2 methods, it spend about 1.25 seconds for 2000 episodes. From the Figure.3 we can find that Q-Learner did not converge, but the rewards are similar after about 1600 episodes. The rolling average of the final rewards is around 0.5, and the rolling average of number of iterations of each episode is about 70.

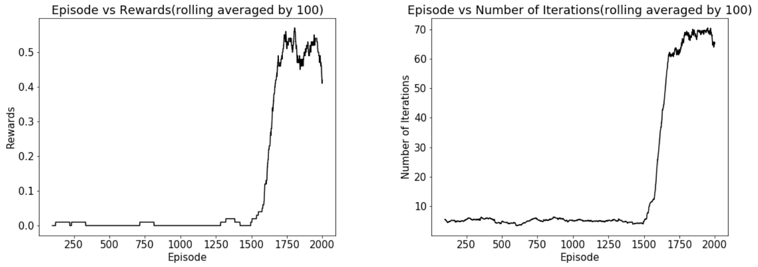


Figure 3: Q-Learning rewards and number of iterations by episodes.

We take the Q table with the highest reward (=1) as our optimized Q table. It contains the probabilities of each action for each state. By taking the maximum Q value action for each state, we got the optimized policy, as shown in Figure.4

The policy from Q learning is not the same with value iteration or policy iteration. The reason is that Q learner randomly played the game for a lot

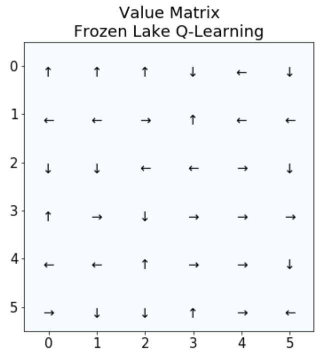


Figure 4: Optimized policy: Q-learning

of episodes. It may not have the whole picture of the grid world, but it optimizes the policy from its experiencing. In general, the optimized policy of Q Learning still leads the finder to the target on the right bottom corner and try to avoid the holes.

2. Forest Management Problem

We set the number of states is 1000.(So that we will manage the forest for 1000 years! Just kidding, but imaging that we consider the problem day by day, rather than year by year).
We set the probability of fire each year about 0.1, so that the probability of fire each day is about $1 - 0.9^{1/365} = 0.0002886$. The reward of cutting the forest is 10 and the reward of keep the old forest is 50.

Value Iteration and Policy Iteration

Value iteration converged at iteration 188 and the running time is 0.099 seconds.

Policy iteration converged at iteration 58 and the running time is 1.065 seconds.

Policy iteration used less number of iterations, but value iteration ran faster than policy iteration.

Both of them returns the same optimized value function and the same optimized policy. It suggest to cut the forest when it is young, but to keep the forest if it is already old enough.

Q-Learning

The Q-Learner is slower than the previous 2 methods, it spend about 55.65 seconds for 1000000 iterations. And it did not converge yet.
The Q learning returns the different policy, and worse policy. The reward of Q learning is only 7.68, which is much lower than value iteration(198.95) and policy iteration(198.96)