Georgia Institute of Technology
CS 7641: Machine Learning

Assignment 4 – Markov Decision Processes

Implement two MDPs, solve them using value iteration as well as policy iteration and use reinforcement learning algorithm (Q learning) to solve them

Name: Tianyu Yang
GTid: 903645962
Date: 2020/11/20

1. Introduction

In this project, I will work on two interesting Markov Decision Processes problems. One is the grid world problem and the other is non grid world problem. I will solve these two problems using value iteration and policy iteration. I will analyze these two methods from various perspectives. Lastly, I will use the reinforcement learning algorithms which is Q learning to solve these two problems. I will make a comparison for these two models and draw the conclusion.

In this report, I will firstly introduce the requirements of this project in the introduction section. I will introduce Value iteration, policy iteration and Q Learning in the Methodology part. In the result part, I will display some figures and outputs of the solution of two MDP problems. Lastly, I will analyze the results and answer the questions of this project.

1.1 Frozen Lake

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 states are in two maps, one is 4x4 and the other is 8x8.

This is an interesting grid world problem because this problem set contains two state version. One contains 16 states and one contains 64 states so that I can make a comparison of these two problems.

1.2 Taxi

There are four designated locations in the grid world indicated by R(ed), G(reen), Y(ellow), and B(lue). When the episode starts, the taxi starts off at a random square and the passenger is at a random location. The taxi drives to the passenger's location, picks up the passenger, drives to the passenger's destination (another one of the four specified locations), and then drops off the passenger. Once the passenger is dropped off, the episode ends.

Especially, there are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is in the taxi), and 4 destination locations. To compare with the frozen lake problem above, the states of this problem is much larger than that one. Therefore, this taxi problem is another interesting MDP problem set I used in this project.

2. Methodology

2.1 Value Iteration

Value iteration is a method of calculating the best MDP strategy and its value. The value iteration starts at the "end" and proceeds backwards to complete the estimate of Q* or V*. There

really is no end point, so it uses any end point. Suppose there are k stages to be carried out, let V_k be a value function; suppose there are k stages, let Q_k be a Q function. These can be defined recursively. The value iteration starts from an arbitrary function V0 and uses the following equation to obtain a function of k + 1 stages from a function of k stages:

$$Q_{k+1}(s,a) = \Sigma_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma V_k(s')) \quad \text{for } k \geq 0$$

$$V_k(s) = \max_a Q_k(s,a) \quad \text{for } k>0.$$

Here is the pseudocode of value iteration algorithms.

Procedure Value_Iteration(S,A,P,R,θ)
  Inputs
    S is the set of all states
    A is the set of all actions
    P is state transition function specifying $P(s'|s,a)$
    R is a reward function $R(s,a,s')$
    θ a threshold, $\theta>0$
  Output
    $\pi[S]$ approximately optimal policy
    $V[S]$ value function
  Local
    real array $V_k[S]$ is a sequence of value functions
    action array $\pi[S]$
  assign $V_0[S]$ arbitrarily
  $k \leftarrow 0$
  repeat
    $k \leftarrow k+1$
    for each state s **do**
      $V_k[s] = \max_a \Sigma_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma V_{k-1}[s'])$
  until $\forall s\ |V_k[s]-V_{k-1}[s]| < \theta$
  for each state s **do**
    $\pi[s] = \text{argmax}_a \Sigma_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma V_k[s'])$
  return π, V_k

2.2 Policy Iteration

Policy iteration starts with a policy and iteratively improves it. It starts with an arbitrary policy π0 (an approximation to the optimal policy works best) and carries out the following steps starting from i=0.

Policy evaluation: determine Vπi(S). The definition of Vπ is a set of |S| linear equations in |S| unknowns. The unknowns are the values of Vπi(S). There is an equation for each state. These equations can be solved by a linear equation solution method (such as Gaussian elimination) or they can be solved iteratively.

Policy improvement: choose πi+1(s)= argmaxa Qπi(s,a), where the Q-value can be obtained from V using Equation (9.5.1). To detect when the algorithm has converged, it should only change the policy if the new action for some state improves the expected value; that is, it should set πi+1(s) to be πi(s) if πi(s) is one of the actions that maximizes Qπi(s,a).

Stop if there is no change in the policy - that is, if πi+1=πi - otherwise increment i and repeat.

Here is the pseudocode of policy iteration algorithms.

Procedure Policy_Iteration(*S,A,P,R*)
Inputs
S is the set of all states
A is the set of all actions
P is state transition function specifying *P(s'|s,a)*
R is a reward function *R(s,a,s')*
Output
optimal policy π
Local
action array $\pi[S]$
Boolean variable *noChange*
real array *V[S]*
set π arbitrarily
repeat
noChange ←*true*
Solve $V[s] = \Sigma_{s' \in S} P(s'|s,\pi[s])(R(s,a,s')+\gamma V[s'])$
for each $s \in S$ **do**
Let *QBest*=*V[s]*
for each $a \in A$ **do**
Let $Qsa=\Sigma_{s' \in S} P(s'|s,a)(R(s,a,s')+\gamma V[s'])$
if (*Qsa* > *QBest*) **then**
$\pi[s] \leftarrow a$
QBest ←*Qsa*
noChange ←*false*
until *noChange*
return π

2.3 Q learning

Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

Here is the pseudocode of Q learning algorithms.

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Choose a from s using policy derived from Q
         (e.g., ε-greedy)
   Repeat (for each step of episode):
      Take action a, observe r, s'
      Choose a' from s' using policy derived from Q
         (e.g., ε-greedy)
      Q(s, a) <-- Q(s, a) + α[r + γQ(s', a')- Q(s, a)]
      s <-- s'; a <-- a';
   until s is terminal
```

3. Result

Here are the iterations, running time and the scores for these two models. In the program, I count the iteration and processing time. Besides, I also count the average scores for each of the solution. In this specific project, I use 0.99 as the gramma of the Q learning to solve the problem.

Frozen Lake 4x4:

The frozen lake 4x4 problem has 16 states and 4 actions. Here is the table for value iteration,

policy iteration and Q learning.

Frozen Lake 4x4	Iterations	Processing Time	Scores
Value Iteration	806	0.18707549999999173	0.88
Policy Iteration	4	0.09995000000003529	0.85
Q learning	500000	356.81261530000006	0.76

Frozen Lake 8x8:

The frozen lake 8x8 problem has 64 states and 4 actions. Here is the table for value iteration, policy iteration and Q learning.

Frozen Lake 4x4	Iterations	Processing Time	Scores
Value Iteration	1425	1.2107356999999865	1.0
Policy Iteration	12	2.667106800000056	1.0
Q learning	500000	538.6307535000001	0.15

Taxi:

The taxi problem has 500 states and 6 actions. Here is the table for value iteration, policy iteration and Q learning.

Frozen Lake 4x4	Iterations	Processing Time	Scores
Value Iteration	19	0.09314059999996971	7.8
Policy Iteration	17	28.794077600000037	8.28
Q learning	500000	790.9325297	7.58

4. Conclusion

4.1 Comparison of each problem and algorithms

As a result, with the comparison of value iteration and policy iteration, I found that value iterations need more iterations than policy iterations. But as the states increase, the difference between these two algorithms decreases. When the size of state is small, the processing time is almost close to each other. However, when the state is very much, the value iteration has less processing time than policy iteration. The score of these two algorithms is close. But if the states are large, we had better choose policy iteration because the scores are better than value iteration.

What's more, when using 50000 iterations of Q learning reinforcement algorithm to these questions, we can find that the processing time is extremely increasing. And the scores are worse than value iteration and policy iteration algorithm.

4.2 Analysis

In the Q learning function, I use 0.99 as the value of gramma because when I tried to use the gramma value that is larger than 1, The Q learning function takes too much time to solve the problem. Therefore, I use the fixed gramma value in this project to save executing time. What's more, in Q Learning function, the stopping condition of Q learning, which is similar to the optimal in policy iterations, is to get higher scores in the problem. To change the parameter of this stopping condition can help to improve the performance of Q learning. I am testing more to get the best iterations in Q learning to get a decent policy because the parameters I set needs more test to get the best results in

each of the problem.

Look at the result for the project, I am so excited to finish the tasks in value iteration, policy iteration and Q learning reinforcement learning algorithm. From the course, I learned that the value iteration should be much quicker in time than policy iteration. I am glad that I get the correct result in my solution to the MDP problems. The iterations for the main loops of policy iteration is much lower than the value iteration. This might be because for each steps of the policy iteration, the policy needs to be evaluated in each of the steps. This will waste much time. Therefore, as a result, the time complexity of policy iteration is low, but the iterative complexity is good

Besides, from the course materials, I learned that Q learning can converge to the optimal policy when comparing to the other algorithms. This is because it can visit each of the states in infinite number of times. From the result above, comparing to different count of states number, we know that when the count increases, both these three algorithms converge. The policy iteration converge when the count is small, the value iteration is more than policy iteration and the Q learning is the most.

4.3 Summary

In this project, I choose two interesting MDP problems, the frozen lake and the taxi. Especially, I set two different state sizes for the frozen lake and use another large state problem as the taxi. I found the number of iterations they use for value iteration, policy iteration and the reinforcement algorithm, Q learning. After running the program and solve the problem, I get the result and compare with the iterations, running time and scores for each of the algorithms. After comparing the result and knowledge I learned from the course. I give my analyze and conclude the features of these three algorithms. So that in the future, when I face the real-world questions, I can know which algorithms I need to use to solve the MDP problem.

What's more, by using the function of value iteration, policy iteration and Q learning, I know how many iterations each of the algorithms converges and I can make a comparison of each of them. I use three dimensions to compare with them, including executing time, the count of iterations and average scores of the solution. The problems I choose is interesting because actually I use three different count of the states problems to make the comparison. What's more, I also make the comparison of large states number and small states number of grid world problems. These experiences are really helpful to strengthen my understanding of Markov Decision Processing.