CS181 Artificial Intelligence Final Porject: Reinforcement Learning in Partially Observable Markov Decision Process

Guanqi He, Xiaoyu Lin, Ziyun Zeng, Chengyi Xu, Yichen Lu
ShanghaiTech University, China

Contents

1. Abstract 1

2. Introduction 1
2.1. POMDP 1
2.2. Q-learning 2

3. Approximate POMDP 2
3.1. Methodology 2
3.2. Pseudo Code 3
3.3. External Resources 3

4. Partially Model-based Method 3
4.1. MLE Q-learning 3
4.2. Expectation Q-learning 3

5. K-Memory Q-learning 4

6. Experiment 4
6.1. Grid World 4
6.1.1 Approximate POMDP Result . . . 4
6.2. Pacman 5
6.2.1 Baseline 5
6.2.2 MLE Q-learning, Expectation Q-Learning and k-Memory Q-Learning Results 5
6.3. Comparison 6

7. Conclusion 6

8. External Resources 6

1. Abstract

In this project, we developed many kinds of reinforcement learning algorithms for POMDP model. They are model-based approximate POMDP algorithm, partially model-based MLE Q-learning algorithm, expectation Q-learning algorithm, and model-free k-memory Q-learning algorithm. We also designed a variety of experiments to evaluate our algorithm.

2. Introduction

The description of Markov decision processes assume that the environment is fully observable. With this assumption, the agent always knows which state it is in. This, combined with the Markov assumption for the transition model, means that the optimal policy depends only on the current state. When the environment is only partially observable,the situation is, one might say, much less clear. The agent does not necessarily know which state it is in, so it cannot execute the action (s) recommended for that state.

Furthermore, the utility of a state s and the optimal action in s depend not just on s, but also on how much the agent knows when it is in s. For these reasons, POMDPs are usually viewed as much more difficult than ordinary MDPs. However, we cannot avoid POMDPs, because the real world and a lot of real problems in the world are all POMDPs. Therefore, learning how to solve a POMDP problem is full of practical significance. [1]

2.1. POMDP

A POMDP has the same elements as an MDP the transition model P(s|s,a), actions A(s), and reward function R(s). As a partially observed model, it may also have a sensor model P(e|s) which specifies the probability of perceiving evidence e in state s.Furthermore, we should calculate the distribution of states that the agent considers possible:

$$b'(s') = \alpha P(e|s') \sum_s P(s'|s,a)b(s)$$

where α is a normalizing constant that makes the belief state sum to 1. The probability of perceiving e, given that a was performed starting in belief state b, is given by summing over all the actual states s that the agent might reach:

$$\begin{aligned} P(e|b) &= \sum_{s'} P(e|a,s',b)P(s'|a,b) \\ &= \sum_{s'} P(e|s')P(s'|a,b) \\ &= \sum_{s'} P(e|s') \sum_s P(s'|s,a)b(s) \end{aligned}$$

Let us write the probability of reaching b from b, given action a, as $P(b|b,a)$. Then that gives us

$$P(b'|b,a) = \sum_e P(b'|e,a,b)P(e|a,b)$$
$$= \sum_e P(b'|e,a,b) \sum_{s'} P(e|s') \sum_s P(s'|s,a)b(s)$$

2.2. Q-learning

Q-learning is an alternative TD method, which learns an action-utility representation instead of learning utilities. We will use the notation $Q(s,a)$ to denote the value of doing action a in state s. Q-values are directly related to utility values as follows:

$$U(s) = \max_a Q(s,a).$$

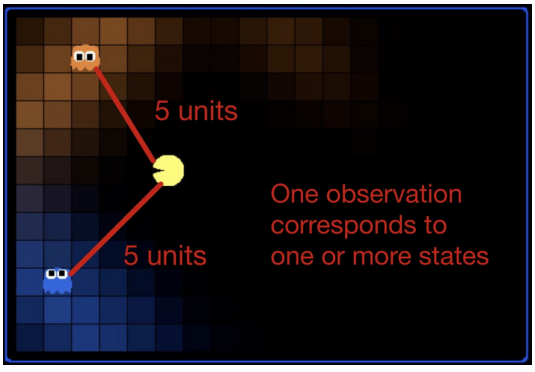
Q-functions may seem like to be another way of storing utility information, but they have a very important property: a TD agent that learns a Q-function does not need a model of the form $P(s' \mid s,a)$, either for learning or for action selection. For this reason, Q-learning is called a model-free method. As with utilities, we can write a constraint equation that must hold at equilibrium when the Q-values are correct:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a').$$

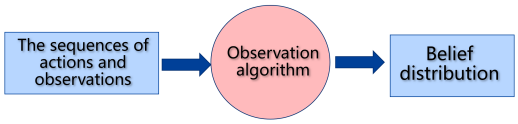
As in the ADP learning agent, we can use this equation directly as an update equation for an iteration process that calculates exact Q-values, given an estimated model. This does, however, require that a model also be learned, because the equation uses $P(s' \mid s,a)$. The temporal-difference approach, on the other hand, requires no model of state transitions-all it needs are the Q values. The update equation for TD Q-learning is

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

which is calculated whenever action a is executed in state s leading to state s'.



The difficulty of solving a POMDP problem is that we can't observe the current state of the process, and what we only have is the sequences of actions and observations, so what we should do is to create an observation algorithm which can change the sequences of actions and observations into a belief distribution.



3. Approximate POMDP

3.1. Methodology

By constructing beliefs about the distribution of possible states, we transform the problem into an MDP problem in a agent's belief space. However, beliefs are continuous, policy iteration in class cannot work. Value iteration is also difficult to use because of computational complexity (PSPACE-hard, time complexity $O(|A|^{|E|^d})$, d is depth). We chose to use an online agent, which is based on a dynamic decision network and gives an approximate solution by derivation of the next few steps, whose time complexity is $O(|A|^d * |E|^d)$.

In this chapter, we first introduce updates to random variables (beliefs, observation distributions, etc.), and then introduce our online agent.

1. updates to random variables

Belief update:

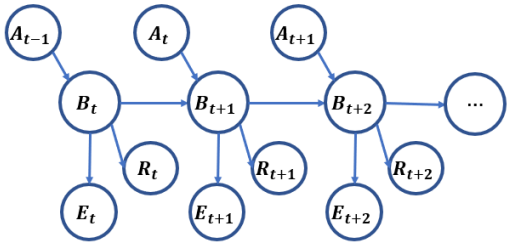
$$b'(s') = \alpha P(e|s') \sum_s P(s'|s,a)b(s)$$

We call this function as $b' = forward(b,a,e)$.

Get distribution of evidence.

$$P(e|a,b) = \sum_{s'} P(e|s') \sum_s P(s'|s,a)b(s)$$

2. Dynamic Decision Network



We assume that the depth of the algorithm is d. At time t, we will consider all possible situations at time t+1, t+2,..., t+d, and choose an action that maximizes the benefit. If

t+d is not a terminal state, the agent evaluates that state. In this problem, since the agent knows all the rules of the environment, each state is evaluated by

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

. This can be calculated by value iteration.

How do we evaluate the action by considering next d steps? Here we use a game algorithm similar to that in PA1B: Expectation-Maximum Tree. At the behavior level, we select the behavior with the highest evaluation benefit. At the observation layer, we calculate the probability of obtaining each observation, and calculate the expectation as an evaluation value. The expectation is calculated as

$$\rho(b) = \sum_{s} b(s)R(s)$$

. Because the rules of the environment are fixed, belief updates can also be determined, independent of actual outcomes.

By evaluating d steps in the future, the agent chooses a good action in the current belief space. Then we completed a POMDP.

3.2. Pseudo Code

3.3. External Resources

In this part, we write **hiddengridworld.py** based on PA5 **gridworld.py**. We used some files of PA5 as the grid world environment. Because the visualization part is too deeply coupled with the homework, it is not used here.

Algorithm 1 Expectation-Maximum

Input: belief, depth, max_depth
Output: action

if depth = max_depth **then**
  res=$\sum_s b(s)evaluate(s)$
  return γ * res
end if
max = -1e9
choose_action = None
for $a \in A$ **do**
  calculate P(E|a, b)
  expect = 0
  for $e, prob \in P(E|a,b)$ **do**
    if e in terminal state **then**
      expect += prob * γ * R(e)
    else
      next_belief = forward(belief, a, e)
      expect += prob *Expectation-Maximum(next_belief, depth + 1)
    end if
  end for
end for
if expect > max **then**
  max = expect
  choose_action = a
end if
if depth = 0 **then**
  return choose_action
else
  return γ * max
end if

4. Partially Model-based Method

4.1. MLE Q-learning

The key assumption in this method is a reasonable estimation of transition model T(s,a,s) and an emission model E(s).
Our method is to get a belief distribution from the given observation sequence and take the most likely estimation of the brief distribution to estimate a most-likely state, then implement the origin Q-learning on the estimated state.

$$s = \mathbf{MLE}(\tilde{s})$$
$$sample = R + \gamma \max_{a'} Q(s, a')$$

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot sample \cdot Q(\tilde{s}, a)$$

4.2. Expectation Q-learning

The key assumption in this part is a reasonable estimation of transition model T(s,a,s) and an emission model E(s).

Our method is to get a belief distribution from the given observation sequence first. For the fact that the Q-value for brief distribution s' does not exist, we design a novel approach to estimate current state value and update the Q-value table. First, consider that $Q(\tilde{s}, a')$ does not exist, we replace the $\max_{a'} Q(\tilde{s}, a')$ with $\mathbb{E}[\max_{a'} Q(\tilde{s}, a')]$. Then, to maximally exploit the information from the brief distribution, we redesign the update law. Intuitively, the reward should make more contribution to more-likely states, and should make less contribution to less-likely states. Thus,

we modify the update factor α as $\alpha \cdot P(s)$.

$$
\begin{aligned}
sample &= R + \gamma \mathbf{E}[\max_{a'} Q(\tilde{s}, a')] \\
&= R + \gamma \sum_{s} [P(s|e) \cdot \max_{a'} Q(s, a')] \\
\beta &= P(s) \cdot \alpha \\
Q(s,a) &\leftarrow (1-\beta) Q(s,a) + \beta \cdot sample
\end{aligned}
$$

5. K-Memory Q-learning

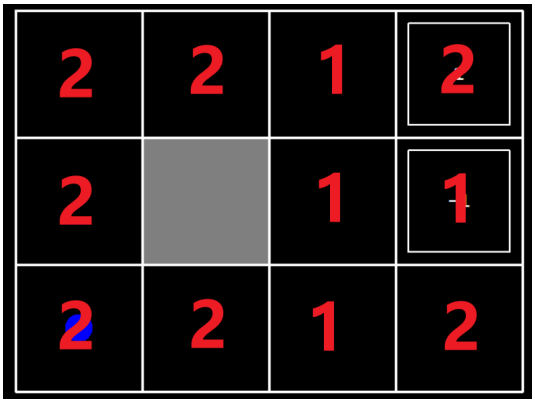
In this section, we would like to discuss a model-free reinforcement learning method. Recall the beginning, the key-point of reinforcement learning in partially observable markov decision process is to make optimal decision via observation sequence and action sequence. Thus, why not we just take the augmented past states s=$(e_k, a_k, e_{k-1}, a_{k-1}, \cdots, e_1, a_1)$ as the current state and implement the learning algorithm? However, we cannot directly implement the format algorithm for the fact that the dimension of state space will increase exponentially fast.

Thought we cannot store all the past actions and observations, it is still possible to memorize past k steps actions and observations. Inspired by [2], we define the k-step augmented state as s_a=$(e_k, a_k, e_{k-1}, a_{k-1}, \cdots, e_1, a_1)$ then implement Q-learning on the augmented state space. The algorithm will converge if the model is k-stop Markov. However, in the experiments, we observe that even though the parameter k is small than the real Markov order, the algorithm could still present a fancy performance.

6. Experiment

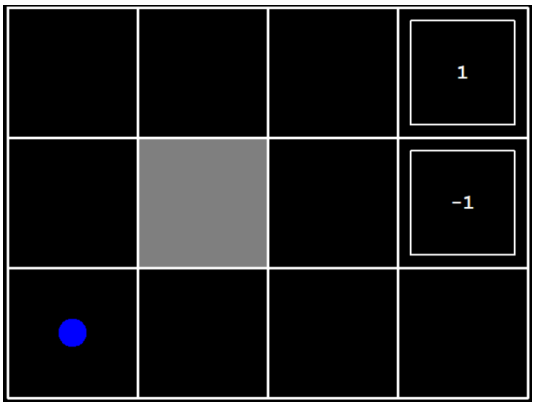
6.1. Grid World

Consider the grid world of PA5, we would like to change it to a partially observed environment. The idea of modification is this: the agent cannot directly know his position, but observes several walls within distance 1 through the sensor. The sensor give the correct observation in 70% solutions, otherwise it returns {0,1,2} with the same probability.



For example, in the above grid world, if the sensor is giving correct observation, then its observation should be the same as the red font in the picture. In addition, the environment has noise, which means the agent's behavior has only $1 - noise$ probability of obtaining the expected result. The agent knows all the rules of the environment (that is, the possible situations of actions and observations), he just cannot get all the observations. The agent wants to get the highest expected score, which is our approximate POMDP's target.

6.1.1 Approximate POMDP Result



We test the agent under different noise and discount rates and depths in a 4*3 grid world. We repeated 100 trials each time and calculated the average number of moving steps (length) and score (score), and the results were in table1.

We see that as depth goes up, both length and score go up.

When $noise = 0$, the agent quickly finds its position and successfully reaches the destination. In particular, $discount = 0.7, depth = 3$, the agent always chooses a path of length 7 $['west','north','north','east','east','east','exit']$, This is the optimal route, the first step is to go west because the agent does not know its position, any other actions may result in a -1 position.

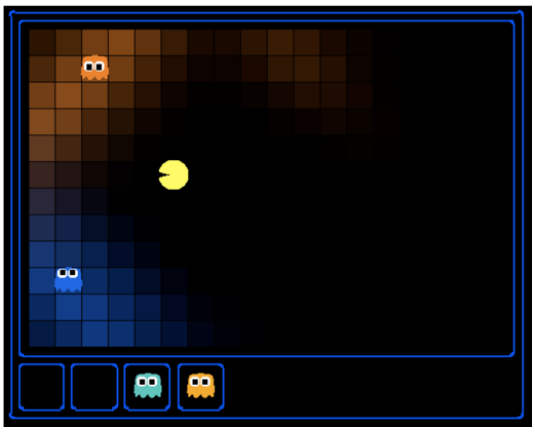
Repeat	Discount	Noise	Depth	Length	Score
100	0.9	0	1	10.53	100
100	0.9	0	2	7.58	100
100	0.9	0	3	7.14	100
100	0.7	0	1	8.9	100
100	0.7	0	2	7.65	100
100	0.7	0	3	7	100
100	0.9	0.2	1	13.23	82
100	0.9	0.2	2	12.13	84
100	0.9	0.2	3	10.4	92
100	0.7	0.2	1	12.08	50
100	0.7	0.2	2	11.15	74
100	0.7	0.2	3	10.67	74

Table 1. Result for Approximate POMDP

When $noise = 0.2$, agents at high discount rates will choose more aggressive strategies, resulting in shorter average routes but lower scores. Because of the noise, the agent cannot determine its position, which results in a higher average length than without noise.

6.2. Pacman

In the pacman world, pacman agents will use sensors to locate and eat invisible ghosts. However, due to the partially observable ghost location which means that we can only know the distribution of ghost's location.



6.2.1 Baseline

In this experiment, we implement an random decision-making agent as the baseline. The agent will randomly choose an action a_i from the legal action set $\mathcal{A}$ with even probability. Averagely, the baseline will get about 80 points in the experiment setting, we take it as the baseline score (0.0).

6.2.2 MLE Q-learning, Expectation Q-Learning and k-Memory Q-Learning Results

Definition of algorithm parameters are shown below. Repeat means how many times we implement the algorithm and get the average score. Alpha means the learning rate. Discount means sooner rewards probably do have higher utility than later rewards. The hyperparameter K decides the order number of the Markov Decision Process Model we assumed.

Experiment results are shown in tables below.

Repeat	Alpha	Discount	Score
100	0.05	0.7	47.93
100	0.05	0.8	53.74
100	0.05	0.9	62.68
100	0.1	0.7	52.58
100	0.1	0.8	57.84
100	0.1	0.9	62.94
100	0.2	0.7	57.47
100	0.2	0.8	62.17
100	0.2	0.9	69.29

Table 2. Result for MLE Q-learning with 100 epoch training

Repeat	Alpha	Discount	Score
100	0.05	0.7	62.25
100	0.05	0.8	68.06
100	0.05	0.9	77.01
100	0.1	0.7	66.91
100	0.1	0.8	72.16
100	0.1	0.9	77.26
100	0.2	0.7	71.79
100	0.2	0.8	75.64
100	0.2	0.9	83.61

Table 3. Result for Expectation Q-learning with 100 epoch training

Repeat	Alpha	Discount	Score
100	0.05	0.7	64.19
100	0.05	0.8	70.01
100	0.05	0.9	78.95
100	0.1	0.7	68.85
100	0.1	0.8	74.11
100	0.1	0.9	79.21
100	0.2	0.7	73.74
100	0.2	0.8	77.59
100	0.2	0.9	85.56

Table 4. Result for K-Memory Q-learning with 100 epoch training and K=3

Repeat	Alpha	Discount	Score
100	0.05	0.7	34.19
100	0.05	0.8	40.01
100	0.05	0.9	48.95
100	0.1	0.7	38.85
100	0.1	0.8	44.11
100	0.1	0.9	49.21
100	0.2	0.7	43.74
100	0.2	0.8	47.59
100	0.2	0.9	48.56

Table 5. Result for K-Memory Q-learning with 100 epoch training and K=2

6.3. Comparison

- Expectation Q-learning performs better than MLE Q-learning due to MLE Q-learning is just like a kind of gambling so it may fail in some edge situations, however, Expectation Q-learning prefers to choosing a state by making sure that it and it's nearby all have a big probability
- Although Expectation Q-learning has a better performance compared with MLE Q-learning, it has a very high time complexity.
- In K-Memory Q-learning algorithm, the score will rise a lot when enlarging the hyperparameter K even if the time complexity rises and the state space becomes larger.

Algorithm	Score(100 games average)
Random(baseline)	0.00
2-Memory Q-learning	47.59
MLE Q-learning	62.17
Expectation Q-learning	75.64
3-Memory Q-learning	77.59

Table 6. Comparison of several algorithms

7. Conclusion

In this project, we developed a variety of reinforcement learning algorithms for POMDP model, including model-based approximate POMDP algorithm, partially model-based MLE Q-learning algorithm, expectation Q-learning algorithm, and model-free k-memory Q-learning algorithm. We also designed a variety of experiments to evaluate our proposed algorithm in detail. Through this project, we gain a deeper understanding of the reinforcement learning algorithm and search algorithm learned in the course, and also exercise our programming ability and expression ability. Finally, due to time limitation, we do not combine the algorithm with deep learning. We will continue to try to combine relevant algorithms with deep learning to deal with state-space dimension exploration and improve the performance of the algorithm.

8. External Resources

CS181 Programming Assignments

References

[1] Russell and J. Stuart. Artificial intelligence: A modern approach. *People's Posts and Telecommunications Publishing House*, 2002.

[2] Lei Zheng, Siu-Yeung Cho, and Chai Quek. A memory-based reinforcement learning algorithm for partially observable markovian decision processes. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 800–805, 2008.