IE 685: MSc-PhD Research Project 1

Q-Learning with constraint ?

Anuj Sharma | 20i190010

Supervisor: Prof. V.Kavitha

Industrial Engineering and Operations Research
Indian Institute of Technology, Bombay

Autumn, 2021-2022

Contents

1 Introduction **4**

1.1 Markov Decision Process 4

1.2 Q-Learning 5

2 Basic definition **6**

2.1 Decision Epochs: 6

2.2 State And Action Sets : 7

2.3 Transition Probabilities : 7

2.4 Rewards : 7

2.5 Decision Rules: 7

2.6 Policy: 8

3 Value functions **9**

3.1 State value function: 9

3.2 State Action value function: 10

4 Q-Learning algorithm **11**

4.1 Epsilon Greedy algorithm: 12

4.1.1 The Algorithm is: 13

5 Experiments and Results **14**

5.1 Experiment: 14

5.2 Environment: 15

5.2.1 States: 15

5.2.2 Action: 15

5.2.3 Rewards: 16

5.2.4 Probabilities: 16
5.3 Value iteration algorithm: 16
5.4 Epsilon Greedy Algorithm: 17
5.4.1 Q-Learning with Constraint ? 18
5.5 Result: 20
5.5.1 when constraint are not added. 20
5.5.2 when constraint are added. 21

6 Conclusion and Future Work **23**

Chapter 1

Introduction

1.1 Markov Decision Process

A Markov decision Process (MDP) is a powerful tool in planning tasks and sequential decision making problems . At a specified point of time, a decision maker or agent observes the state of a system. Based on this state, the decision maker chooses an action. The action choice produces two results: the decision maker receives an immediate reward , and the system evolves to a new state at a subsequent point in time according to a probability distribution determined by the action choice. [1] The Markov Decision Process (MDP) model contains -

- Decision epochs
- A set of possible States S
- A set of possible Actions A
- The transition probabilities
- The Rewards

We refer to the collection of objects

$$\{\text{T,S},A_s,P_t(.|s,a),r_t(s,a)\}$$

as a Markov decision process.

1.2 Q-Learning

Q-Learning is a Reinforcement learning policy from which the agent learn from his previous decisions and the agent are able to find out the next best action in the current state .

What's 'Q' in Q-learning ?
The 'Q' in Q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

Q-learning is a model-free , value-based , off- policy learning algorithm .

- Model-free: The algorithm that estimates its optimal policy without the need for any transition or reward functions.
- Value-based : Q learning updates its value functions based on equations, rather than estimating the value function with a policy .
- Off-policy : The function learns from its own actions and does not depend on the current policy.

Goal of Q-Learning : With the help of Q-learning the agent learn from his previous decisions and will try to get the best action so that he can maximize the total reward.

Chapter 2

Basic definition

2.1 Decision Epochs:

In Markov decision process , The time where the decision maker required to make a decision , that points of time are said to be Decision Epochs . [1]

Decision Epochs are classified in two ways :

1. The set of Decision Epochs is either finite or infinite .
   i.e
   T = {1,2,3....,N } where N $\leq \infty$
2. The set of Decision Epochs is either Discrete or Continuous.

When N is finite, the decision problem will be called a finite horizon problem; otherwise it will be called an infinite horizon problem. In a discrete time Markov decision process, decision epochs occur at regular, fixed intervals, whereas in a continuous time Markov decision process, they may occur at randomly distributed intervals.

2.2 State And Action Sets :

At each decision epoch, the system occupies a state. The State is a specific place at which an agent is present at current time . We denote the set of possible system states by S. At some decision epoch, the decision maker observes the system in state s ∈ S .
Then the decision maker choose action a from the Actions sets in state s ,A_s

2.3 Transition Probabilities :

When the Decision maker or agent are in state s and choose a action A_s in state s at decision epoch t , the system state at the next decision epoch is determined by the probability distribution $p_t(.|s, a)$. Let after choosing action a in state s , the decision maker go to the next state s' then transition probability

$$T(s, a, s') = P(s'|s, a)$$

2.4 Rewards :

When the Decision maker or agent are in state s and choose a action A_s in state s at decision epoch t ,the decision maker receives a reward $r_t(s, a)$ for $s \in S$ And $a \in A_s$
When positive, $r_t(s, a)$ may be regarded as income, and when negative as cost. When the reward depends on the state of the system at the next decision epoch,
we let $r_t(s, a, j)$ denote the value at time t of the reward received when the state of the system at decision epoch t is s, action $a \in A_s$, is selected, and the system occupies state j at decision epoch t + 1. Its expected value at decision epoch t may be evaluated by computing

$$r_t(s, a) = \sum_{j \in S} r_t(s, a, j) p_t(j|s, a)$$

The function$p_t(j|s, a)$ is called a transition probability function. and

$$\sum_{j \in S} p_t(j|s, a) = 1$$

2.5 Decision Rules:

A decision rule prescribes a procedure for action selection in each state at a specified decision epoch. Decision rules are functions

$$d_t : S \to A_s$$

which specify the action choice when the system occupies state s at decision epoch t. For each $s \in S$, $d_t(s) \in A_s$.
This decision rule is said to be Markovian (memoryless) because it depends on previous system states and actions only through the current state of the system, and deterministic because it chooses an action with certainty.

2.6 Policy:

A policy, specifies the decision rule to be used at all decision epoch. It provides the decision maker with a prescription for action selection under any possible future system state or history. A policy π is a sequence of decision rules,
i.e.,

$$\pi = (d_1, d_2, , d_{N-1})$$

We call a policy stationary if $d_t = d$ for all $t \in T$. A stationary policy has the form
$\pi = (d_1, d_2, , d_{N-1})$; we denote it by d^∞.

We sometimes refer to stationary deterministic policies as pure policies. Stationary policies are fundamental to the theory of infinite horizon Markov decision processes.

Thus The policy is functions mapping from states to actions.

$$\pi : S \to A$$

Chapter 3

Value functions

Utility: When the agent is in state s and choose a action a .where $s \in S$ and $a \in A_s$. Then we get a reward r and go to the next state j where $j \in S$. Then the total utility is given by the discounted sum of the rewards of a policy:

$$U = r_1 + \gamma r_2 + \gamma^2 r_3 + ...$$

$$U = \sum_{t=1}^{\infty} \gamma^{t=1} r_t$$

where $\gamma \in (0, 1)$ is the discount factor .
and here our main goal is to find out the maximum of total expected utility .
i.e.

$$\max E_\pi[\sum_{t=1}^{\infty} \gamma^{t-1} r_t]$$

3.1 State value function:

A State value is a function:

$$V : S \to R$$

It is the expected utility if the agent start from state s and taken action according to policy π.
i.e.

$$V(s) = E[U|s_t = s]$$

$$V(s) = E[r_1 + \gamma r_2 + \gamma^2 r_3 + ...|s_t = s]$$

$$V(s) = E[r_1|s_t = s] + \gamma E[r_2 + \gamma r_3 + \gamma^2 r_4 + ...|s_t = s]$$

$$V(s) = r(s, a) + \gamma \sum_{j \in S} p(j|s, a) V(j)$$

and the optimal state value function is :

$$V^*(s) = max_a[r(s, a) + \gamma \sum_{j \in S} P(j|s, a) V^*(j)]$$

3.2 State Action value function:

A State Action value is a function:

$$Q : S \times A \rightarrow R$$

It is the expected utility if the agent start from state s and taken action according to policy π ,taking action a
i.e.

$$Q(s, a) = E[U|s_t = s, a_t = a]$$

In the state value function we get the optimal value . The relationship between V and Q function is :

$$V^*(s) = max_a Q^*(s, a)$$

Then the optimal value for the state action function is :

$$Q^*(s, a) = [r(s, a) + \gamma \sum_{j \in S} P(j|s, a) max_a Q^*(j, a)]$$

Chapter 4

Q-Learning algorithm

In the Q-Learning algorithm, the goal of this algorithm is to learn the optimal Q-value function [2] -

For each state action pair (s,a) , initialize the table entry $Q^*(s, a)$ to zero.

observe the current state s.

Do forever-

Select an action a and execute it .

Receive immediate reward r.

observe the new state s'.

Update the table entry for $Q^*(s, a)$ as follows -

$$Q^*(s, a) \leftarrow r(s, a) + \gamma max_a Q^*(s', a)$$

s $\leftarrow$ s'

After a long time we get a good Q table o and optimize Q value.

4.1 Epsilon Greedy algorithm:

The epsilon-greedy approach selects the action from which we get the maximum reward most of the time. and it is a technique to balance between exploration and exploitation. Exploration allows us to try new things, sometimes contradicting what we have already learned, with a small probability of ϵ, we choose to explore, i.e., not to exploit what we have learned so far. In this case, the action is selected randomly, independent of the action - value estimates. [3]
The epsilon-greedy action selection policy discovers the optimal actions for sure.
Now, we would start implementing the Q-Learning algorithm. But, we need to talk about the exploration-exploitation trade-off.
But Why?
In the beginning, the agent has no idea about the environment. He wants to explore new things than to exploit his knowledge because. Through time steps, the agent will get more and more information about how the environment works and then, he is more likely to exploit his knowledge than exploring new things. If we skip this important step, the Q-Value function will converge to a local minimum which in most of the time, is far from the optimal Q-value function.
So to overcome this problem we try to use epsilon greedy algorithm and with the help of this Q function will converge to the global optimal Q value function .

4.1.1 The Algorithm is:

Data: α : learning rate ,γ : Discount factor , ϵ : a small number

Result: A Q table containing Q(S,A) pairs defining estimated optimal policy π^*

/* Initialization */

Initialization Q(s,a) arbitrarily ,except Q(terminal,.);

Q(terminal,.) $\leftarrow$ 0 ;

/* For each step in each episode ,we calculate the Q- value and update the Q-table */

for each episode **do**

  /* Initialize state S */

  Initialize state S;

  for each step in episode **do**

    do

      /* Choose Action A from S using epsilon greedy policy derived from Q */

      A $\leftarrow$ Select Action (Q,S,ϵ);

      Take action A ,then observe reward R and next state S' ;

      Q(S,A) Q(S.A) + α [R + γmax_a Q(S',a)-Q(S,A)];

      S$\leftarrow$ S'

    while S is not terminal;

  end

end

By using this algorithm we get the optimal Q value. and maximize the reward function .

Chapter 5

Experiments and Results

5.1 Experiment:

Problem Scenario – :

- There are two states and for each states there are two actions.
- If we choose a state and apply action on this state then we get some rewards.
- We get the next state by some probability.
- Our main aim is to get the maximum reward.

We will use Q-learning to accomplish this task!

5.2 Environment:

The Environment Consists of States, actions and rewards and probabilities.

5.2.1 States:

- Two states { S_1 , S_2}.

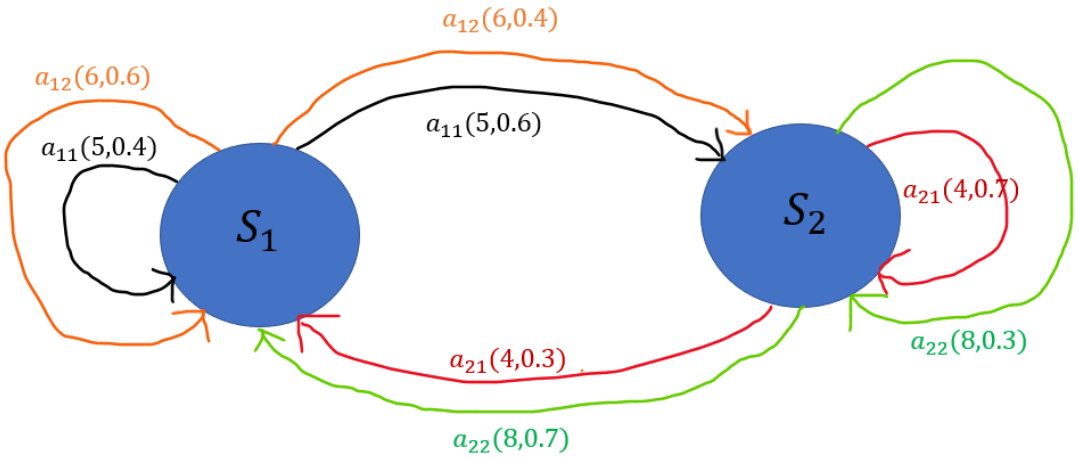

Figure 5.1

5.2.2 Action:

For each of two states there are two actions.

For state 1 there are two actions a_{11} and a_{12} .

1. $A_{S_1} = \{a_{11}, a_{12}\}$

   For state 2 there are two actions a_{21} and a_{22}

2. $A_{S_2} = \{a_{21}, a_{22}\}$

5.2.3 Rewards:

After choosing action on a state there will be some rewards on going to next state

- If we choose action a_{11} on state 1 then the reward will be 5 .
- If we choose action a_{12} on state 1 then the reward will be 6 .
- If we choose action a_{21} on state 2 then the reward will be 4 .
- If we choose action a_{22} on state 2 then the reward will be 8 .

i.e. The reward matrix is :

$$R = \begin{bmatrix} 5 & 6 \\ 4 & 8 \end{bmatrix}$$

5.2.4 Probabilities:

By some probability we get the next state after choose the action on the state .The probability matrix is :

```
# P [from state] [to state] [action]
P = np.array([
    [[0.4, 0.6], [0.6, 0.4]],
    [[0.3, 0.7], [0.7, 0.3]]
 ])
```

Figure 5.2

5.3 Value iteration algorithm:

1. Select $v^0 \in V, specify\ \delta > 0$ and t = 0 .
2. For each s ∈ S ,compute $v^{t+1}(s)$ by :
   $v^{t+1}(s) = max_{a \in A_s}\{r(s,a) + \sum_{j \in S} p(j|s,a)v^t(j)\}$
3. if $\left|v^{t+1} - v^t\right| < \delta$
   Stop. Otherwise increment t by 1 and return to step 2.

5.4 Epsilon Greedy Algorithm:

Data: α : learning rate ,γ : Discount factor , ϵ : a small number

Result: A Q table containing Q(S,A) pairs defining estimated optimal policy π^*

/* Initialization */

Initialization Q(s,a) arbitrarily ,except Q(terminal,.);

Q(terminal,.) $\leftarrow$ 0 ;

/* For each step in each episode ,we calculate the Q- value and update the Q-table */

for each episode **do**

  /* Initialize state S */

  Initialize state S;

  for each step in episode **do**

    do

      /* Choose Action A from S using epsilon greedy policy derived from Q */

      A $\leftarrow$ Select Action (Q,S,ϵ);

      Take action A ,then observe reward R and next state S' ;

      Q(S,A) $\leftarrow$ Q(S.A) + α [R + γmax_a Q(S',a)-Q(S,A)];

      S$\leftarrow$ S'

    while S is not terminal;

  end

end

By using this algorithm we get the optimal Q value. and maximize the reward function .

5.4.1 Q-Learning with Constraint ?

Now we add constraints in our problem and check if it will work or not. if it will work we try to find its maximum reward. i.e.

$max_\pi \quad E_x^\pi[\sum_{t=1}^{\infty} \lambda^{t-1} R_t]$
s.t.
$E_x^\pi[\sum_{t=1}^{\infty} \lambda^{t-1} C_t] \leq B$

where B is any constant.
and C be the cost matrix as :

$$C = \begin{bmatrix} 1 & 5 \\ 3 & 2 \end{bmatrix}$$

Then the DP equation are not satisfies .
choose $\epsilon \in (0, 1)$ and $0 < c < \infty$ Now we choose a probability measure using an Linear Programming -

$\max_{\{p_a\}} \sum_a p_a Q_k^r(S', a)$
subject to.

1. $\sum_a p_a Q_k^c(S', a) \leq B + c.\epsilon^k$
2. $\sum_a p_a = 1$ ...(1)
3. $p_a \geq 0$

where $Q_k^r(S', a)$ and $Q_k^c(S', a)$ are given.

we solve the above LP and find out the $\{p_a^*\}$ then choose action a' according to $\{p_a^*\}$ if the LP (1) has not feasible solution then we increase our constant B by a small number ϵ and then try to solve our LP.
and then we update the Q factors according as

Data: α : learning rate ,γ : Discount factor

Result: A Q table containing $Q_k^r(S,A)$ and $Q_k^c(S,A)$ pairs defining estimated optimal policy π^*

/* Initialization */

Initialization $Q_k^r(S,A)$ and $Q_k^c(S,A)$ arbitrarily ;

/* For each step in each episode ,we calculate the Q- value and update the Q-table */

for each episode **do**

/* Initialize state S */

Initialize state S;

/* Choose Action A from S using p_a^* derived from solving LP */

$p_a^* \leftarrow$ solve LP (Q,S)
A $\leftarrow$ Select Action (S,p_a^*);

for each step in episode **do**

do

Take action A ,then observe reward R and next state S' ;

$p_a^* \leftarrow$ solve LP (Q, S')
$A' \leftarrow$ Select Action (S', p_a^*);

$Q_k^r(S,A) \leftarrow Q_k^r(S,A) + \alpha[R + \gamma max_{A'} Q_k^r(S',A') - Q_k^r(S,A)]$;

A" $\leftarrow$ select action $max_{A'} Q_k^r(S',A')$

$Q_k^c(S,A) \leftarrow Q_k^c(S,A) + \alpha[R + \gamma Q_k^c(S',A") - Q_k^c(S,A)]$;

S$\leftarrow S'$

A$\leftarrow A'$

while S is not terminal;

end

end

By using this algorithm we get the optimal Q value. and maximize the reward function .

5.5 Result:

5.5.1 when constraint are not added.

Value iteration
The value iterations of this algorithm is as follows:

For state 1 , the value iteration is 26.22116736809993.

For state 2 , the value iteration is 28.081632484379 .

Here $\delta = 0.01$
And $\left|v^{t+1} - v^{t}\right| < \delta$

5.5.2 when constraint are added.

➢ Results for Q-Learning with constraint :

- **For Constrained Problem-**

1. If we take B = 14 and c = 30 then

Decision variables p1 = 0.660490309002817
p2 = 0.339509690997183
state 0

Update Q - values for Rewards
[[24.93515373 26.57861389]
[1. 28.52686049]]

Update Q - values for Cost
[[13.09679097 16.13091529]
[1. 13.20892991]]

Decision variables p1 = 0.0136894095988471
p2 = 0.986310590401153
state 1

Update Q - values for Rewards
[[24.9369437 26.57861389]
[1. 28.42382089]]

Update Q - values for Cost
[[13.09469773 16.13091529]
[1. 13.36338419]]

➢ Results for Q-Learning with constraint :

For Constrained problem-

2. If we take B = 16 and c = 10 then

Decision variables p1 = 0.074976485937967
p2 = 0.925023514062033
state 0

Update Q - values for Rewards
[[23.39806759 26.80750689]
[1. 28.10185772]]

Update Q - values for Cost
[[12.30906716 15.78483654]
[1. 13.84252457]]

Decision variables p1 = 0
p2 = 1
state 1

Update Q - values for Rewards
[[25.273312934624506 26.769261765600312]
[1. 28.07116384731986]]

Update Q - values for Cost
[[11.35566892717285 15.82854520227306]
[1. 13.88943114292655]]

➢ Results for Q-Learning with constraint :

- **For Unconstrained Problem-**

1. If we take B = 30 and c = 50 then

Decision variables p1 = 0.0
p2 = 1.0
state 0

Update Q - values for Rewards
[[1. 27.18339149]
[1. 28.39040369]]

Update Q - values for Cost
[[1. 15.22491277]
[1. 13.41439447]]

Decision variables p1 = 0.0
p2 = 1.0
state 1

Update Q - values for Rewards
[[1. 27.18339149]
[1. 28.38868764]]

Update Q - values for Cost
[[1. 15.22491277]
[1. 13.41696853]]

➢ Results for Q-Learning with constraint :

- **For Unconstrained Problem**

2. If we take B = 45 and c = 50 then

Decision variables p1 = 0.0
p2 = 1.0
state 0

Update Q - values for Rewards
[[1. 27.44000059]
[1. 28.58554715]]

Update Q - values for Cost
[[1. 14.83999911]
[1. 13.12167927]]

Decision variables p1 = 0.0
p2 = 1.0
state 1

Update Q - values for Rewards
[[1. 27.44126093]
[1. 28.58554715]]

Update Q - values for Cost
[[1. 14.8381086]
[1. 13.12167927]]

Chapter 6

Conclusion and Future Work

Conclusion:

For the Constrained Problem:

- If we will take smaller B then we will increase the value of c to overcome the problem of infeasible solution of LP.
- The actions are used here by randomization .

For the Unconstrained Problem:

- If we take large value of B then the LP is unconstrained.
- For this, at every time step in state S_1 only action a_{12} and in S_2 only action a_{22} will be chosen , which is not by randomization.

Future Work:

Now in the future I will try to get more deep idea of Markov Decision Process and Q-learning and try to add some conditions in the problem and try to find that it will also work foe smaller values of B and execute it by Q-learning algorithm. And will check that if it will be work or not .

END OF REPORT

Bibliography

[1] M. L.PUTERMAN, "Markov decision processes," *A JOHN WILEY SONS, INC., PUBLICATION*, 2005.

[2] ""https://www.slideshare.net/MelakuEneayehu/reinforcement-learning-qlearning"."

[3] ""https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementati