Reinforcement Learning II
(Model-Free)

Ning Xiong

Mälardalen University

1

Content

- Motivation: what (target function) should be learnt for making decisions without a model?
- How to learn that (useful) target function in deterministic environment?
- How to learn that (useful) target function in stochastic environment?

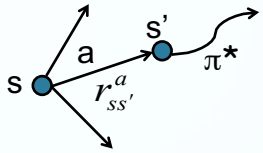
2

When Model is Available

Deterministic Case:

1. Predict best outcome for every act

$$r^a_{s,s'} + \gamma V^*(s')$$



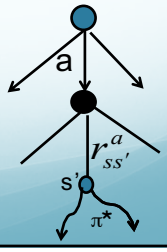
2. Select optimal act

$$a^* = \underset{a}{\operatorname{argmax}}(r^a_{s,s'} + \gamma V^*(s'))$$

Stochastic Case:

1. Predict best expected outcome for every act

$$\sum_{s'} P^a_{s,s'}\left[r^a_{s,s'} + \gamma V^*(s')\right]$$



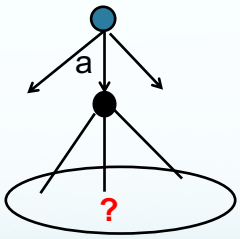
2. Select optimal act

$$a^* = \arg\max_a \sum_{s'} P^a_{s,s'}\left[r^a_{s,s'} + \gamma V^*(s')\right]$$

3

When Model Not Available

When the environment model is not available, we can not predict the possible successor states. Consequently we can not calculate the best expected result for every act even if the V* values are fully established.



- The best values V* for states are not useful for decisions without a model.

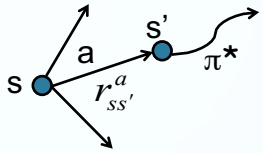
4

What Needs to Be Learnt

Deterministic Case:

1. Predict best outcome for every act

$$r^a_{s,s'} + \gamma V^*(s')$$



2. Select optimal act

$$a^* = \underset{a}{\operatorname{argmax}}\left(r^a_{s,s'} + \gamma V^*(s')\right)$$

Q*(S,a)

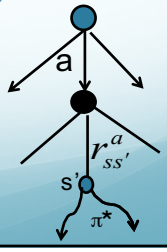
Stochastic Case:

1. Predict best expected outcome for every act

$$\sum_{s'} P^a_{s,s'}\left[r^a_{s,s'} + \gamma V^*(s')\right]$$

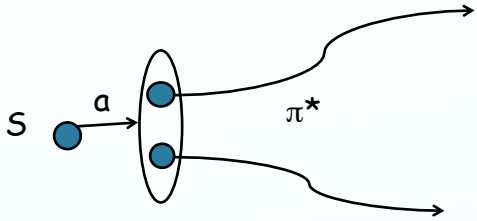
2. Select optimal act

$$a^* = \arg\max_a \sum_{s'} P^a_{s,s'}\left[r^a_{s,s'} + \gamma V^*(s')\right]$$



5

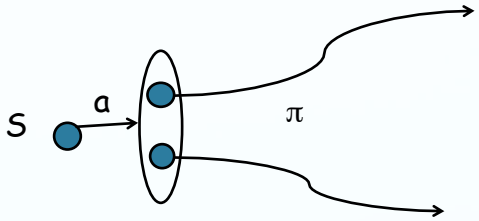
The Meaning of Q*(s,a)



- Q*(s,a) is the expected payoff when first taking the action at that state s and thereafter following the optimal policy π^*

6

Action Value Functions



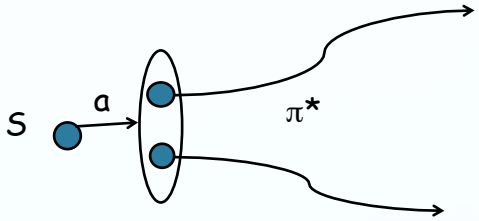
- The **value of taking an action in a state under policy π** is the expected payoff when first taking the action at that state and thereafter following π :

Action value function for policy π :

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\}$$

7

Optimal Action Value Functions

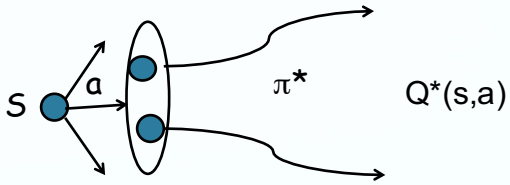


- The **optimal value of taking an action in a state** is the expected payoff when first taking the action at that state and thereafter following the optimal policy π^*

$$Q^*(s,a) = Q^{\pi^*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \text{for any } s,a$$

8

Optimal Action Value Function for Decision



- Once the optimal value function Q*(s,a) is available, we can simply chooses the action to maximize Q*

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a)$$

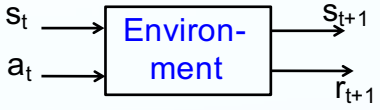
as the long term optimal choice

- Neither prediction nor model is entailed for decision when Q* function is fully established.
- Q* function is the target of learning when model is not available.

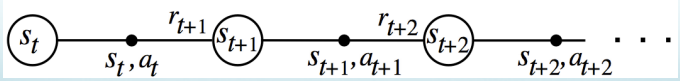
9

How to Learn Q* without Model

- Learn by interaction with the environment to obtain experiences



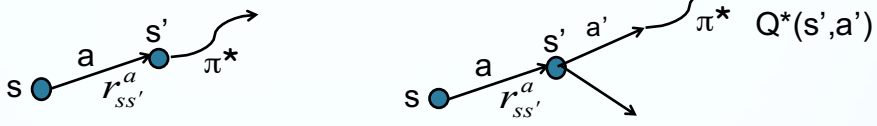
- Experience is expressed as sequence of states, actions, and rewards.



10

How to Learn Q* (Deterministic)

π: a + π* (from next)



$$Q^*(s,a) = V^\pi(s) = r^a_{ss'} + \gamma V^*(s')$$

$$V^*(s') = \max_{a'} Q^*(s',a')$$

$$Q^*(s,a) = r^a_{ss'} + \gamma \max_{a'} Q^*(s',a')$$

Optimal action value at a state can be derived from the optimal action values in the next state

11

Q-Learning Rule (Deterministic)

$$Q^*(s,a) = r^a_{ss'} + \gamma \max_{a'} Q^*(s',a')$$

Let Q(s,a) be the estimate for Q*(s,a), we can refine the estimate for a state through the current estimates at the successor state, so we can write:

$$Q(s,a) \leftarrow r^a_{ss'} + \gamma \max_{a'} Q(s',a')$$ Q-learning rule

- The next state and the reward is obtained through interaction, i.e., doing action *a* at state *s* and getting next state s' and the reward $r^a_{ss'}$

12

A Table Storing the Estimates

	a_1	a_2	…..	a_m
s_1	*	*	*	*
s_2	*	*	*	*
…	*	*	*	*
s_n	*	*	*	*

13

Q Learning Algorithm for Deterministic World

For every state-action pair (s,a), initialize the estimate Q(s,a) arbitarily

Observe current state s;
Do for ever:

1. Select an action *a* and execute it;
2. Receive immediate reward *r*;
3. Observe the new state *s'* ;
4. Update the estimate Q(s,a) as follows:

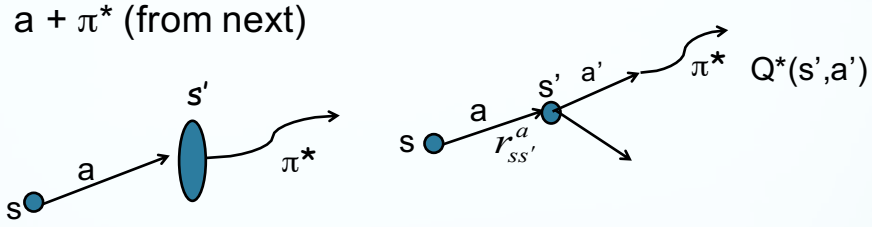
$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

5. s←s'

14

How to Learn Q* (Stochastic)

π: a + π* (from next)



$$Q^*(s,a) = V^\pi(s) = E\{r^a_{ss'} + \gamma V^*(s')\}$$

$$V^*(s') = \max_{a'} Q^*(s',a')$$

$$Q^*(s,a) = E\{r^a_{ss'} + \gamma \max_{a'} Q^*(s',a')\}$$

Monte-Carlo method: using sample information to estimate the expectation value

15

Idea of Monte Carlo Methods

- Roughly, the very basic idea of Monte Carlo method is to use **average of samples to approximate the expected value** of a random variable if theoretical solution is not available
- Example

A random variable X: $\{0.7/a_1, 0.3/a_2\}$

Exact: $E(X)=0.7a_1+0.3a_2$

Sample averaging:

$$Ava(X) = \frac{x_1 + \cdots x_N}{N} \approx \frac{(0.7N)a_1 + (0.3N)a_2}{N} = 0.7a_1 + 0.3a_2 = E(X)$$

16

Two Ways of Averaging Payoffs

The average of k payoffs can be simply calculated as

$$Q_k = \frac{R_1 + R_2 + \cdots R_k}{k}$$

Can we do this incrementally (without storing all the payoffs)?

$$Q_{k+1} = f(Q_k, R_{k+1})$$

17

Incremental Implementation

$$Q_{k+1} = \frac{1}{k+1}\sum_{i=1}^{k+1} R_i = \frac{1}{k+1}\left(R_{k+1} + \sum_{i=1}^{k} R_i\right) = \frac{1}{k+1}\left(R_{k+1} + kQ_k + Q_k - Q_k\right)$$

$$= \frac{1}{k+1}\left(R_{k+1} + (k+1)Q_k - Q_k\right) = Q_k + \frac{1}{k+1}\left(R_{k+1} - Q_k\right)$$

$$Q_{k+1} = Q_k + \frac{1}{k+1}\left[R_{k+1} - Q_k\right]$$

This is a common form for update rules:

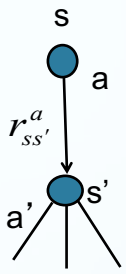
NewEstimate = *OldEstimate* + *StepSize*[Sample– *OldEstimate*]

Step size parameter: α=1/N (N is total number of samples including the new sample).

18

Employing Monte Carlo Method

$$Q^*(s,a) = E\{r^a_{ss'} + \gamma V^*(s')\}$$
$$= E\{r^a_{ss'} + \gamma \max_{a'} Q^*(s',a')\}$$



Revise the estimate using the sample value :

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r^a_{ss'} + \gamma \max_{a'} Q^*(s',a') - Q(s,a)\right]$$

Monte Carlo

Sample value

Step size parameter α=1/N, N is the number of times that the state-action pair (s,a) has been visited so far.

19

Q-Learning Rule (Stochastic)

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r^a_{ss'} + \gamma \max_{a'} Q^*(s',a') - Q(s,a)\right]$$

Monte Carlo

Sample value

Using estimates Q(s',a') to replace Q*(s,a)

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r^a_{ss'} + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Q-Learning rule

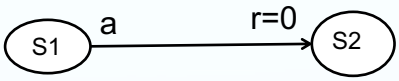
Step size parameter α=1/N, N is the number of times that the state-action pair (s,a) has been visited so far.

20

Example: Updating Q Estimates

Q(s1, a)=72, Visit(S1,a)=20 γ=0.9

Q(s2, b1)=63, Q(s2, b2)=100, Q(s2, b3)=81



Deterministic:

Q(S1,a)=r+γmax$_b$Q(S2,b)=0+0.9max(63,100,81)=90

Stochastic:

$$Q(S1,a) = Q(S1,a) + (r + 0.9 \max Q(S2,b) - Q(S1,a)) / Visit(S1,a)$$
$$= 72 + (0 + 0.9 \max(63,81,100) - 72) / 20 = 72.9$$

21

Q Learning Algorithm for Stochastic World

For every state-action pair (s,a), initialize the estimate Q(s,a) arbitarily

Observe current state s;
Do for ever:

1. Select an action *a* and execute it;
2. Receive immediate reward *r*;
3. Observe the new state *s'*;
4. Update the estimate Q(s,a) as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r_{ss'}^{a} + \gamma \max_{a'} Q(s',a',) - Q(s,a)\right]$$

5. s←s'

22

How to Choose Action in Learning

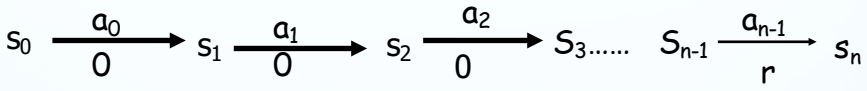
- In principle, actions should be taken randomly during learning to give a chance to every legal action at each state (**exploration**)
- If learning is performed when doing real task, we should also use the learning results, i.e. current estimates of Q*, to choose possibly good actions (**exploitation**)
- The actual decisions on the acts should be a compromise between exploration and exploitation.
- Action selection strategy: actions with higher estimates of Q* have a higher chance to be selected:

$$P(a_i \mid s) = \frac{\kappa^{Q(s,a_i)}}{\sum_j \kappa^{Q(s,a_j)}} \qquad (\kappa > 1)$$

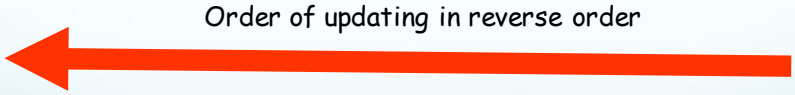
23

A Practical Suggestion

When dealing with a sequence of actions with rewards very delayed at terminal state, perhaps doing updating in the reverse order could increase the efficiency of learning.



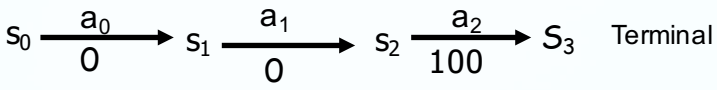
Order of updating in reverse order



It needs storing the whole episode before training

24

Example: Updating with a Sequence



Suppose deterministic environment and initial estimates of Q* values for all state-action pairs to be zero. $\gamma=0.9$

Applying Q learning rule (1st time)

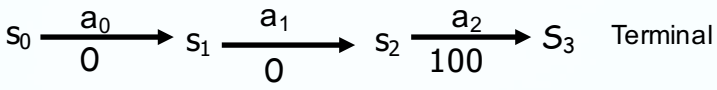
$Q(s_0,a_0)=0+\gamma \max_{a'} Q(s_1,a')=0$
$Q(s_1,a_1)=0+\gamma \max_{a'} Q(s_2,a')=0$
$Q(s_2,a_2)=100+\gamma \max_{a'} Q(s_3,a')=100$

Applying Q learning rule (2nd time)

$Q(s_0,a_0)=0+\gamma \max_{a'} Q(s_1,a')=0$
$Q(s_1,a_1)=0+\gamma \max_{a'} Q(s_2,a')=0+\gamma Q(s_2,a2)=90$
$Q(s_2,a_2)=100+\gamma \max_{a'} Q(s_3,a')=100$

25

Example: Updating with a Sequence



Suppose deterministic environment and initial estimates of Q* values for all state-action pairs to be zero. $\gamma=0.9$

Applying Q learning rule (3rd time)

$Q(s_0,a_0)=0+\gamma \max_{a'} Q(s_1,a')=0+\gamma Q(s_1,a1)=81$
$Q(s_1,a_1)=0+\gamma \max_{a'} Q(s_2,a')=0+\gamma Q(s_2,a2)=90$
$Q(s_2,a_2)=100+\gamma \max_{a'} Q(s_3,a')=100$

Applying Q learning rule (once but in reverse order)

$Q(s_2,a_2)=100+\gamma \max_{a'} Q(s_3,a')=100$
$Q(s_1,a_1)=0+\gamma \max_{a'} Q(s_2,a')=0+\gamma Q(s_2,a2)=90$
$Q(s_0,a_0)=0+\gamma \max_{a'} Q(s_1,a')=0+\gamma Q(s_1,a1)=81$

26

Function Approximation

- Q learning maintains an explicit lookup table, conducts a sort of rote learning. Every entry in the table has to be updated directly. The table can be huge in continuous and high-dimensional cases

	a_1	a_2	…..	a_m
s_1	*	*	*	*
s_2	*	*	*	*
…	*	*	*	*
s_n	*	*	*	*

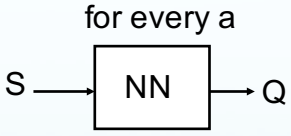
- Incorporating function approximation with Q learning.
  Idea is to replace the lookup table with a continuous target function learned with a learning algorithm. The target function can be used to estimate the Q* values of actions in continuous input situation.

27

Neural Network Approximating Q* Values

Use Neural network to approximate Q* values

Use updated $Q(s,a)$ as training example for revising weights of NN



But using the neural network might cause the learning algorithm to fail to converge

Ning's view: fuzzy systems may be better by more focused revision(change only those rules that are fired)

28

Reading Guidance

1. Read carefully the slides, as they extract and combine material from different books on the topic.

2. Read chapter 13 of the "Machine learning" book by Mitchell

3. Read some sections of the "Reinforcement learning" book as reference. Examples are:

   - Section 2.5 for incremental averaging
   - Section 6.5 for Q-learning

29