Machine Learning Assignment 4

Que Shuhao(4739124)

May 10, 2018

Exercise 1

$r_{min} \sum_{h=0}^{\infty} \gamma^h \leq R_t = \sum_{h=0}^{\infty} \gamma^h r_{t+h+1} \leq r_{max} \sum_{h=0}^{\infty} \gamma^h$
where,
$\sum_{h=0}^{\infty} \gamma^h = \frac{1-\gamma^{\infty}}{1-\gamma}$ is bounded for $0 \leq \gamma < 1$
When $0 \leq \gamma < 1$, $\sum_{h=0}^{\infty} \gamma^h = \frac{1-\gamma^{\infty}}{1-\gamma} = \frac{1}{1-\gamma}$
then,
$\frac{r_{min}}{1-\gamma} \leq R_t \leq \frac{r_{max}}{1-\gamma}$
Therefore, for bounded $-10 \leq r \leq 10$, R_t is also bounded as $\frac{-10}{1-\gamma} \leq R_t \leq \frac{10}{1-\gamma}$

Exercise 2

1. Q_0

+1	0 0	0 0	0 0	0 0	+5

2. Q_1

+1	1 0	0 0	0 0	0 5	+5

3. Q_2

+1	1 0	0.5 0	0 2.5	0 5	+5

4. Q_3

+1	1 0.25	0.5 1.25	0.25 2.5	1.25 5	+5

5. Q_4

Therefore, the optimal policy $\pi*$ will be:

+1	1 0.625	0.5 1.25	0.625 2.5	1.25 5	+5

+1	$\Leftarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	+5

Exercise 3

1. $\gamma = 0$

action \ state	1	2	3	4	5	6
left	0.0	1.0	0.0	0.0	0.0	0.0
right	0.0	0.0	0.0	0.0	5.0	0.0

Table 1: optimal value functions Q^*

2. $\gamma = 0.1$

action \ state	1	2	3	4	5	6
left	0.0	1.0	0.1	0.01	0.05	0.0
right	0.0	0.01	0.05	0.5	5.0	0.0

Table 2: optimal value functions Q^*

3. $\gamma = 0.9$

action \ state	1	2	3	4	5	6
left	0.0	1.0	3.2805	3.645	4.05	0.0
right	0.0	3.645	4.05	4.5	5.0	0.0

Table 3: optimal value functions Q^*

4. $\gamma = 1$

action \ state	1	2	3	4	5	6
left	0.0	1.0	5.0	5.0	5.0	0.0
right	0.0	5.0	5.0	5.0	5.0	0.0

Table 4: optimal value functions Q^*

When $\gamma = 0$, the optimal policy will be [left, uncertain, uncertain, right]; when $\gamma = 0.1$, the optimal policy will be [left, left, right, right]; when $\gamma = 0.9$, the optimal policy will be [right, right ,right, right]; when $\gamma = 1$, the optimal policy will be [right, uncertain, uncertain, uncertain].

As the value of the discount factor γ increases, different optimal policies are generated. And that is because, when $\gamma = 0$, the agent will only consider the current reward (reward of next state) and neglect the future rewards (rewards of states further from the current state than the next state). As the value of γ increases and smaller than 1, different degrees of importance will be assigned to different states depending on their distances to the current state. When γ is equal to 1, same degree of importance will be assigned to all the other states, in which case assuming the current state is 1, although state 6 is the furthest from state 1, since state 6 carries the biggest reward value 5, the agent will still choose to go right even if it is in state 1.

Exercise 4

In this exercise, $\gamma = 0.5$, and different plots are displayed for different values of ϵ and α.

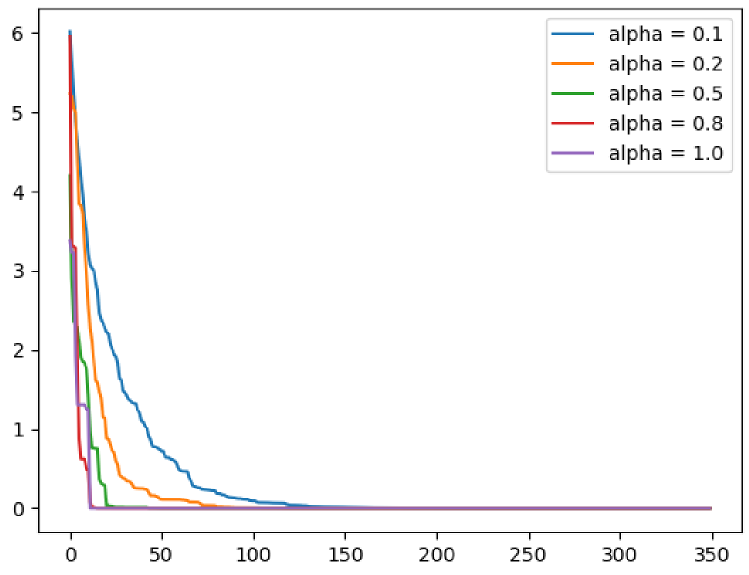

Figure 1: 2-norm differences over the number of interactions with the system, $\epsilon = 0.9$

As illustrated in Figure 1, when the value of α increases from 0.1 to 1.0, the convergence speed of Q-learning algorithms increases as well. Because α indicates the learning rate. If the learning rate is high, in each interaction with the system, the agent will learn more about next state and consequently the optimal value function Q can be obtained more quickly(i.e. the Q learning converges more quickly). If the learning rate is low, then in each interaction with the system, the agent will learn less about the next state and consequently it will cost more interaction times to obtain the optimal value function Q.

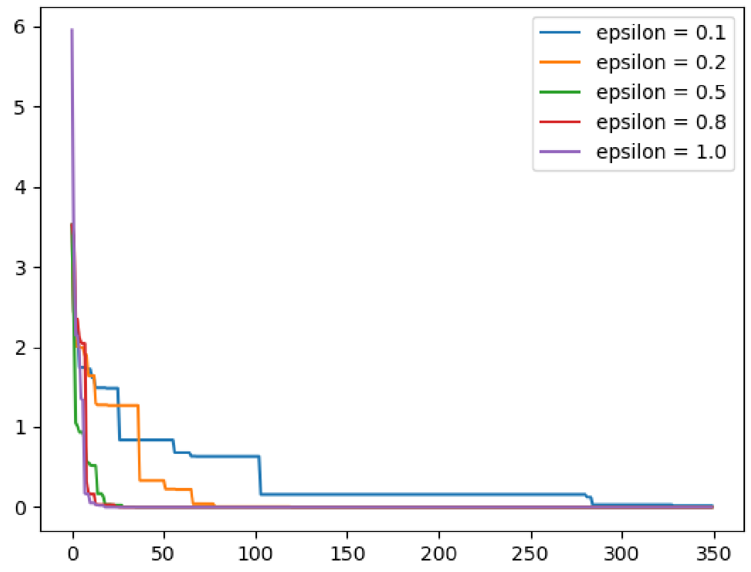

Figure 2: 2-norm differences over the number of interactions with the system, $\alpha = 0.8$

As illustrated in Figure 2, when the value of ϵ increases from 0.1 to 1.0, the convergence speed of Q-learning algorithms increases as well. Because ϵ indicates the degree of exploration. If ϵ is high, in each interaction with the system, the agent will be more likely to explore more and choose a random action and consequently the optimal value function Q will be obtained more quickly. If ϵ is low, in each interaction with the system, the agent will be more likely to explore less and act greedy(i.e. making decisions based on previous knowledge), and consequently it will take more interaction times to obtain the optimal value function Q.

Exercise 5

Now that the robot has the probability of 30% that it stays at the same state, the transition probability from one state to another will be 70% instead of 100%

state / action	1	2	3	4	5	6
left	0.0	0.82352941	0.39295746	0.49867698	1.21107266	0.0
right	0.0	0.36788113	0.69814777	1.69550173	4.11764706	0.0

Table 5: optimal value functions Q^* of Q-iteration

As illustrated in Figure 3 and 4, the output of Q-learning cannot converge to a fixed value matrix completely. There is always some randomness involved. In this case, the same values of parameter ϵ is used while the values of parameter α have to be 10 times smaller than previous settings to show relatively stable results. In this case, alpha = 0.01 yields the best results in terms of convergence.

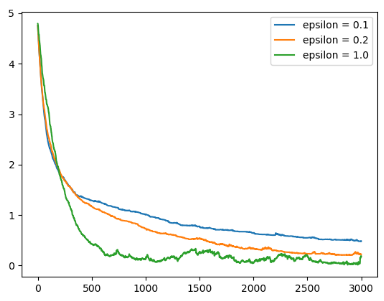

Figure 3: 2-norm differences over the number of interactions with the system, $\alpha = 0.01$

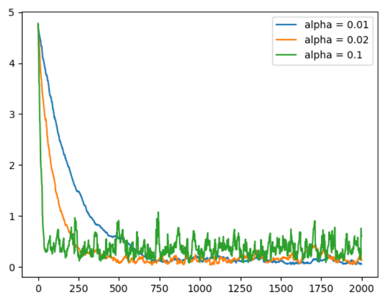

Figure 4: 2-norm differences over the number of interactions with the system, $\epsilon = 0.9$

Exercise 6

Pseudo-Code[1] is provided in Figure 5

As shown in Figure 6, 7 and 8, as the value of beta[2] decreases, the width of the basis functions increases, and consequently the Q value curve becomes smoother. Through experiments, it is found out that when beta exceeds the value of 50 or a bit higher, the difference between the output plots can no longer be perceived. Therefore, I set beta value as 50 to mimic when sigma is equal to 0.

The number of basis functions also has an influence on the performance. In the experiment, two sets of centers are tested, which are [1,2,3,4,5,6] and [2,3,4,5] respectively. As illustrated by the Figure 10, it can be observed that with less basis functions, the RBFN[3] fails to perceive the Q values on terminal states (s<1.5 and s>5.5). But still the generated optimal policy is the same as when there are 6 basis functions.

As for the minimum number of basis functions needed, according to the fact that basis function[4] is to include prior knowledge into the system. There are 2 terminal states with rewards and that information about the position of those states should spread in a wave-like manner, thus, the radial bases. The waves are one-dimensional and represented by the Gaussian kernels. Therefore, at least two kernel functions are needed to capture the two peaks (2 terminal states 1.5 and 5.5). When there are only 2 kernel functions, in order to spread the information of each peak to all the states, the width of kernel function should be large, i.e. the value of beta should be small enough. The results are shown

[1]https://danieltakeshi.github.io/2016/10/31/going-deeper-into-reinforcement-learning-understanding-q-learning-and-linear-function-approximation/

[2]http://mccormickml.com/2015/08/26/rbfn-tutorial-part-ii-function-approximation/

[3]$https://en.wikipedia.org/wiki/Radial_basis_function_network$

[4]$https://en.wikipedia.org/wiki/Radial_basis_function$

in Figure 11 and 12.

```
Initialization: centers
                width = beta,  beta = 1/(2*sigma^2)
                Using Gaussian as Basis Function
                Construct basis functions RBF(s) using centers and the same width
                left_weights = [0,0,0,0,0,0]
                right_weights = [0,0,0,0,0,0]
                alpha = 0.5
                gamma = 0.7
                epsilon = 0.7
Iteration for x times:
   choose initial state s
   while 1.5 < s < 5.5 :
        calculate Q(s,left) = left_weights*RBF(s)
                  Q(s,right) = right_weights*RBF(s)
        choose action:
                  a = arg max(Q(s,left),Q(s,right))
                  with probability epsilon that:
                  a = random(left,right)
        next state:
                  s' = s + a + normal(0,0.01)
                  Q(s',left) = left_weights*RBF(s')
                  Q(s',right) = right_weights*RBF(s')
        if action == 1:
                  right_weights = right_weights + alpha * (R + gamma*max(Q(s',left), Q(s',right)) - Q(s,right)) * RBF(s)
        if action == -1:
                  left_weights = left_weights + alpha * (R + gamma*max(Q(s',left), Q(s',right)) - Q(s,left)) * RBF(s)
        update state:
                  s = s'
```

Figure 5: Pseudo-Code

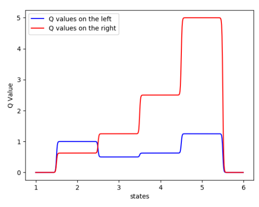

Figure 6: beta = 50

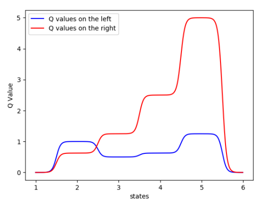

Figure 7: beta = 10

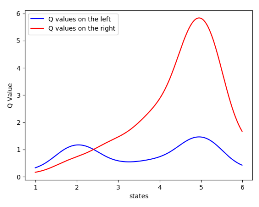

Figure 8: beta = 2

Figure 9: Optimal Q values, 6 basis functions centered at [1,2,3,4,5,6]

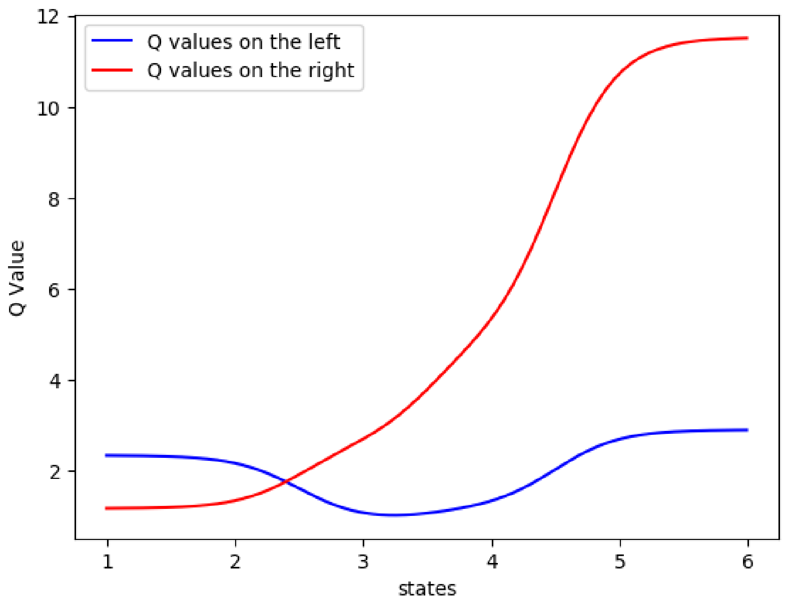

Figure 10: Optimal Q values, 4 basis functions centered at [2,3,4,5], beta = 2

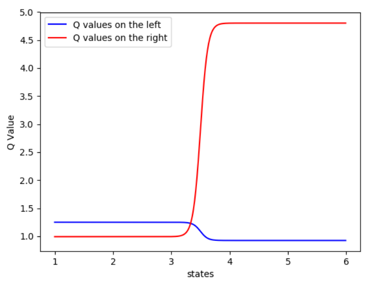

Figure 11: Optimal Q values, 2 basis functions centered at [1.5, 5.5], beta = 2

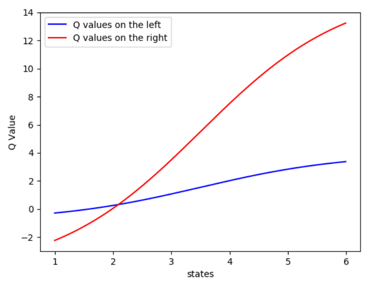

Figure 12: Optimal Q values, 2 basis functions centered at [1.5, 5.5], beta = 0.1