Machine Learning Assignment 6

Xiaoxu Gao | 4504348 | highsmallxu@gmail.com

Exercise1

$R_t = \sum_{h=0}^{\infty} \gamma^h r_{t+h+1}$

In this formula, if we can prove that $\gamma^h r_{t+h+1}$ is going to become zero when h is positive infinite, then the sum will be bounded. Here, γ is in the range of [0,1) and r_{t+h+1} is in the rage of [-10,10].

Since γ is less than 1, γ^h will decrease to nearly zero exponentially. And r_{t+h+1} will not essentially change the tendency. Therefore, we can prove that sum is bounded.

Exercise2

$\gamma = 0.5$

Q-iteration
Input: dynamics *f*, reward function *p*, discount factor γ
1. initialize Q-function, e.g. Q_0<-0
2. **repeat** at every iteration *l*=0,1,2,...
3. **for** every (x, u) **do**
4. $Q_{l+1}(x, u)$ <- $p(x, u) + \gamma max_{u'} Q_l(f(x, u), u')$
5. **end for**
6. **until** $Q_{l+1} = Q_l$
Output: $Q^* = Q_l$

iteration	S1(L/R)	S2(L/R)	S3(L/R)	S4(L/R)	S5(L/R)	S6(L/R)
1	0/0	1.0/0	0.5/0	0.25/0	0.125/5.0	0/0
2	0/0	1.0/0	0.5/0	0.25/2.5	0.125/5.0	0/0
3	0/0	1.0/0	0.5/1.25	0.25/2.5	0.125/5.0	0/0
4	0/0	1.0/0.625	0.5/1.25	0.25/2.5	0.125/5.0	0/0

The optimal policty π^* is to do left in S_2, to do right in S_3, S_4, S_5

Exercise3

Q1: value functions

$\gamma = 0$

A/S	S1	S2	S3	S4	S5	S6
Left	0	1	0	0	0	0
Right	0	0	0	0	5	0

$\gamma = 0.1$

A/S	S1	S2	S3	S4	S5	S6
Left	0	1	0.1	0.01	0.05	0
Right	0	0.01	0.05	0.5	5	0

$\gamma = 0.9$

A/S	S1	S2	S3	S4	S5	S6
Left	0	1	3.2805	3.6450	4.05	0
Right	0	3.6405	4.05	4.5	5	0

$\gamma = 1$

A/S	S1	S2	S3	S4	S5	S6
Left	0	1	5	5	5	0
Right	0	5	5	5	5	0

Q2: discount factor

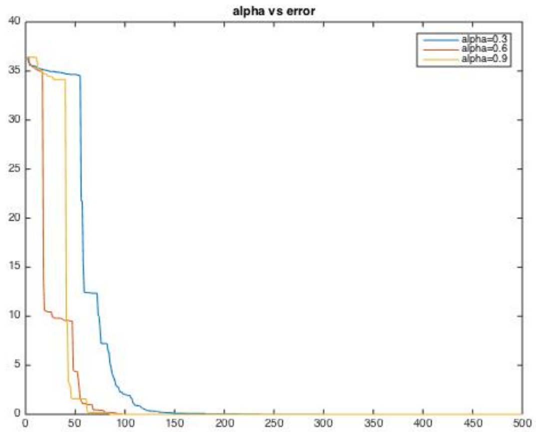
γ is used to decide how future steps would affect the value function. if γ is low, it means that we want quick reward. if γ is high, it means that we want reward with high value no matter how many steps we actaully need.

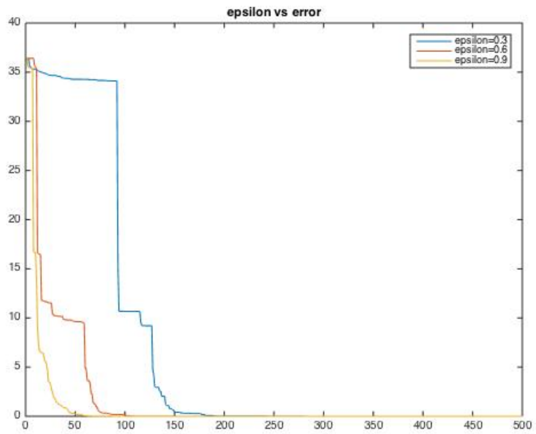
Q3: $\gamma = 1$

In this case, when agent reaches to state 1 or state 6, it will stop. So, even if $\gamma = 1$, value function will stop growing at some point.

Exercise4

Q-learning
Input: discount fator γ
1. initialize Q-function
2. measure initial state x_0
3. **for** every time step k=0,1,2...**do**
4. u_k <- exploration / exploitation
5. $Q_{k+1}(x_k, u_k)$ <- $Q_k(x_k, u_k) + a_k[r_{k+1} + \gamma * max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k)]$
6. **end for**





α is learning rate, ϵ is for deciding to use exploration mode or exploitation mode. We can see from the above figures that if α is lager, it will converge faster, because the step is larger; it influences the number of transitions required by Q-learning to obtain a good solution. if ϵ is larger, it will also converge faster. The agent chooses the action that it believes has the best long-term effect with probability $1 - \epsilon$.

Exercise5

Q-iteration

$Q_{l+1}(x,u)$ <- $\sum_{x'} f(x,u,x')[p(x,y,x') + \gamma max_{u'} Q_l(f(x,u),u')]$

```
Q(j,z) = 0.7*(reward(state(j),action(z)) + gamma *
max(Q_old(model(state(j),action(z)),:)))...
                +0.3*(0 + gamma * max(Q_old(j,:)));

result: the number of iteration increases to 9 before it converges.
```

A/S	S1	S2	S3	S4	S5	S6
Left	0	0.8235	0.3929	0.4986	1.2111	0
Right	0	0.3678	0.6981	1.6955	4.1176	0

Q-learning

```
for Q-learning, it doesn't matter if we have transition probabilities,
because the frequency of obsevering each s' already depends on them.
```

Exercise6

In a continous observation space, it is impossible for the agent to visit every state and store the value for this state in a table. This is why the value function needs to be approximated. RBF is continuous-valued, so the feature values will be in the range of [0,1], reflecting the degree of "presence" of the feature. The feature value is calculated based on the distance of the measuring point to the center and the width of RBF.

In terms of width, if RBF is not wide enough, then the values for some feature maybe too small to be distinguished from zero. On the other hand, if RBF is too wide which means there is a high degree of generalization and may cause instability in the value function.

Q-learning using RBF as function approximator
1. for l=1,2,..
2. initialize s
3. Q_a <- $\sum_{i=1}^{n} \theta_a(i) f_s(i)$
4. repeat
5. a <- $argmax_a Q_a$ or a <- random action
6. δ <- $r - Q_a^*$
7. $Q_a < -\sum_{i=1}^{n} \theta_a(i) f_s'(i)$
8. update a',δ, s 9. until s is terminal
10. end for

Exercise7

Policy iteration:
Input: policy π to be evaluated, dynamics f, reward function p, discount factor γ
1. initialize Q-function
2. **repeat** at every iteration l = 0,1,2,...
3. **for** every (x, u) **do**
4. $Q_{l+1}^{\pi}(x, u)$ <- $p(x, u) + \gamma Q_l^{\pi}(f(x, u), \pi(f(x, u)))$
5. **end for**
6. **until** convergence
Output: $Q^{\pi} = Q_l^{\pi}$

```
Final policy: -1 -1 1 1 1 -1
Iteration of Q: 11
Iteration of policy: 2

Advantages:
1. It can converge faster.

Disadvantages:
1. It is not necessarily computationally less costly than value iteration,
because maybe every policy iteration requires more time.
```

Appendix1: Implement Q-iteration in MATLAB

```
gamma = 0.5;
epsilon = 0.001;

state = [1,2,3,4,5,6];
action = [-1,1];
Q = zeros(length(state),length(action));
Q_old = Q;

for i = 1:4
    for j = 1:length(state)
        for z = 1:length(action)
            Q(j,z) = reward(state(j),action(z)) +
gamma*max(Q_old(model(state(j),action(z)),:));
        end
    end
    if abs(sum(sum(Q - Q_old))) < epsilon
        break;
    else
        Q_old = Q;
    end
end
```

Appendix2: Implement Q-learning in MATLAB

```
gamma = 0.5;
alpha = 0.5;
epsilon = 0.5;

state = [0,1,2,3,4,5];
action = [-1,1];
Q = zeros(length(state),length(action));
state_idx = 1;

for i=1:250
    r = rand;
    x = sum(r>=cumsum([0,1-epsilon,epsilon]));

    if x==1 % exploit
        [~,umax] = max(Q(state_idx,:));
        current_action = action(umax);
    else % explore
        current_action = datasample(action,1);
    end

    action_idx = find(action==current_action);
    [next_state,next_reward] = next(state(state_idx),action(action_idx));
    next_state_idx = find(state==next_state);

    Q(state_idx,action_idx) = Q(state_idx,action_idx) + alpha*(next_reward +
gamma*max(Q(next_state_idx,:)) - Q(state_idx,action_idx));

    if(next_state_idx==6)||(next_state_idx==1)
        state_idx = datasample(2:length(state)-1,1);
    else
        state_idx = next_state_idx;
    end
end

function [next_state,next_reward] = next(s,a)
    if(s<=4 && s>=1)
        next_state = s + a;
    else
        next_state = s;
    end

    if(s==4 && a==1)
        next_reward = 5;
    elseif(s==1 && a==-1)
        next_reward = 1;
    else
        next_reward = 0;
    end
end
```