Reinforcement Learning: ML assignment 4

Ang Li (4593820)

May 9th, 2018

Exercise 1

The return can be expressed as

$$R_t = \sum_{h=0}^{\infty} \gamma^h r_{t+h+1} \quad (1.1)$$

with the given conditions as $0 \leq \gamma < 1$ and $-10 \leq r_{t+h+1} \leq 10$

Firstly we scale the bounded reward to its upper bound as $r_{t+h+1} \leq 10$, and substitute into equation (1.1), yielding

$$R_t \leq 10 \sum_{h=0}^{\infty} \gamma^h \quad (1.2)$$

which contains a geometric sequence that has be rewritten as $\sum_{h=0}^{\infty} \gamma^h = \frac{0^0(1-\gamma^{h\to+\infty})}{1-\gamma}$

Since $0 \leq \gamma < 1$, one can derive that $\gamma^{h\to+\infty}$ will exponentially decrease to zero. Substituting it back into formula (1.2), gives a bounded Rt

$$R_t \leq \frac{10}{1-\gamma} \quad (1.3)$$

Exercise 2

Implementing the Q-iteration with a zero-value initialization of Q function, gives the value after each iteration as,

Table 2.1 Q-values for each iteration

Iteration	Action	State 1	State 2	State 3	State 4	State 5	State 6
1	Left	0	1	0	0	0	0
	Right	0	0	0	0	5	0
2	Left	0	1	0.5	0	0	0
	Right	0	0	0	2.5	5	0
3	Left	0	1	0.5	0.25	1.25	0
	Right	0	0.25	1.25	2.5	5	0
4	Left	0	1	0.5	0.625	1.25	0
	Right	0	0.625	1.25	2.5	5	0

The optimal policy $\prod_2^* = Left,\ \prod_3^* = Right,\ \prod_4^* = Right,\ \prod_5^* = Right$

Exercise 3

The value functions for $\gamma = 0,0.1,0.9,1$ can be obtained from a zero-value initialization as,

$\gamma = 0$

Action	State 1	State 2	State 3	State 4	State 5	State 6
Left	0	1	0	0	0	0
Right	0	0	0	0	5	0

$\gamma = 0.1$

Action	State 1	State 2	State 3	State 4	State 5	State 6
Left	0	1	0.1	0.01	0.05	0
Right	0	0.01	0.05	0.5	5	0

$\gamma = 0.9$

Action	State 1	State 2	State 3	State 4	State 5	State 6
Left	0	1	3.2805	3.6450	4.05	0
Right	0	3.6450	4.05	4.5	5	0

$\gamma = 1$

Action	State 1	State 2	State 3	State 4	State 5	State 6
Left	0	1	5	5	5	0
Right	0	5	5	5	5	0

The discount factor represents for the future influence on the value function. High discount factor contributes to more concentration on future rewards despite of a large number of steps; while low discount factor reflects more on current rewards.
In this exercise when the robot reaches the state 1 or state 6, it will stop. Therefore even if the discount factor is set as 1, the return function will not unboundedly grow.

Exercise 4

The difference (2-norm) between the value function estimated by Q-learning and the true value function (table above) over the number of iterations with zero-value initialization are plotted as in Figure 4.1. Observations

demonstrate that generally higher α contributes to a faster convergence because of the larger and steeper step it takes; The epsilon greedy ϵ also helps increasing convergence speed when learning rate is fixed, but extremely low exploration ($\epsilon = 0$) will have convergence issues that the error will not converge to global minimum, due to the fact that the system will not discover any global minimum if no exploration action exists.

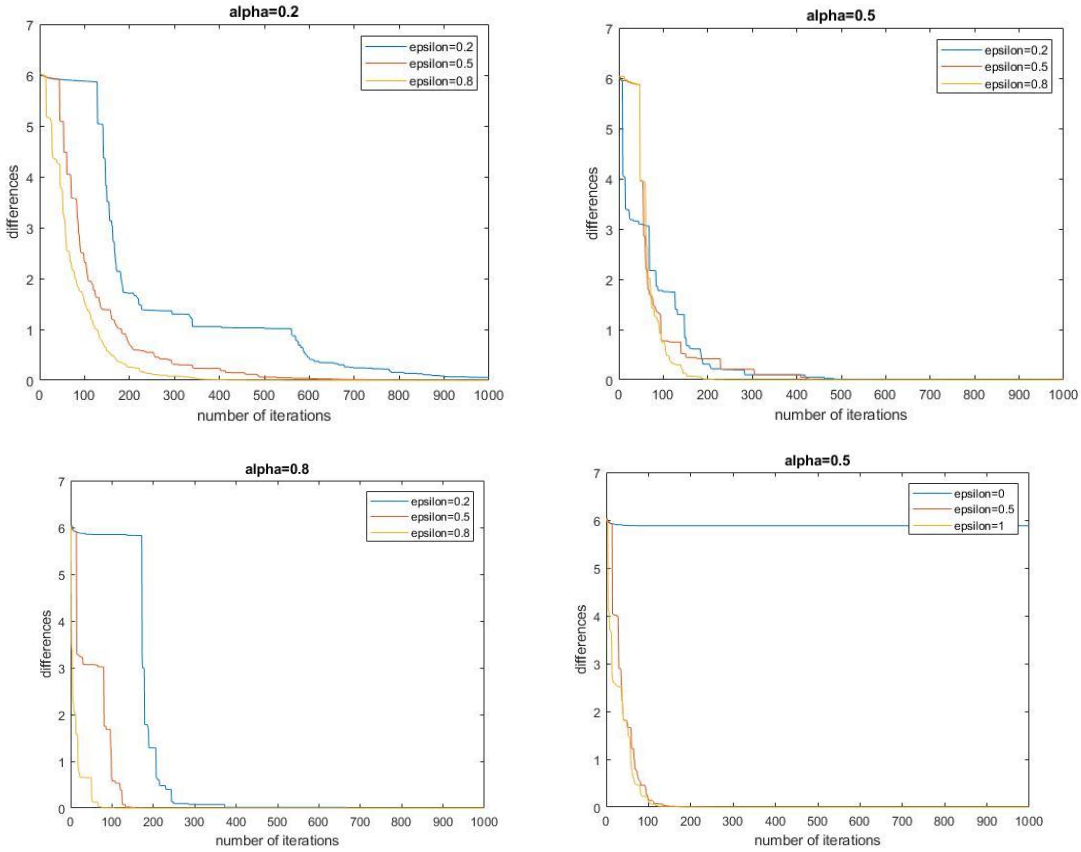

Figure 4.1 Plots of different epsilon and alpha(First row alpha=0.2, 0.5; second row alpha=0.8 and the special condition respectively)

Exercise 5

Q-iteration:

The probability term of staying at the same state is added in Bellman optimality equation, which gives a following optimum value function as,

Action	State 1	State 2	State 3	State 4	State 5	State 6
Left	0	0.8235	0.3929	0.4986	1.2111	0
Right	0	0.3678	0.6981	1.6955	4.1176	0

The optimum value function changed and the iteration needed for convergence increased to 9.

Q-learning:
Nothing changed for Q-learning since Q-learning algorithm does not learn a model of the environment, which means that the transition probabilities do not need to be considered in this case.

Exercise 6

In this exercise, the model parameters are set as $\gamma = 0.5, \alpha = 0.5, \epsilon = 0.1$. Since the state space is not discrete anymore, the value function approximation is evaluated by generalized radial basis function networks, where the Q values are computed by the dot product of the vector of weights and output vector from RBF. The variance of the Gaussian RBF is initialized as 0. 1, and number of Gaussian RBF is initialized as 6. These functions are centered at {1, 2, 3, 4, 5, 6} as recommenced. The weights are initialized as ones. Considering the number of features and two different actions, a total number of 2n weights are needed with n RBF functions. The pseudo-code can be found as follows. Compared with the Q-learning implementation in a discrete state-action space, this case considers of the convergence of the weight to make them optimal, while the Q-learning algorithm with epsilon-greedy policy remains the same. Furthermore, the pseudo code is given as follows,

Model parameters initialization
Loop
    Initialize state
    Approximate the value functions by RBF
    Loop
        Choose action based on epsilon-greedy policy
        Update the state and corresponding reward
        Update the value functions $\delta = R + \gamma \max_{a' \leftarrow A} Q(s', a') - Q(s, a)$
        Update the vector of weights $\theta = \theta + \alpha \delta \varphi(s, a)$
    until reaching the terminal state
until the vector of weights achieve convergence

Starting from $\sigma = 0.1$, the expected return for different actions and convergence check are plotted in Figure 6.1. Apparently the algorithm does not converge at all, showing high frequency oscillations in the difference of the weights. Followed by the results proposed in exercise 5, we can adjust the model parameters α, ϵ to 0.5 and 0.05 respectively to achieve the convergence. In this case, the error quickly converged, with a distribution of the expected return as shown in Figure 6.2, which requires 7563 iterations for convergence. Increasing σ from 0.1 to 0.5, we can find the distribution of expected return and the fact that the convergence issues appear again, as shown in Figure 6.3. Therefore, the larger width of RBF contributes to a slower convergence speed by introducing some degree of instability, and a wider distribution of the expected return evolution in the state-action space. This is, small width of the Gaussian may not the enough the represent all the features, while large width has the possibility to cause instability issues with high-frequency oscillations.

For the influence of the number of RBF, results with 3 RBF, 5RBF are plotted in Figure 6.4, 6.5 respectively. We can observe that the optimal policy in state 3 changes from 5 RBF to 6 RBF without convergence. This phenomenon is caused by the large interval in the state. After 6 RBF, increasing number of RBF does not change the response anymore and thus 6 would be the optimal number with less computational efforts.

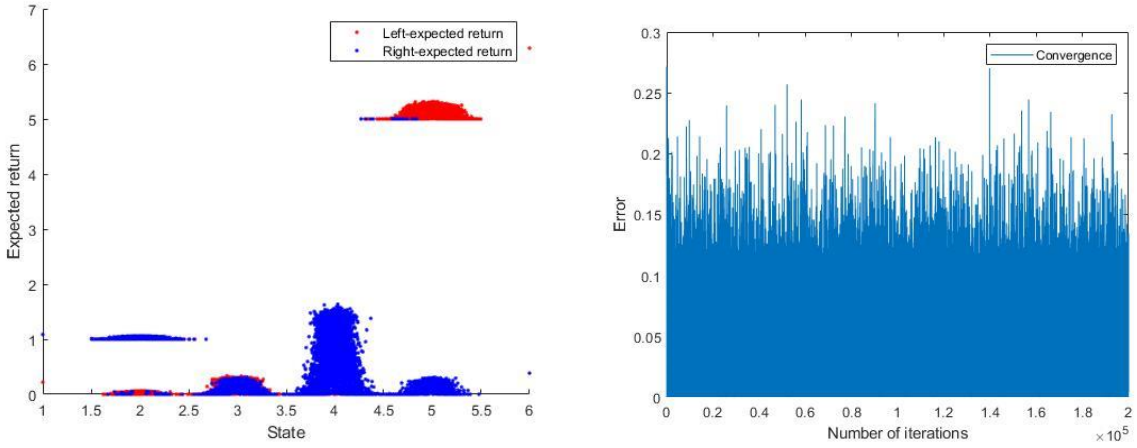

Figure 6.1 Expected return and convergence check for sigma=0.1 with initial model parameters

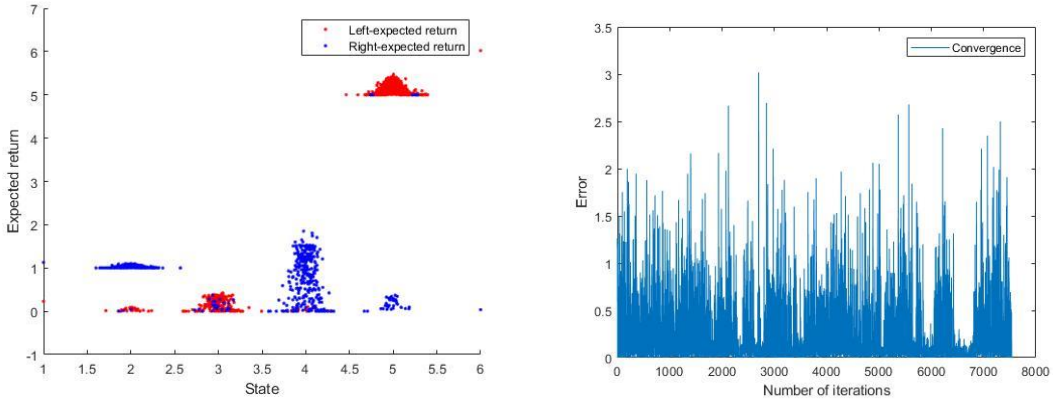

Figure 6.2 Expected return and convergence check for sigma=0.1 with $\alpha = 0.5, \epsilon = 0.05$

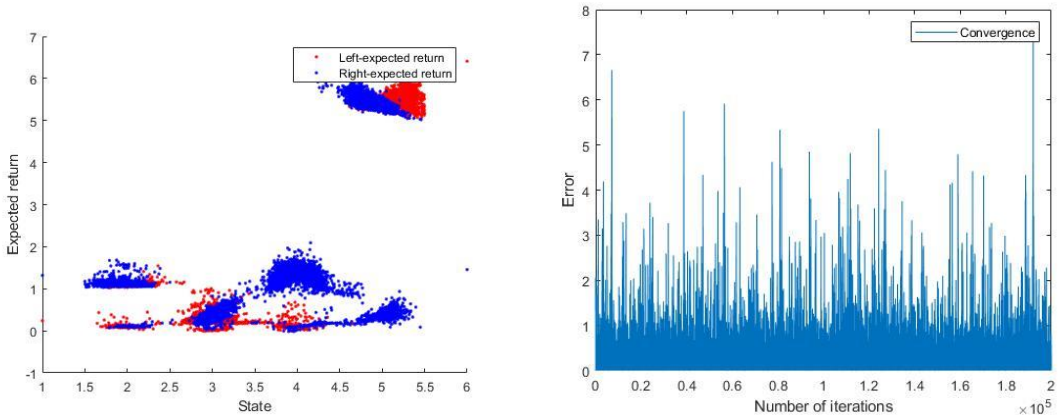

Figure 6.3 Expected return and convergence check for sigma=0.5 with $\alpha = 0.5, \epsilon = 0.05$

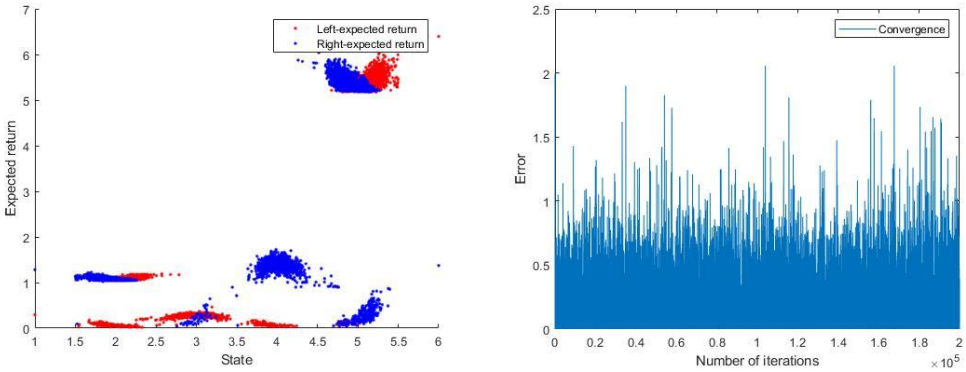

Figure 6.4 Expected return and convergence check for sigma=0.5 with $\alpha = 0.5, \epsilon = 0.05$ and 3 RBF

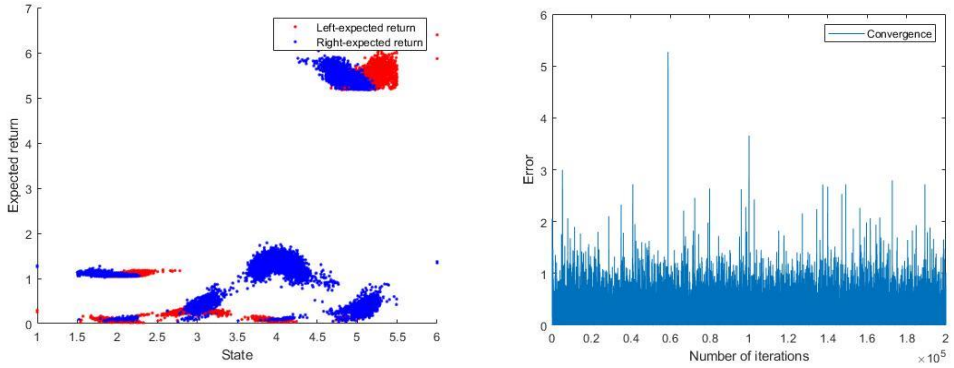

Figure 6.5 Expected return and convergence check for sigma=0.5 with $\alpha = 0.5, \epsilon = 0.05$ and 5 RBF

Exercise 7

In the exercise, SARSA algorithm is implemented, which stands for State-Action-Reward-State-Action. Instead of updating the value function from the maximum estimate of possible next actions, SARSA updates the value function based on and take the same action so information needs to be stored longer before the action values can be updated. The pseudo code of the SARSA control algorithm can be expressed as follows,

Initialize $Q(s, a)$ randomly

Loop for each episode

Initialize state s

Choose action a based on epsilon-greedy policy

Loop

Take action a, observe r, s'

Choose a' from s' based on epsilon-greedy policy

Update the value functions $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma\, Q(s',a') - Q(s,a)]$

$s \leftarrow s';\ a \leftarrow a';$

until reaching the terminal state

The experiments are conducted on the robot problem proposed in Exercise 2. The optimal policies and the convergence check from each algorithm are showed as below,

Q-learning with random initialization, alpha=0.5, epsilon=0.1:

Action	State 1	State 2	State 3	State 4	State 5	State 6
Left	0	1	0.5	0.625	1.25	0
Right	0	0.6243	1.25	2.5	5	0

Optimal policy: $\prod_2^* = Left,\ \prod_3^* = Right,\ \prod_4^* = Right,\ \prod_5^* = Right$

SARSA with random initialization, alpha=0.5, epsilon=0.1:

Action	State 1	State 2	State 3	State 4	State 5	State 6
Left	0	1	0.4878	0.5369	1.1104	0
Right	0	0.5743	1.1759	2.4580	5	0

Optimal policy: $\prod_2^* = Left,\ \prod_3^* = Right,\ \prod_4^* = Right,\ \prod_5^* = Right$

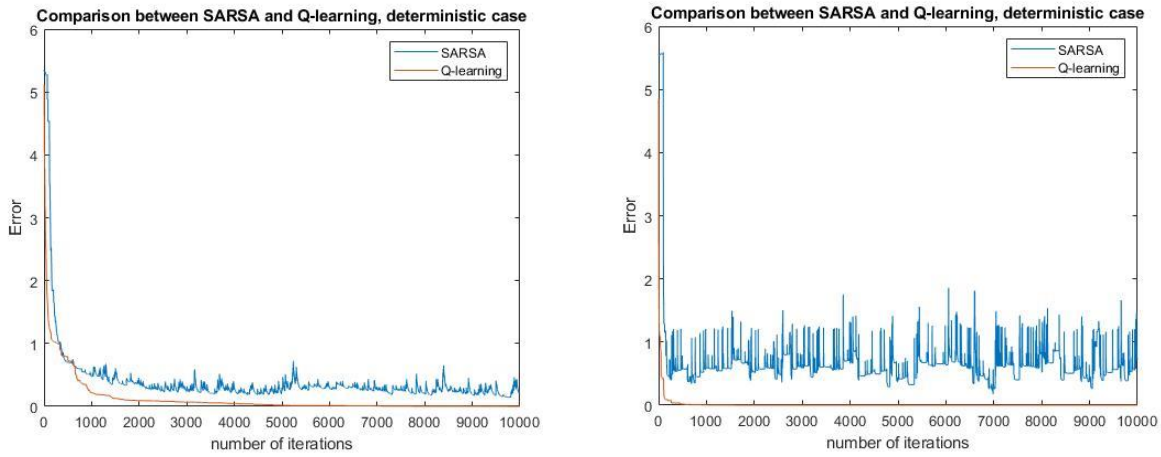

Figure 7.1 Comparison between SARSA and Q-learning with alpha=0.1 and 0.5, respectively

From the observations, it can be found that SARSA converges slower than Q-learning with more oscillations. The optimal Q values from Q-learning are closer to the optimal result given in the beginning. Therefore, some remarks regarding the advantages and disadvantages of SARSA compared to Q-learning can be made:

1) SARSA learns a near-optimal policy while exploring, while Q-learning directly learns the optimal policy
2) SARSA may suffer from convergence problems, while Q-learning has higher per-sample variance
3) SARSA is more conservative, and tends to avoid a dangerous optimal path and only slowly learn to use it