IN4320 Machine Learning Assignment 4

Dilan Gecmen 4221168

May 10, 2018

Note: Worked together with Jasper Hemmes.

Exercise 1

Proof. It is given that

$$-10 \leq r_{t+h+1} \leq 10 \tag{1}$$

$\forall t, h$ where $h \in \{0, 1, \ldots, \infty\}$. Multiply inequality (1) with γ^h. As γ^h $\forall h$ is a positive number this multiplication results in the following inequality

$$-10\gamma^h \leq \gamma^h r_{t+h+1} \leq 10\gamma^h \tag{2}$$

If we take the summation of h from 0 until ∞ in inequality (2) then we still get an inequality that is bounded on both sides,

$$\sum_{h=0}^{\infty} -10\gamma^h \leq \sum_{h=0}^{\infty} \gamma^h r_{t+h+1} \leq \sum_{h=0}^{\infty} 10\gamma^h \tag{3}$$

Note that in inequality (3) we have 3 geometric series. This means that as h goes the infinity, the absolute value of γ must be less than one for the series to converge. As $0 \leq \gamma < 1 \Rightarrow |\gamma| < 1$. Hence, each geometric series converges.

The sum of a geometric series that goes to infinity is equal to $\sum_{k=0}^{\infty} ar^k = \frac{a}{1-r}$ for $|r| < 1$. Using this knowledge we see that the left and right hand side of equation (3) is equal to

$$\frac{-10}{1-\gamma} \leq \sum_{h=0}^{\infty} \gamma^h r_{t+h+1} \leq \frac{10}{1-\gamma} \tag{4}$$

$$\Longleftrightarrow$$

$$\frac{-10}{1-\gamma} \leq R_t \leq \frac{10}{1-\gamma} \tag{5}$$

where $0 \leq \gamma < 1$. Hence, R_t is bounded for $0 \leq \gamma < 1$ and for bounded rewards $-10 \leq r_{t+h+1} \leq 10$ $\forall h \in \{0, 1, \ldots, \infty\}$. □

Exercise 2

Our robot found a new job as a cleaning robot. The robot only has the actions "left" and "right". It is working in a corridor with 6 states, the two end-states are terminal, i.e., the episode ends immediately once the robots reaches them. It gets a reward of 1 when reaching the left state (charger) and a reward of 5 when reaching the right state (trash bin). See Figure 1 for an illustration of what is described.

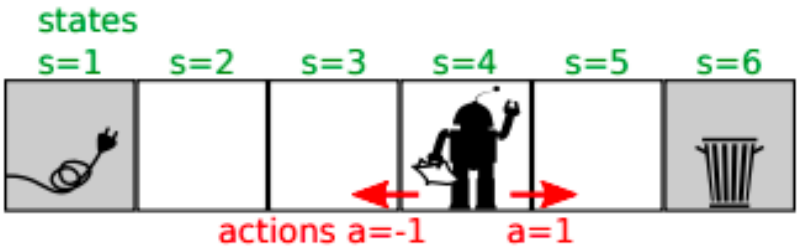

Figure 1

Mathematically we have a set of discrete states $\mathcal{S} = \{1, 2, 3, 4, 5, 6\}$ and a set of discrete actions $\mathcal{A} = \{-1, 1\}$={left, right}. The state at time $t + 1$ is given as

$$s_{t+1} = \begin{cases} s_t & \text{if } s_t \text{ is terminal } (s_t = 1 \text{ or } s_t = 6) \\ s_t + a_t & \text{otherwise, where } a_t \in \mathcal{A} \end{cases}$$

The reward (or return same?) at time $t + 1$ is given as

$$r_{t+1} = \begin{cases} 5 & \text{if } s_{t+1} = 6 \text{ and } s_t \neq 6 \\ 1 & \text{if } s_{t+1} = 1 \text{ and } s_t \neq 1 \\ 0 & \text{otherwise} \end{cases}$$

For $\gamma = 0.5$ the optimal Q-function is given in Table 1.

Table 1: Optimal Q-function for $\gamma = 0.5$

state / action	1	2	3	4	5	6
left	0.0	1.0	0.5	0.625	1.25	0.0
right	0.0	0.625	1.25	2.5	5.0	0.0

Q-iteration is implemented in Matlab we use the pseudo code given in the lecture slides. The values after each iteration is given in the following tables.

Initialize Q

action \ state	1	2	3	4	5	6
left	0	0	0	0	0	0
right	0	0	0	0	0	0

Iteration 1

action \ state	1	2	3	4	5	6
left	0	1	0	0	0	0
right	0	0	0	0	5	0

Iteration 2

action \ state	1	2	3	4	5	6
left	0	1	0.5	0	0	0
right	0	0	0	2.5	5	0

Iteration 3

action \ state	1	2	3	4	5	6
left	0	1	0.5	0.25	1.25	0
right	0	0.25	1.25	2.5	5	0

Iteration 4

action \ state	1	2	3	4	5	6
left	0	1	0.5	0.625	1.25	0
right	0	0.625	1.25	2.5	5	0

For iteration 4 we get the optimal Q-function table. Now using the optimal Q-function table we can determine the optimal policy $\pi^*(s) = \arg\max_{a \in \mathcal{A}}(Q^*(s, a))$. The optimal policy π^* is

- Go left in s_2
- Go right in s_3
- Go right in s_4
- Go right in s_5

Exercise 3

In the following tables the optimal value functions Q^* are given for $\gamma = 0$, $\gamma = 0.1$, $\gamma = 0.9$, and $\gamma = 1$.

Optimal value function Q^* for $\gamma = 0$

action \ state	1	2	3	4	5	6
left	0	1	0	0	0	0
right	0	0	0	0	5	0

Optimal value function Q^* for $\gamma = 0.1$

action \ state	1	2	3	4	5	6
left	0	1	0.1	0.01	0.05	0
right	0	0.01	0.05	0.5	5	0

Optimal value function Q^* for $\gamma = 0.9$

action \ state	1	2	3	4	5	6
left	0	1	3.2805	3.645	4.05	0
right	0	3.645	4.05	4.5	5	0

Optimal value function Q^* for $\gamma = 1$

action \ state	1	2	3	4	5	6
left	0	1	5	5	5	0
right	0	5	5	5	5	0

The discount factor γ is used to model the behavior of the robot, when the robot wants immediate rewards instead of rewards that can be received in the future. Adjusting the value of the discount factor γ we can control which path the robot takes. If the robot wants a quick reward, then γ needs to be low. If γ is high then the robot wants a high valued reward. How many steps it will take to reach an high valued reward is not considered when γ has a high value.
In Exercise 1 an infinite time horizon is given, hence when $\gamma = 1$ the reward goes to infinity. In contrast to Exercise 1, $\gamma = 1$ will work here. The robot stops

when it reaches state 1 or 6. Hence, the value function will not grow to infinity. The value of the Q-function will stop increasing after several iterations.

Exercise 4

We implement Q-learning in Matlab and set $\gamma = 0.5$. The difference between the value function estimated by Q-learning and the true value function is plotted for different values of ϵ and α. Note that we averaged the distance and we took $i = 1000$ iterations to create smooth graphs, see Figure 2.
When we look ate Figure 2 we see that for small values of α and ϵ we get slower convergence compared to high values of α and ϵ. For example, if you look at $\alpha = 0.1$ and $\epsilon = 0.1$, the blue striped line, the convergence is slow. However, if you look at $\alpha = 0.9$ and $\epsilon = 0.9$, the yellow line, convergence is fast.
Set $\alpha = 0.5$. If we variate ϵ we see in Figure 2 that for high value of ϵ convergence is faster, then for low value of ϵ. Set $\epsilon = 0.5$ and variate α. We see in Figure 2 that convergence is faster for high value of α. From variating α and ϵ we can conclude that ϵ has a stronger influence on the convergence then α.
When α gets higher we see in the pseudo code that the estimated Q-value is getting closer to the true value function. For $\alpha = 1$ the update function in the Q-learning algorithm becomes the update function from the Q-iteration algorithm.
If ϵ is high valued then we get exploration, so a random action is picked from $\mathcal{A}$. The robot will explore new actions, if this leads to better discounted rewards in future steps. If $\epsilon = 1$ we have exploration, and the robot always explores and does not exploits. If ϵ is low valued then we get exploitation. It is clear that a high value of ϵ is better for convergence as in our case the robot finds faster a better future reward when it explores instead of exploits.
It is clear that the best values to choose for α and ϵ are high values as the convergence to the true value is faster.

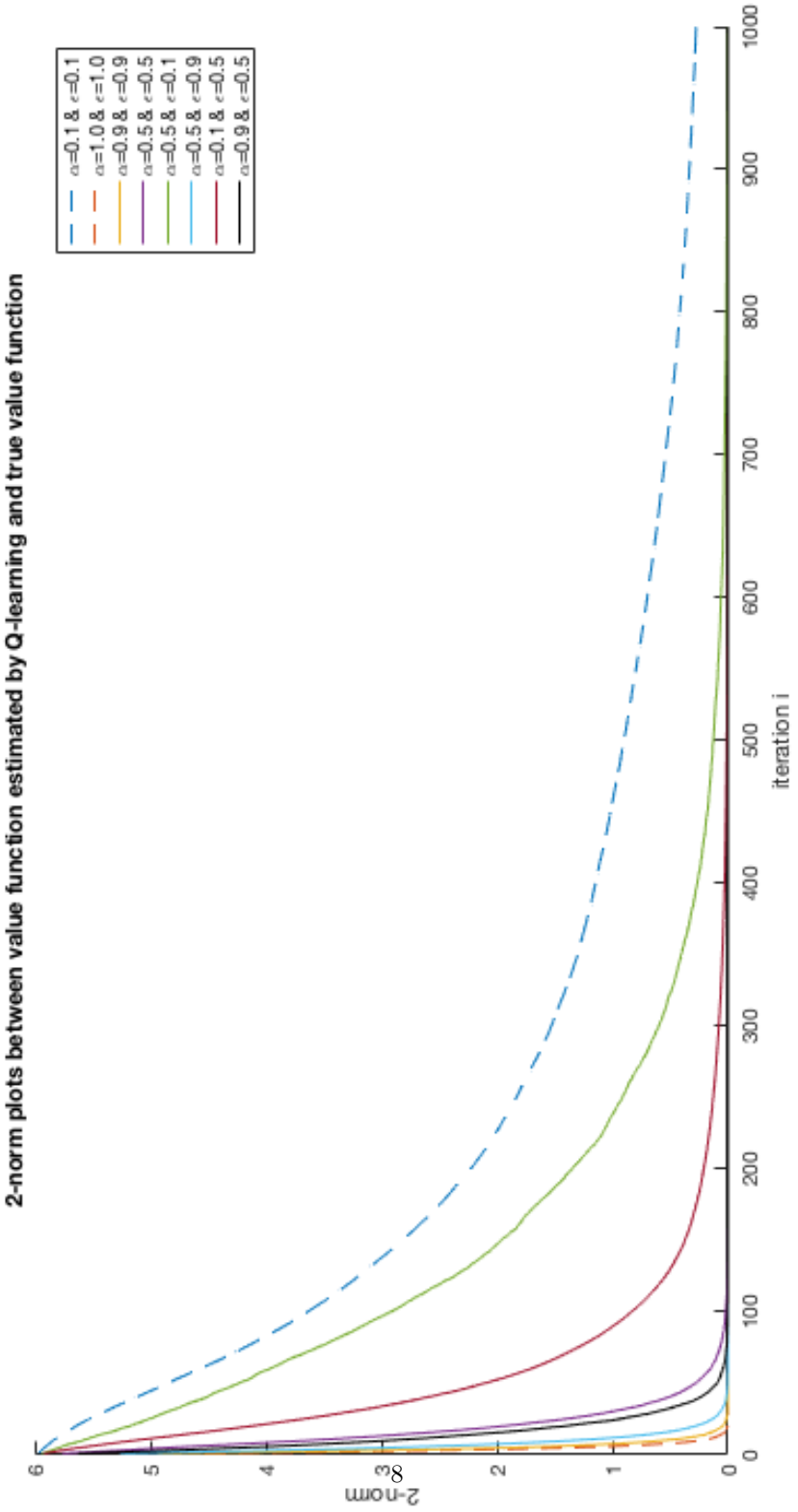

Figure 2: Distance (2-norm) between value function estimated by Q-learning and true value function for different values of ϵ and α. The plots show averaged convergence.

Exercise 5

Now the robot is partially broken and stays at the same state with a probability of 30%, else it works correctly. The values for Q-learning and Q-iteration for this scenario are given in Tables 2 and 3. In Table 3 we can see that the values of Q-iteration are different from the values in the optimal Q-function table, because there is a possibility that the robot does not take an action, which influences future rewards.
For Q-learning we need to take a small learning rate $\alpha = 0.0001$, $\epsilon = 0.5$, and a lot of iterations. Note that taking small α does not matter for convergence, as we took a lot of iterations. Taking a small learning rate we get values close to Q-iteration.

Table 2: Q-learning for partially broken robot.

state / action	1	2	3	4	5	6
left	0	0.8230	0.3930	0.5000	1.2111	0.0
right	0	0.3670	0.6981	1.6960	4.1185	0.0

Table 3: Q-iteration for partially broken robot.

state / action	1	2	3	4	5	6
left	0	0.8235	0.3930	0.4987	1.2111	0.0
right	0	0.3679	0.6981	1.6955	4.1176	0.0

Exercise 6

We use literature [1] [2] to make changes to the Q-learning algorithm, so that it works with Radial Basis Functions.

Pseudo-code

- First initialize θ. For each iteration i apply the following steps.
  - Initialize the Q, s and repeat the following steps for each move the robot does until convergence.
    - $a \leftarrow \arg\max_{a \in \mathcal{A}}[Q(s,a)]$
    - With probability $\epsilon, \alpha \leftarrow \text{rand}(\mathcal{A})$.
    - Apply a, observe r and s'.
    - $\overrightarrow{\theta}(a) \leftarrow \overrightarrow{\theta}(a) + \alpha \nabla_{\overrightarrow{\theta}(a)} Q(s,a)(r + \gamma \arg\max_{a' \in \mathcal{A}}[Q(s',a') - Q(s,a)]$
    - $s = s'$
- Repeat the previous steps until $\overrightarrow{\theta}$ is below a certain value.

Note that

- $Q(a,s) = \sum_{i=1}^{k} \theta_i(a)\varphi_i(s)$
- $\varphi_i(s) = \exp(\frac{||s-c_i||}{2\omega_i})$
- $\overrightarrow{\theta}(a) = [\overrightarrow{\theta}(\text{left}), \overrightarrow{\theta}(\text{right})]$

The weight vectors $\overrightarrow{\theta}(a)$ correspond to the value functions for moving left and right. Implementing our pseudo-code in Matlab we saw that the initialization of the weight vectors determine their end value.
We have found local minimum, which can be seen in Figure 3. Note that the motions in the graphs looks a bit similar to the values given in the optimal Q iteration table in exercise 2.
However, we did not find a global minimum. Trying different values for the parameters, playing with the amount of RBFs, etc, did not work.
If the width of the RBFs is too wide then this can lead to instability in the value function, as generalization increases. Hence, the width of RBFs influence the performance!

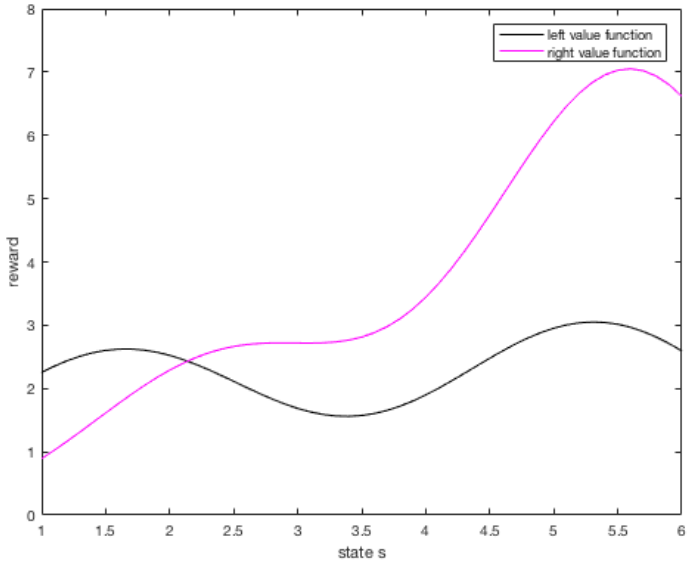

Figure 3: The left and right value function plotted.

Exercise 7

Policy iteration pseudo-code

- choose an arbitrary policy π'
- loop
  - $\pi := \pi'$
  - compute the value function of policy π:
    - Solve the linear equations
      - $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$
  - improve the policy at each state:
    - $\pi'(s) = \arg\max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s'))$

  until $\pi = \pi'$

Q-iteration is simpler then policy iteration, but it is computationally much more heavy. On the other hand, policy iteration is complicated, but it is cheaper computationally then value iteration.

References

[1] V. Heidrich-Meisner, M. Lauer, C. Igel and M. Riedmiller (2007). *Reinforcement Learning in a Nutshell*. In 15th European Symposium on Artificial Neural Networks (ESANN).

[2] Approximate Q-Learning, `https://www.cs.swarthmore.edu/ bryce/cs63/s16/slides/3-25_approximate_Q-learning.pdf`