IN4320 Machine Learning: Assignment 4
Reinforcement Learning

Georgios Dimitropoulos: 4727657

May 10, 2018

Exercise 01

The return is given by the formula $R_t = \sum_{h=0}^{\infty} \gamma^h r_{t+h+1}$. It is also given that $0 \leq \gamma < 1$ and $-10 \leq r_{t+h+1} \leq 10, \forall h \in \{0,1,...,\infty\}$. In order to prove that this return is bounded, it suffices to show that as h approaches infinity, the above expression is bounded, that it is a finite number (it does not go to infinity). We have that, $R_t = \sum_{h=0}^{\infty} \gamma^h r_{t+h+1}$ and we take the limit of this expression as h becomes arbitrary large. More formally, we have $\lim_{h\to\infty} R_t = \lim_{h\to\infty} \sum_{h=0}^{\infty} \gamma^h r_{t+h+1}$. We notice here that this expression consists of two terms. Namely the r_{t+h+1} and the γ^h. However, it is given above that the term r_{t+h+1} is bounded for any value of h, as we have that $-10 \leq r_{t+h+1} \leq 10, \forall h \in \{0,1,...,\infty\}$. Also, it is given that $0 \leq \gamma < 1$. Thus, the $\lim_{h\to\infty} \gamma^h = 0$ as this is an exponential function and its base is between 0 and 1. Hence, the whole expression, $\sum_{h=0}^{\infty} \gamma^h r_{t+h+1}$ is going to become zero as h approaches infinity and for this reason the return is bounded.

Exercise 02

In this question we implement the Q-Iteration algorithm. We iterate this algorithm for $\gamma = 0.5$ until we converge to the optimal Q function, Q^*. As it can be depicted from table 1, we have convergence in the 4th iteration where the Q-values are equal to the optimal Q-function. Thus, based on the formula $\pi^*(s) = argmax_{a\in A}(Q^*(s,a))$, we have that the optimal policy $\pi^*(s)$ for each state is:
State 1: Terminal state
State 2: Left
State 3: Right
State 4: Right
State 5: Right
State 6: Terminal state

Table 1: Q values for each iteration

ITERATION	S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0	0\0	0\0	0\0	0\0	0\0	0\0
1	0\0	1\0	0\0	0\0	0\5	0\0
2	0\0	1\0	0.5\0	0\2.5	0\5	0\0
3	0\0	1\0.25	0.5\1.25	0.25\2.5	1.25\5	0\0
4	0\0	1\0.625	0.5\1.25	0.625\2.5	1.25\5	0\0

Exercise 03

In this question we again implement the Q-Iteration algorithm for various values of γ. We repeat this algorithm until we have two successive identical iterations.

$\boxed{\gamma = 0}$

In table 2 the Q values for each iteration for $\gamma = 0$ can be depicted.

Table 2: Q values for each iteration-$\gamma = 0$

ITERATION	S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0	0\0	0\0	0\0	0\0	0\0	0\0
1	0\0	1\0	0\0	0\0	0\5	0\0
2	0\0	1\0	0\0	0\0	0\5	0\0

The optimal value function Q^* for $\gamma = 0$ is:

S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0\0	1\0	0\0	0\0	0\5	0\0

$\boxed{\gamma = 1}$

In table 3 the Q values for each iteration for $\gamma = 1$ can be depicted.

Table 3: Q values for each iteration-$\gamma = 1$

ITERATION	S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0	0\0	0\0	0\0	0\0	0\0	0\0
1	0\0	1\0	0\0	0\0	0\5	0\0
2	0\0	1\0	1\0	0\5	0\5	0\0
3	0\0	1\1	1\5	1\5	5\5	0\0
4	0\0	1\5	1\5	5\5	5\5	0\0
5	0\0	1\5	5\5	5\5	5\5	0\0
6	0\0	1\5	5\5	5\5	5\5	0\0

The optimal value function Q^* for $\gamma = 1$ is:

S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0\0	1\5	5\5	5\5	5\5	0\0

$\boxed{\gamma = 0.9}$

In table 4 the Q values for each iteration for $\gamma = 0.9$ can be depicted.

Table 4: Q values for each iteration-$\gamma = 0.9$

ITERATION	S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0	0\0	0\0	0\0	0\0	0\0	0\0
1	0\0	1\0	0\0	0\0	0\5	0\0
2	0\0	1\0	0.9\0	0\4.5	0\5	0\0
3	0\0	1\0.81	0.9\4.05	0.81\4.5	4.05\5	0\0
4	0\0	1\3.645	0.9\4.05	3.645\4.5	4.05\5	0\0
5	0\0	1\3.645	3.2805\4.05	3.645\4.5	4.05\5	0\0
6	0\0	1\3.645	3.2805\4.05	3.645\4.5	4.05\5	0\0

The optimal value function Q^* for $\gamma = 0.9$ is:

S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0\0	1\3.645	3.2805\4.05	3.645\4.5	4.05\5	0\0

$\boxed{\gamma = 0.1}$

In table 5 the Q values for each iteration for $\gamma = 0.1$ can be depicted.

Table 5: Q values for each iteration-$\gamma = 0.1$

ITERATION	S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0	0\0	0\0	0\0	0\0	0\0	0\0
1	0\0	1\0	0\0	0\0	0\5	0\0
2	0\0	1\0	0.1\0	0\0.5	0\5	0\0
3	0\0	1\0.01	0.1\0.05	0.01\0.5	0.05\5	0\0
4	0\0	1\0.01	0.1\0.05	0.01\0.5	0.05\5	0\0

The optimal value function Q^* for $\gamma = 0.1$ is:

S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0\0	1\0.01	0.1\0.05	0.01\0.5	0.05\5	0\0

Influence of the discount factor γ

The discount factor γ determines how strongly immediate rewards are weighted in comparison to rewards in the future and thus is about how fast we desire the reward. If γ is closer to zero, and thus for low values of γ, the agent tends to consider only immediate rewards. In comparison, in case that γ is closer to one, and thus for high values of γ, the agent considers future rewards with greater weight, as it is willing to delay in order to take a long-term high reward.

Reason that γ works here in contrast to Exercise 1

Here we have that the states 1 and 6 are terminal states. Hence, in case that the agent reaches these states the episode terminates and the result of the value function is returned. Thus, even if $\gamma = 1$, there is a point where the value function stops growing throughout the infinity.

Exercise 04

In figures 1-10 the difference (2-norm) between the value function estimated by Q-learning and the true value function that is given in the Exercise 02 over the number of interactions and for $\gamma = 0.5$ can be depicted . Particularly, in figures 1-5, the exploration ϵ is constant and equals 0.1 and different values of the learning rate α are examined. More specifically, we examined the cases where $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Also, in figures 6-10, the learning rate α is constant and equals 0.9 and different values of the exploration ϵ are examined. More specifically, we examined the cases where $\epsilon \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. We decided to provide a different plot for each separate value of the variable parameter (i.e α or ϵ) as there was a big overlap in a common plot and it was difficult to distinguish the differences. For all cases the average of 100 iterations was used in order to obtain more representative results. The final optimal policy $\pi^*(s)$ that is returned is the same as the ground truth.

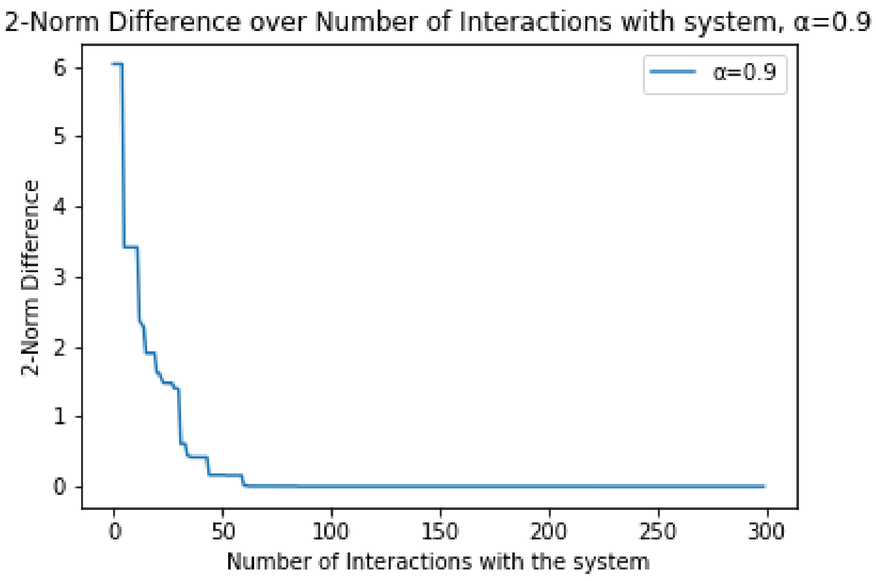

Figure 1: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.9$

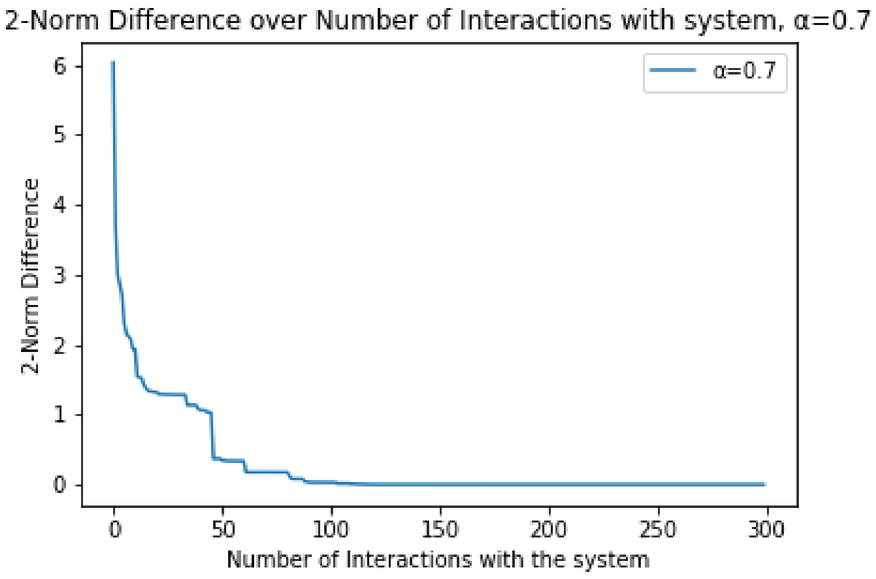

Figure 2: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.7$

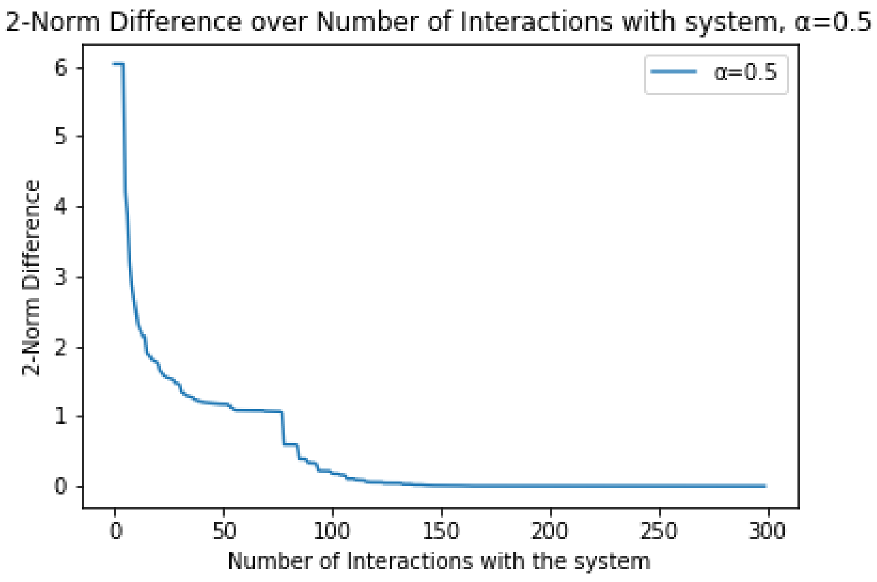

Figure 3: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.5$

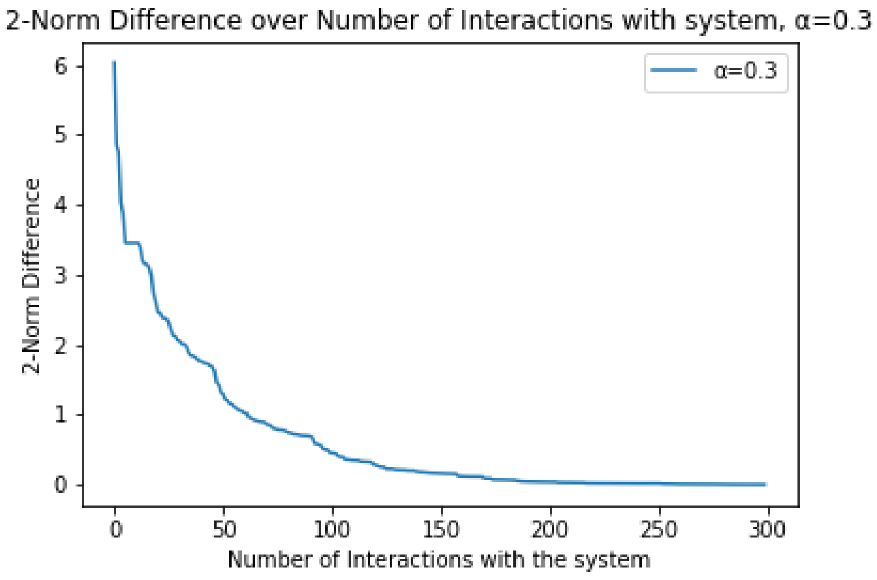

Figure 4: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.3$

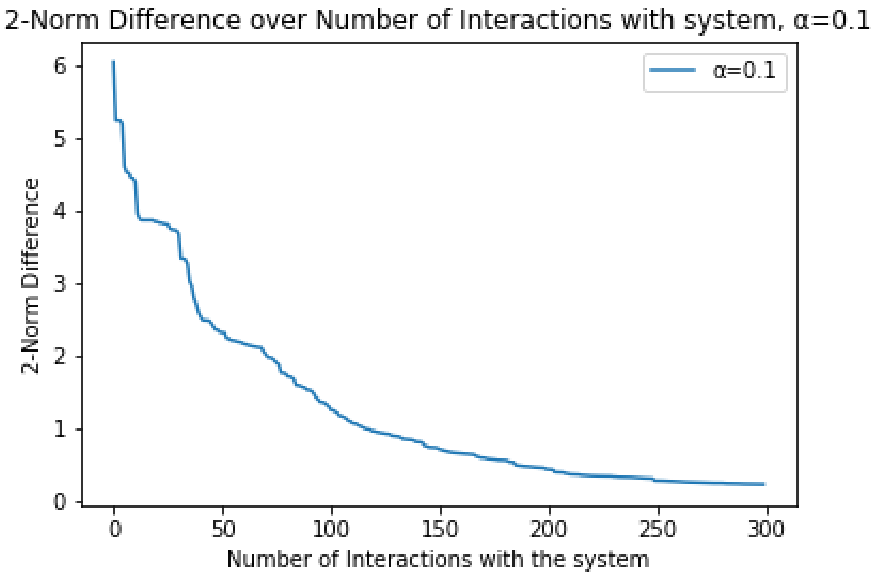

Figure 5: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.1$

Behavior of the learning rate α

As it can be observed by the figures 1-5, the higher the value of the learning rate α is, then the faster our algorithm converges. In contrast, if the learning rate α has a low value then our algorithm converges more slowly. More specifically, we can depict: from figure 1, that when $\alpha = 0.9$, then the algorithm converges (approximately) in 60 interactions with the system, from figure 2, that when $\alpha = 0.7$, then the algorithm converges (approximately) in 100 interactions with the system, from figure 3, that when $\alpha = 0.5$, then the algorithm converges (approximately) in 130 interactions with the system, from figure 4, that when $\alpha = 0.3$, then the algorithm converges (approximately) in 180 interactions with the system and finally from figure 5, that when $\alpha = 0.1$, then the algorithm converges (approximately) in 250 interactions with the system.

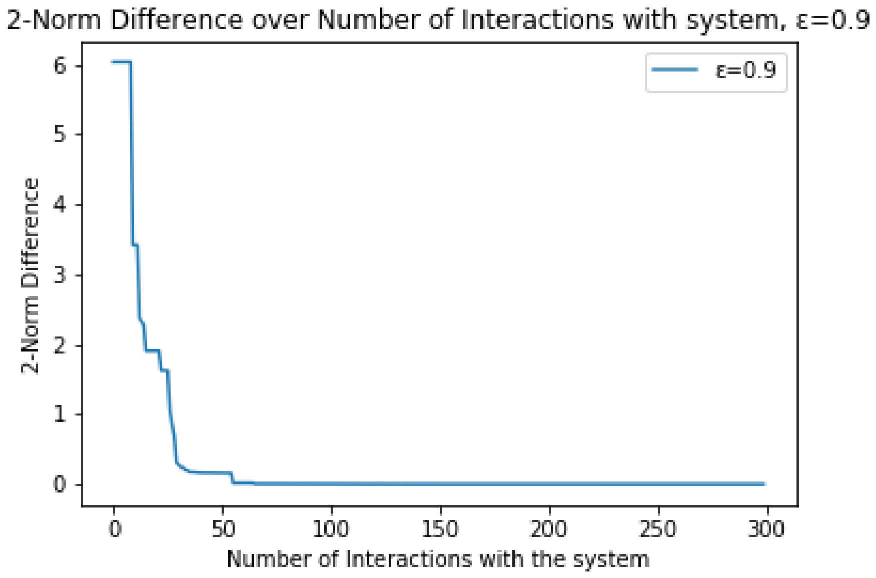

Figure 6: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.9$

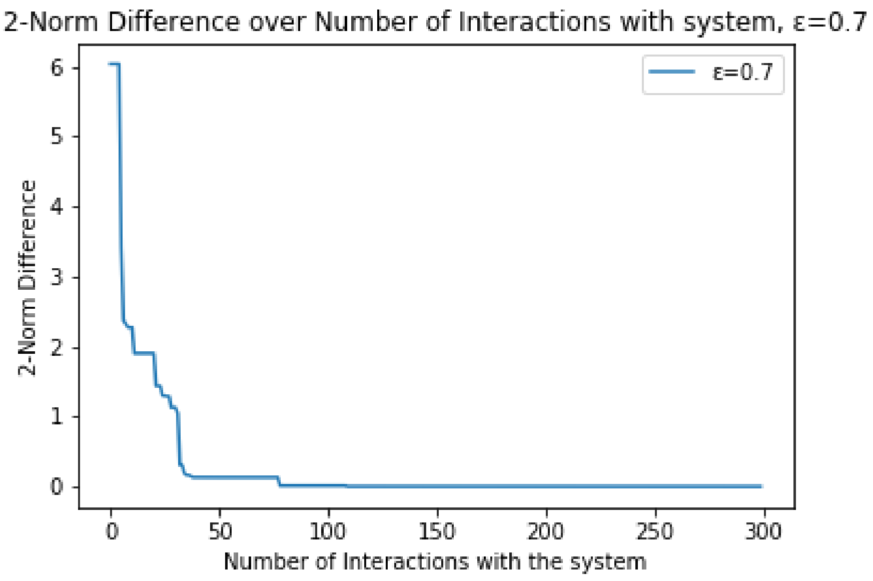

Figure 7: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.7$

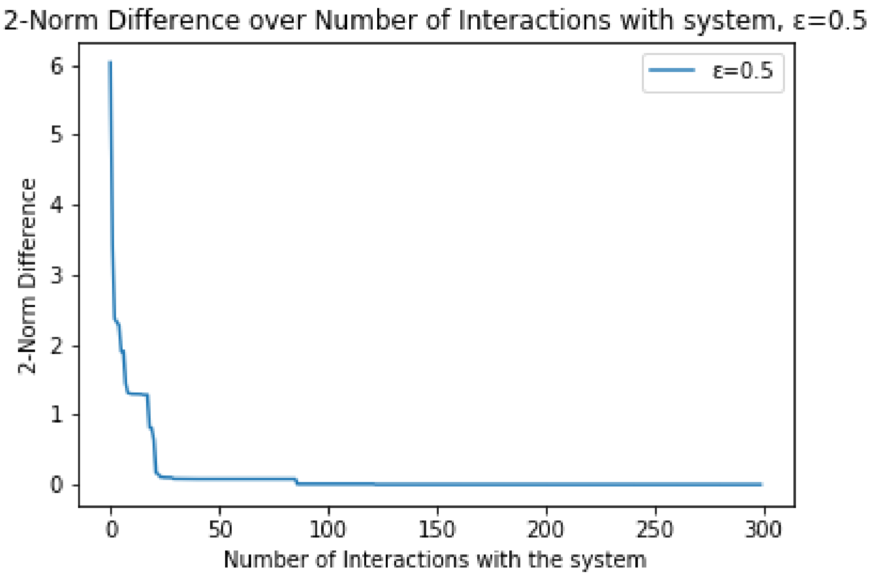

Figure 8: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.5$

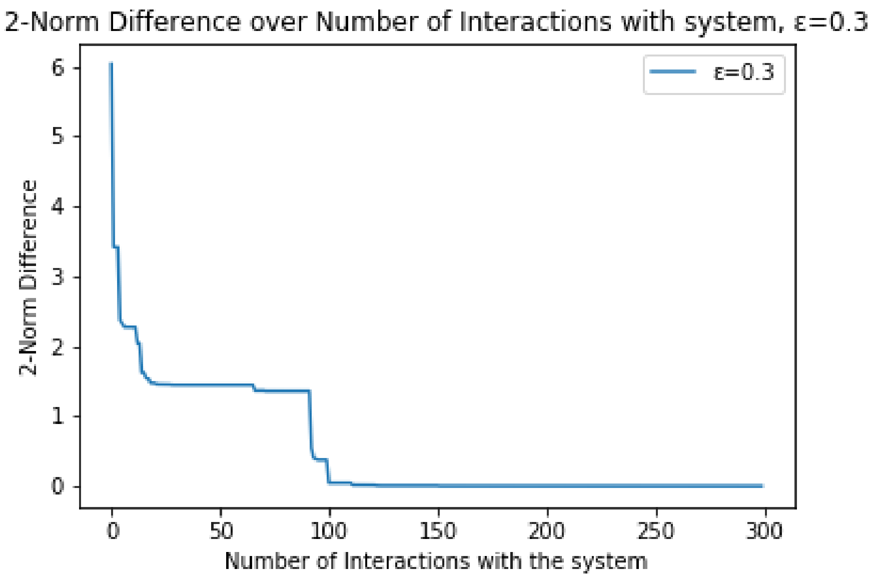

Figure 9: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.3$

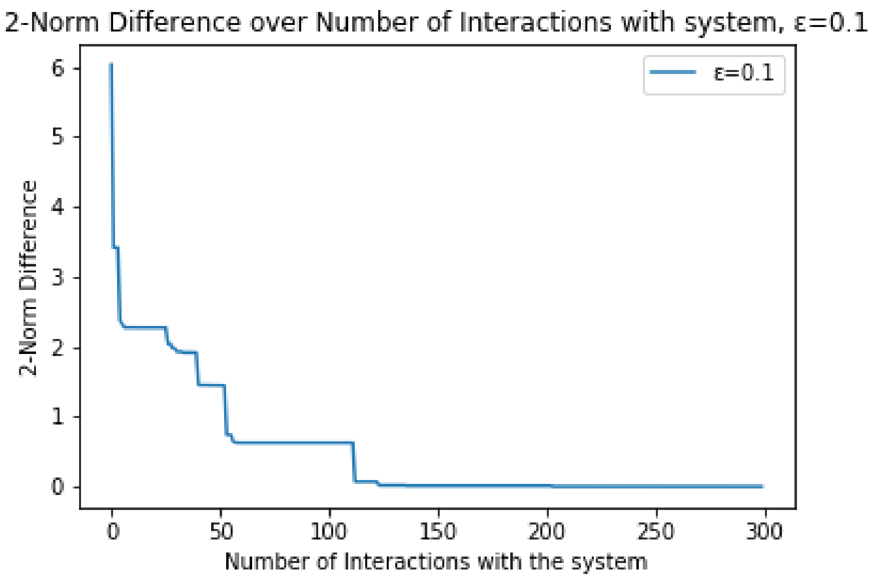

Figure 10: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.1$

Behavior of the learning rate ϵ

In a similar manner, as it can be observed by the figures 6-10, the higher the value of the learning rate ϵ is, then the faster our algorithm converges. In contrast, if the learning rate ϵ has a low value then our algorithm converges more slowly. More specifically, we can depict: from figure 6, that when $\epsilon = 0.9$, then the algorithm converges (approximately) in 55 interactions with the system, from figure 7, that when $\epsilon = 0.7$, then the algorithm converges (approximately) in 75 interactions with the system, from figure 8, that when $\epsilon = 0.5$, then the algorithm converges (approximately) in 90 interactions with the system, from figure 9, that when $\epsilon = 0.3$, then the algorithm converges (approximately) in 110 interactions with the system and finally from figure 10, that when $\epsilon = 0.1$, then the algorithm converges (approximately) in 125 interactions with the system.

Exercise 05

In this exercise we examine the case where our robot is partially broken and stays at the same state with a probability of 30% and else is works correctly. We are going to test this scenario with both Q-iteration and Q-learning.

Q-iteration

In table 6 the Q values for each iteration for $\gamma = 0.5$ can be depicted.

Table 6: Q values for each iteration (partially broken robot) - $\gamma = 0.5$

ITERATION	S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0	0\0	0\0	0\0	0\0	0\0	0\0
1	0\0	0.7\0	0\0	0\0	0/3.5	0\0
2	0\0	0.805\0.105	0.245\0	0\1.225	0.525\0.4025	0\0
3	0\0	0.82075\0.2065	0.3185\0.4655	0.2695\1.5925	1.0325\4.10375	0\0
4	0\0	0.8231\0.2860	0.3570\0.6272	0.4018\0.6751	0.1729\4.1155	0\0
5	0\0	0.8234\0.3586	0.3821\0.6803	0.4864\1.6916	1.2036\4.1173	0\0
6	0\0	0.823\0.362	0.39\0.694	0.492\1.695	1.21\4.117	0\0
7	0\0	0.823\0.366	0.392\0.697	0.497\1.695	1.211\4.117	0\0
8	0\0	0.823\0.367	0.393\0.698	0.498\1.695	1.211\4.117	0\0
9	0\0	0.823\0.367	0.393\0.698	0.498\1.695	1.211\4.117	0\0

The optimal value function Q^* with Q-iteration for $\gamma = 0.5$ in case that our robot is partially broken and stays at the same state with a probability of 30% and else is works correctly is:

S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0\0	0.823\0.367	0.393\0.698	0.498\1.695	1.211\4.117	0\0

In this case, the values of the optimal Q^* function are not as "absolute" as were in case that the robot was not broken (i.e. Exercise 2). For instance, Q^*(5,right) was 5 in the Exercise 2 but now equals to 4.117. Of course, since we have now transition probabilities (the robot is partially broken), obtaining a high reward if the state is 5 and the action is right is now probable only 70% in comparison to 100% that was the case in the Exercise 2. Similarly, this is also the case in all other situations and this is the reason that the values of the optimal Q^* function are not as "absolute" now and this makes the difference in comparison to Exercise 02.

Q-learning

In figures 11-20 the difference (2-norm) between the value function estimated by Q-learning and the true value function that is given in the Exercise 02 for the same values of α and ϵ as in Exercise 04 over the number of interactions and for $\gamma = 0.5$ in the case where our robot is partially broken and stays at the same state with a probability of 30% and else is works correctly can be depicted. Again, we decided to provide a different plot for each separate value of the variable parameter (i.e α or ϵ) as there was a big overlap in a common plot and it was difficult to distinguish the differences.Also, for all cases the average of 100 iterations was used in order to obtain more representative results. In case of Q-learning our algorithm never converges for the values of α and ϵ which we used before. Thus, our algorithm does not return an optimal value function.

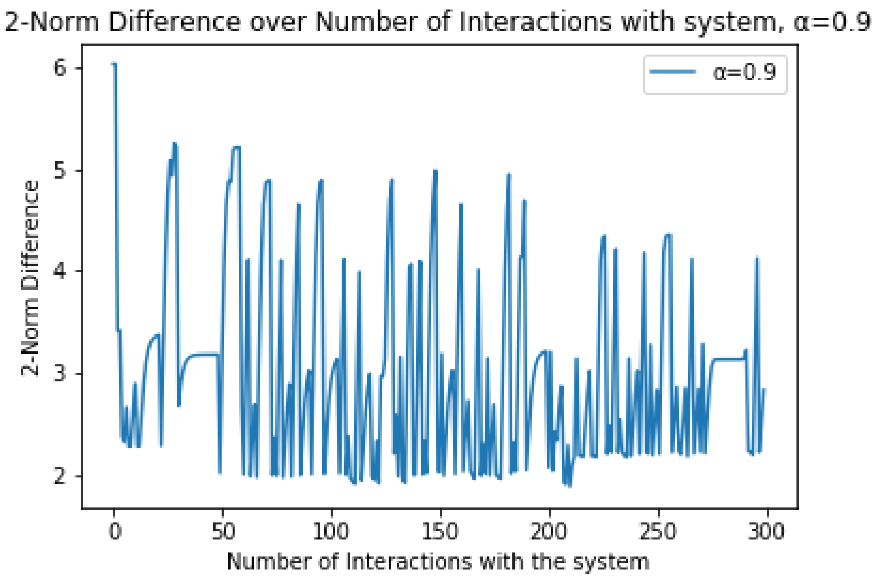

Figure 11: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.9$

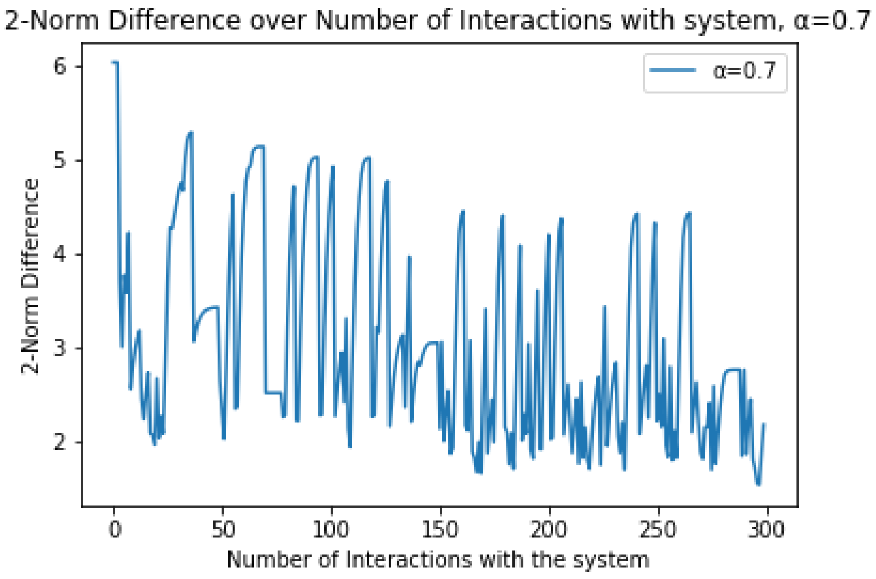

Figure 12: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.7$

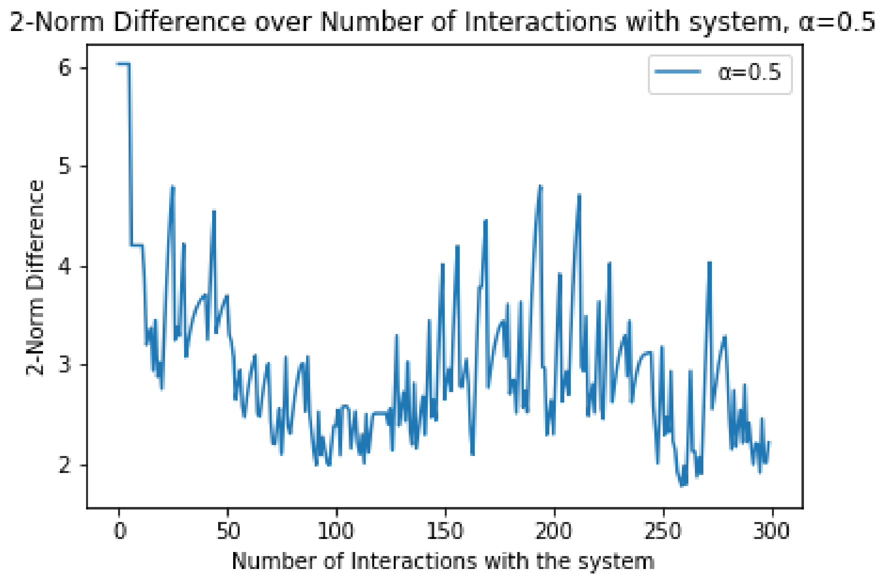

Figure 13: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.5$

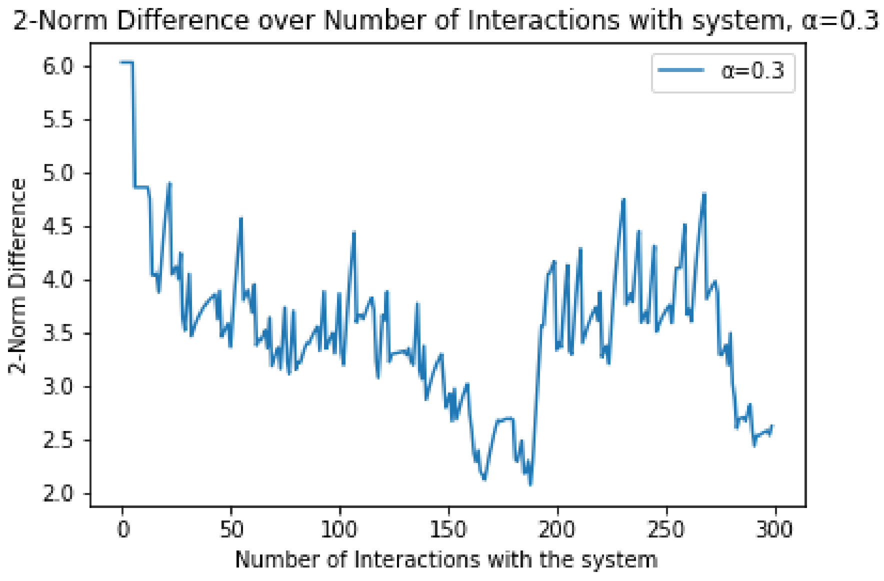

Figure 14: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.3$

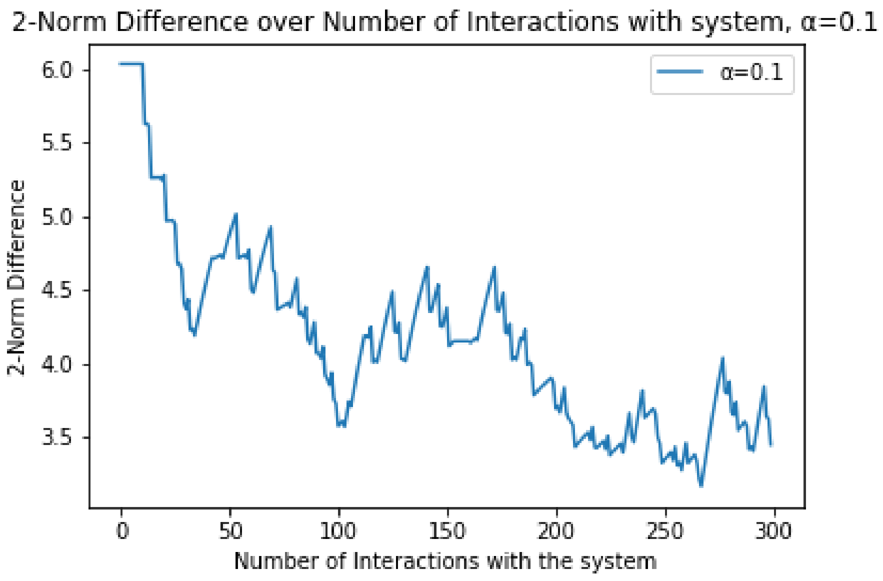

Figure 15: 2-Norm Difference over Number of Interactions with system, $\alpha = 0.1$

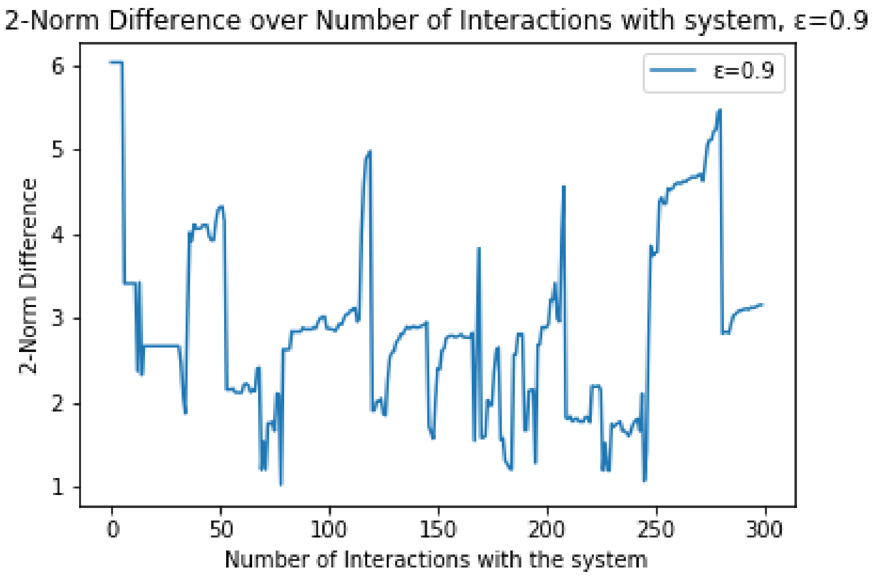

Figure 16: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.9$

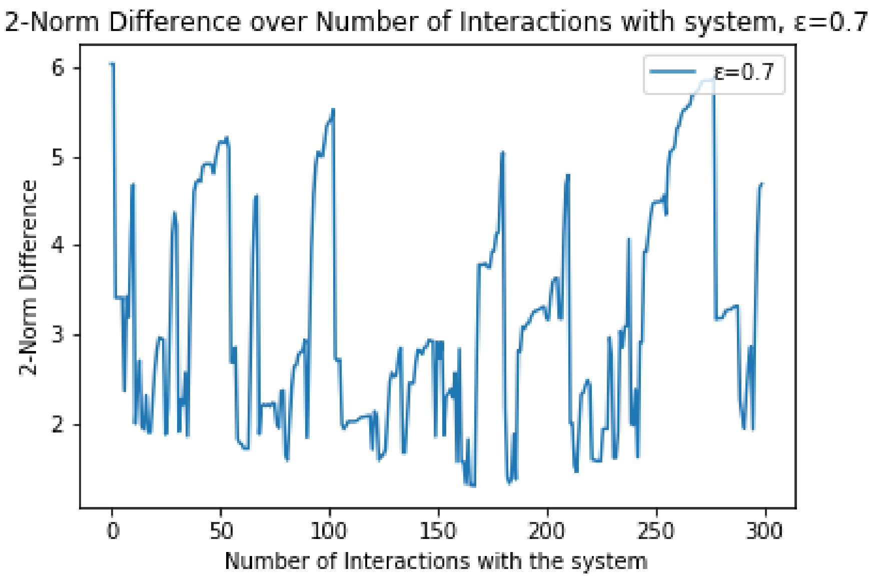

Figure 17: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.7$

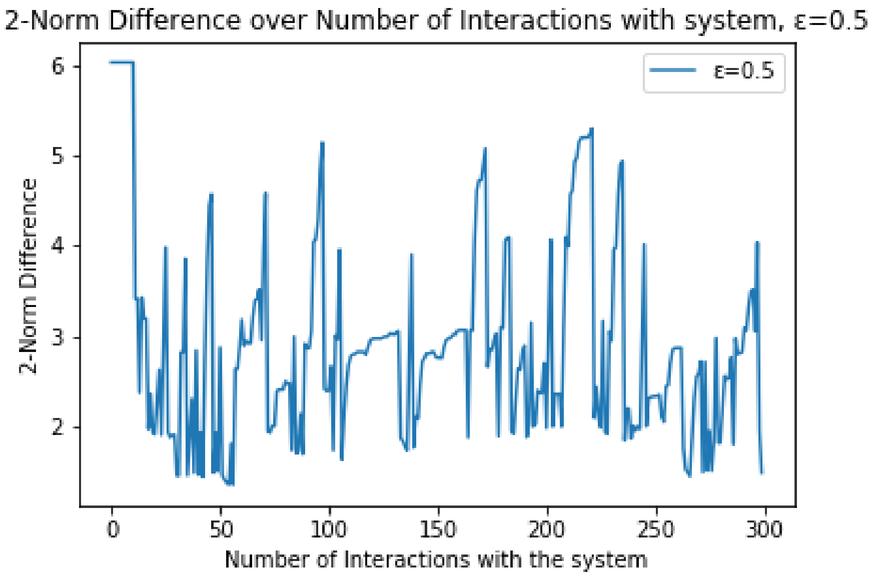

Figure 18: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.5$

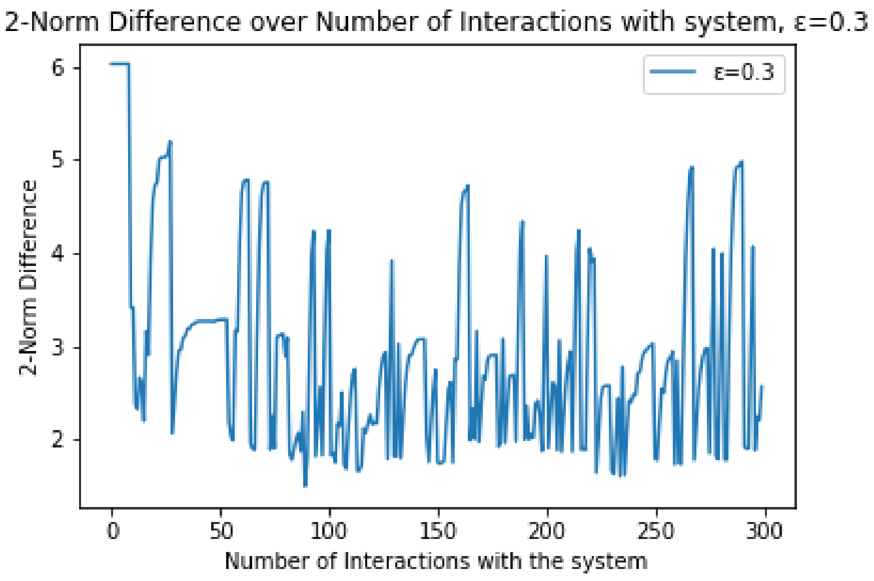

Figure 19: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.3$

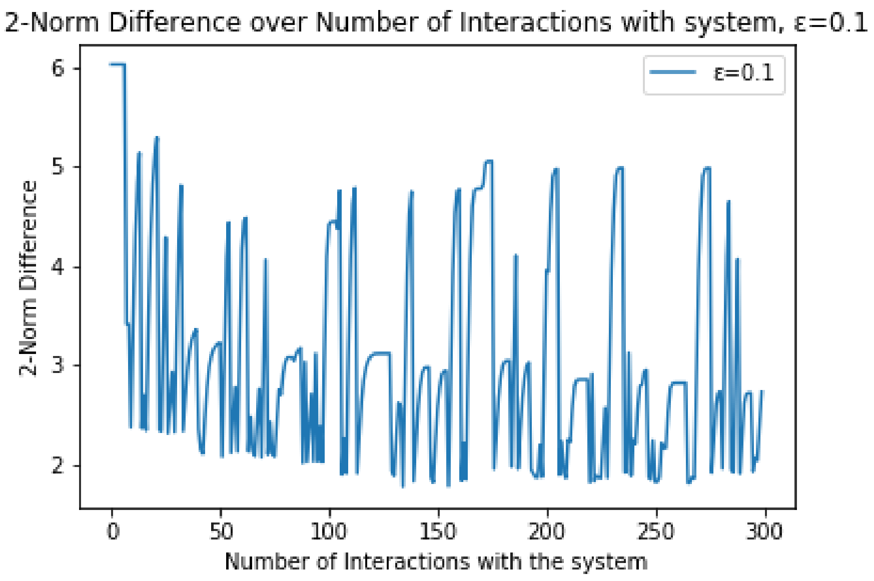

Figure 20: 2-Norm Difference over Number of Interactions with system, $\epsilon = 0.1$

More specifically, as we could see in figures 11-15, in general, for lower values of α the difference between the value function estimated by Q-learning and the ground truth is higher and also there are a lot of fluctuations. Moreover, as it can be observed from figures 16-20 ϵ does not seem to influence in a significant manner the Q values. Finally, from all figures 11-20, we could see that we cannot still use the same Q-learning parameters as our algorithm does not converge. The main reason that the aforementioned values which we used in Exercise 04 do not lead to convergence here is that the "high" learning rate has as a consequence higher weight in the new values. Hence, this leads to bigger changes in the value function in case that an action succeeds in one moment but the other moment this is not the case.

Exercise 06

In order to be able to implement the value function approximation with radial-basis functions, we based on [1]. More specifically, we use a Sarsa(λ) with RBF-features and greedy policy algorithm. The pseudo-code which we employ is as follows:

```
01)Initialize vector $\vec{\theta}$ arbitrarily
02)**Repeat** (for each episode)
03)Initialize vector $\vec{e}$ as a zero vector
04)s,a$\leftarrow$ initial state and action of episode
05)$F_s \leftarrow$ set of features present in s
06)**Repeat** (for each step of episode)
07)**For all** $i \in F_s$:
08)$\vec{e_a}(i) \leftarrow \vec{e_a}(i) + \phi_s(i)$
09)Take action a, observe reward r and next state s'
10)$\delta \leftarrow r - \sum_{i \in F_s} \theta_a(i)\phi_s(i)$
11)With probability $1 - \epsilon$:
12)**For all** $a \in A(s')$:
13)$F_s' \leftarrow$ set of features present in s'
14)$Q_a \leftarrow \sum_{i \in F_s} \theta_a(i)\phi_{s'}(i)$
15)$a' \leftarrow argmax_a Q_a$
16)**else**:
17)$a' \leftarrow a$ random action $\in A(s)$
18)$F_s' \leftarrow$ set of features present in $s'$
19)$Q_a' \leftarrow \sum_{i \in F_s} \theta_{a'}(i)\phi_{s'}(i)$
20)$\delta \leftarrow \delta + \gamma Q_{a'}$
21)$\vec{\theta} \leftarrow \vec{\theta} + a\delta\vec{e}$
22)$\vec{e} \leftarrow \gamma\lambda\vec{e}$
23)$s \leftarrow s'$
24)$a \leftarrow a'$
25)**until** s is terminal
```

In our implementation, we used 6 basis functions centered at $\{1,2,3,4,5,6\}$ and we normalized our RBF network. In figures 21-23, the Q-function which was obtained with width w=0.1, w=0.5 and w=0.9 respectively can be depicted. As far as the influence that the widths have on the performance, based on these figures, we could say that in case that the RBF is too wide (i.e. larger value of w), there is a high degree of generalization. Thus, this may cause instability in the value function. For instance, in case that w=0.1 the value function seems to be more stable (i.e. almost parallel in the x-axis in many points) in contrast to the other cases and mostly to the case that w=0.9 where the value function is not parallel at all with the x-axis.

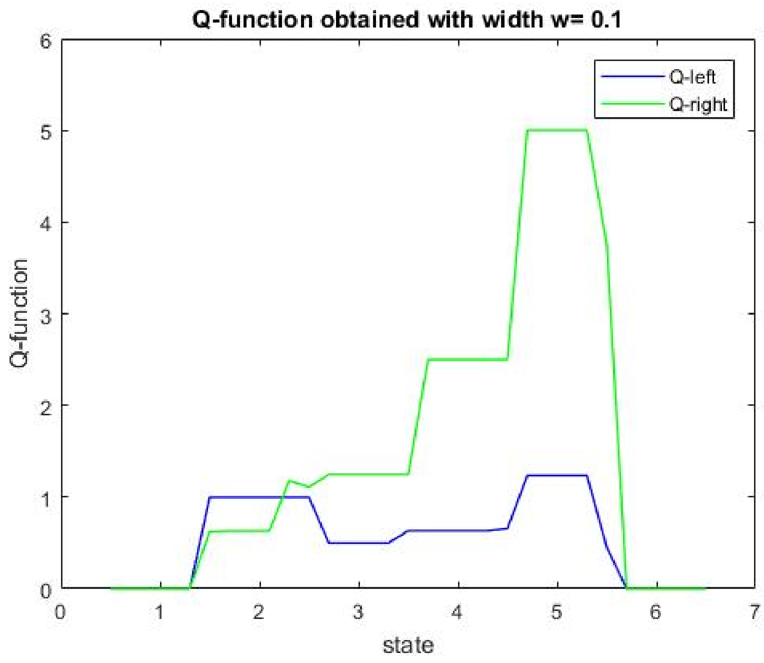

Figure 21: Q-function obtained with width $w = 0.1$

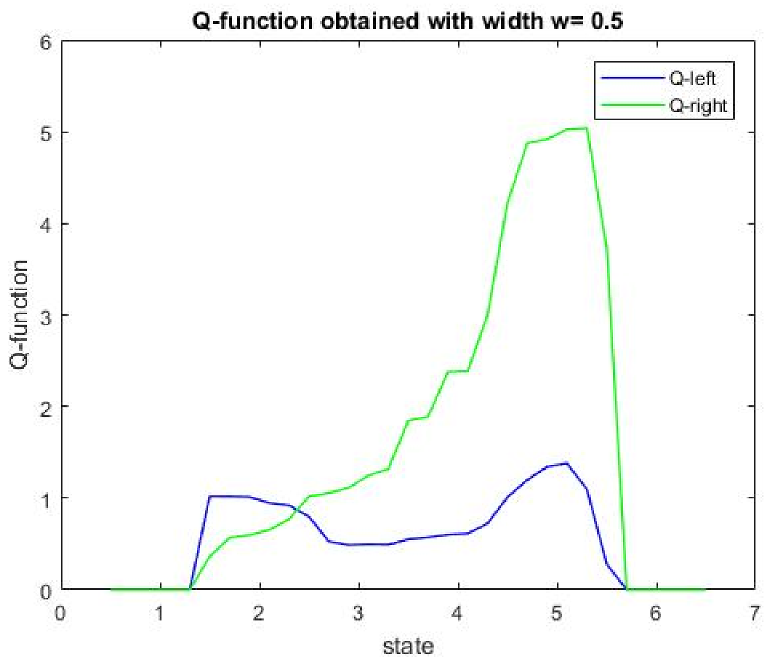

Figure 22: Q-function obtained with width $w = 0.5$

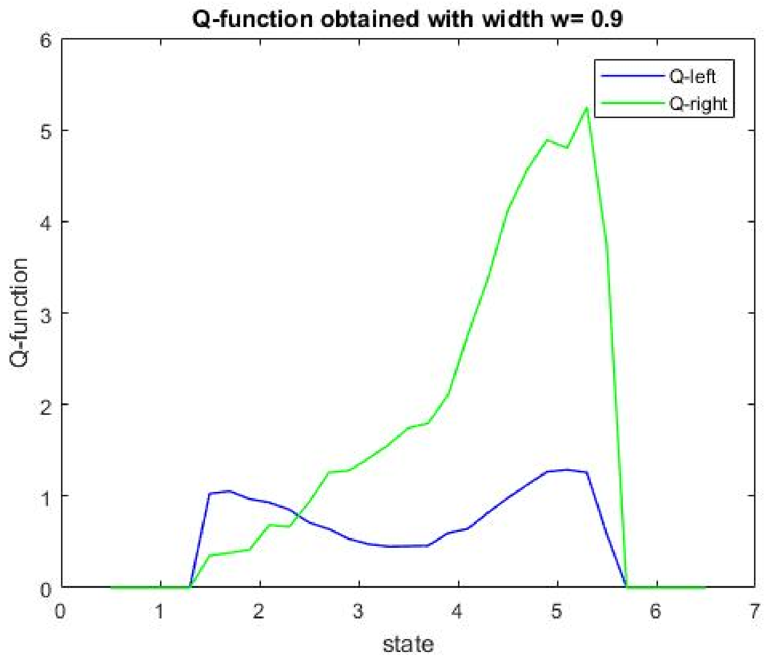

Figure 23: Q-function obtained with width $w = 0.9$

Finally, in figure 24, the 2-Norm Difference over Number of Interactions with system of this algorithm is presented. As far as the learning performance of this algorithm compared to the other approaches which we implemented previously is being concerned, based on this figure, we could say that it requires more interactions with the system in order to converge (approximately 2000). Furthermore, as it can be depicted in this figure it does not converge perfect but with a precision of 3 decimals and also after that convergence, it starts to fluctuate again.

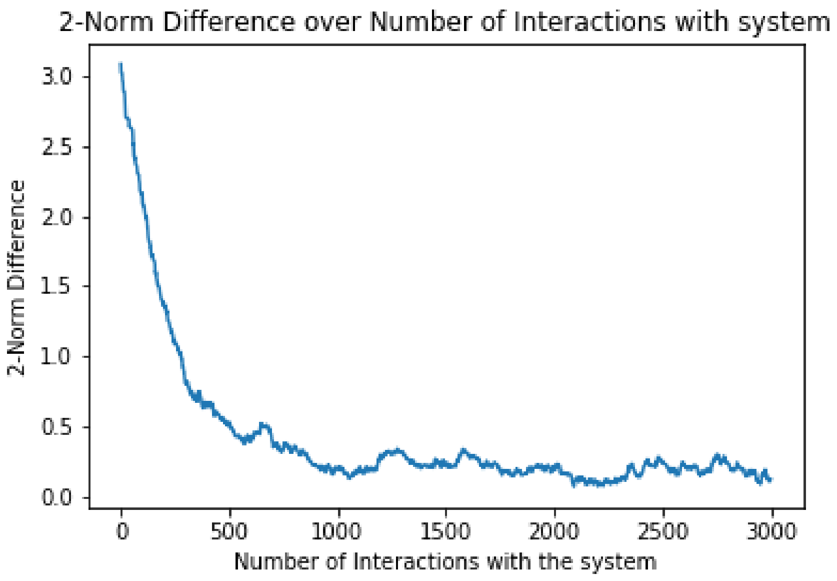

Figure 24: 2-Norm Difference over Number of Interactions with system

Exercise 07

The RL algorithm which was implemented in this exercise is the SARSA algorithm [2]. This algorithm employs on-policy learning in contrast to Q-learning which uses off-policy learning. Hence, a comparison between these algorithms is very interesting.
The pseudocode of the SARSA algorithm is as follows:

```
01)Initialize $Q(s, a)$ arbitrarily
02)**Repeat** (for each episode)
03)Initialize $s$
04)Choose $\alpha$ form $s$ using policy derived from $Q$ (i.e. $\epsilon - greedy$)
05)**Repeat** (for each step episode)
06)Take action $a$, observe $r$, $s'$
07)Choose $a'$ from $s'$ using policy derived from $Q$ (i.e. $\epsilon - greedy$)
08)$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
09)$s \leftarrow s'$
10)$a \leftarrow a'$
11)**until** $s$ is terminal
```

In table 7 the resulted values of the value function Q with $\alpha = 0.7, \epsilon = 0.1$ and $\gamma = 0.5$ can be depicted. We could see that the values are in general lower. Nevertheless, this performance is the expected one, as the SARSA algorithm checks the ϵ-greedy policy on the next state. Thus, this has as a consequence the reward to end up with a smaller value.

Table 7: Optimal value function Q^* for SARSA, $\alpha = 0.7, \epsilon = 0.1, \gamma = 0$

S1(L\R)	S2(L\R)	S3(L\R)	S4(L\R)	S5(L\R)	S6(L\R)
0\0	0.94\0.56	0.47\1.16	0.56\2.4	1.16\5	0\0

The performance of the SARSA algorithm can be observed in figures 25-34 where the difference between the ground truth and the values computed by the SARSA algorithm over the number of interactions are presented. Similarly to the previous exercises, we decided to provide a different plot for each separate value of the variable parameter (i.e α or ϵ) as there was a big overlap in a common plot and it was difficult to distinguish the differences. Also, for all cases the average of

100 iterations was used in order to obtain more representative results. In can be inferred by these figures that the choice of ϵ is more important in SARSA in comparison to Q learning. As fas as the influence of α is being considered, we can depict that it is similar in both cases.

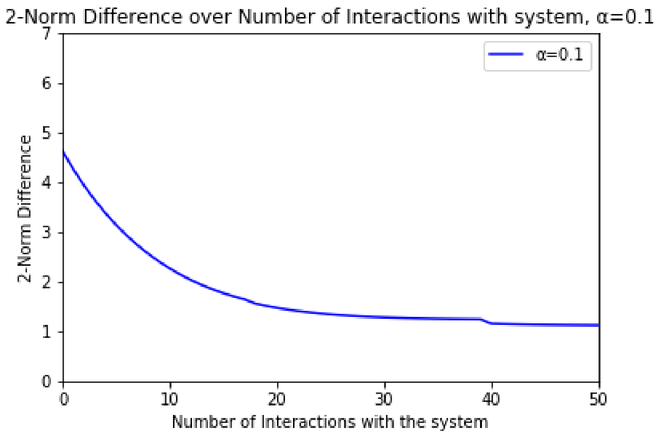

Figure 25: 2-Norm Difference over Number of Interactions with system

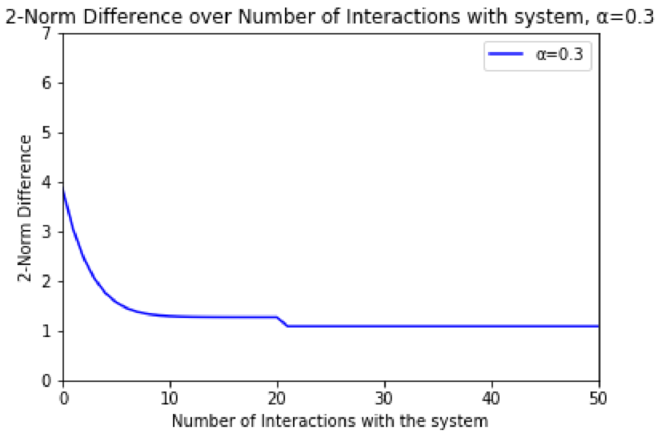

Figure 26: 2-Norm Difference over Number of Interactions with system

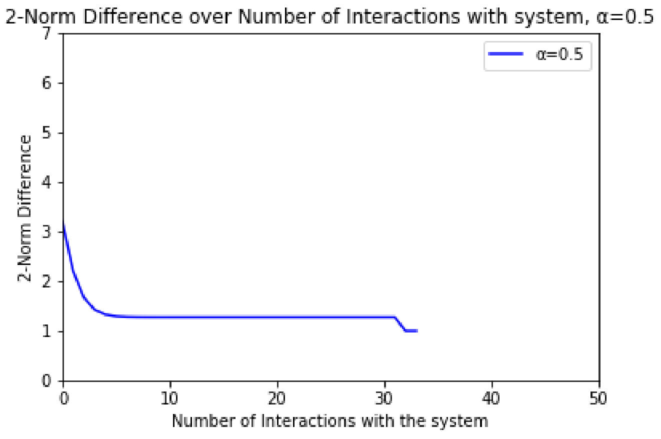

Figure 27: 2-Norm Difference over Number of Interactions with system

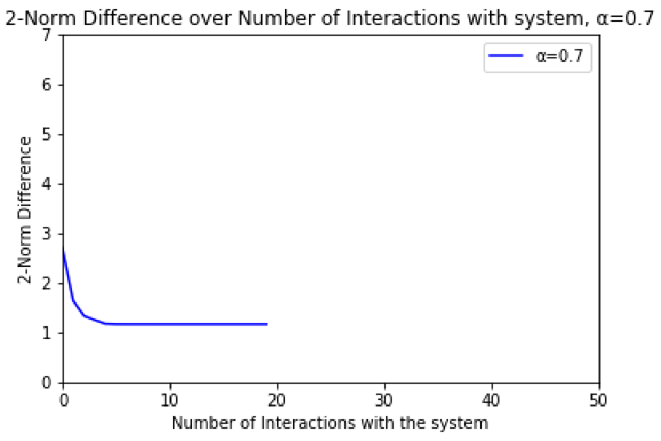

Figure 28: 2-Norm Difference over Number of Interactions with system

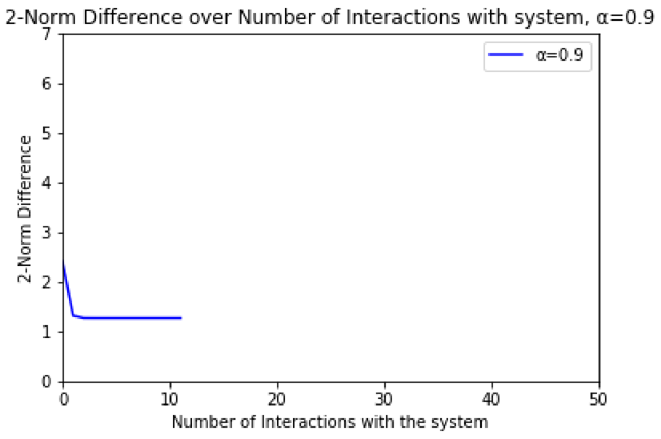

Figure 29: 2-Norm Difference over Number of Interactions with system

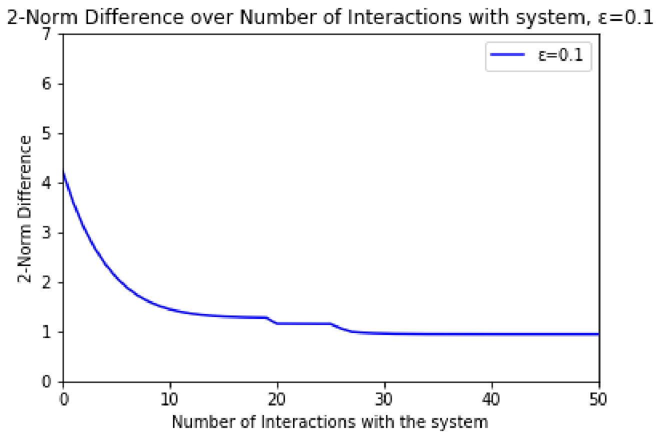

Figure 30: 2-Norm Difference over Number of Interactions with system

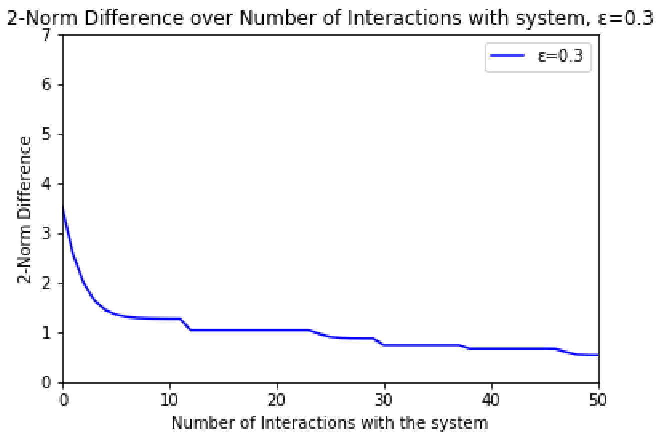

Figure 31: 2-Norm Difference over Number of Interactions with system

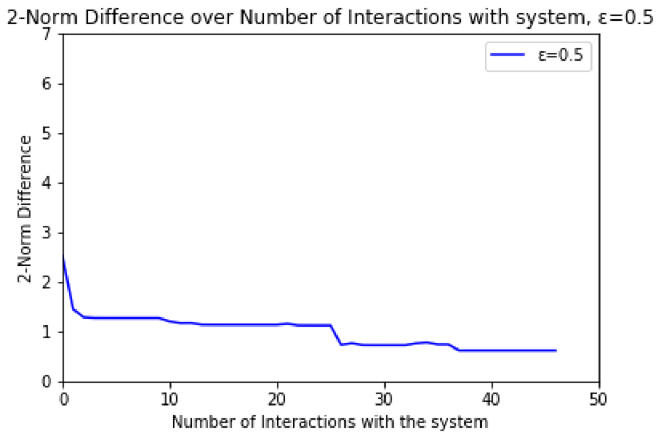

Figure 32: 2-Norm Difference over Number of Interactions with system

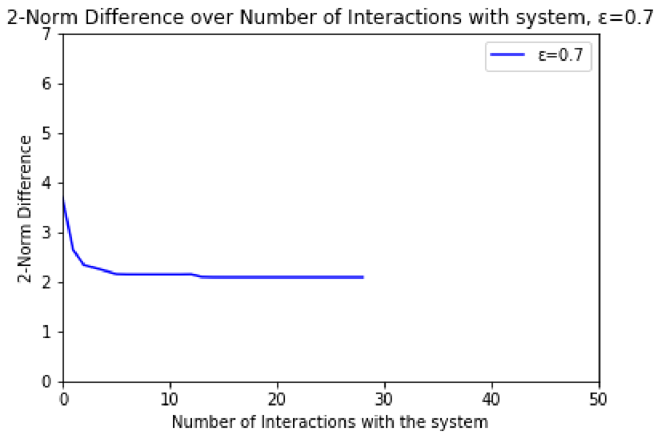

Figure 33: 2-Norm Difference over Number of Interactions with system

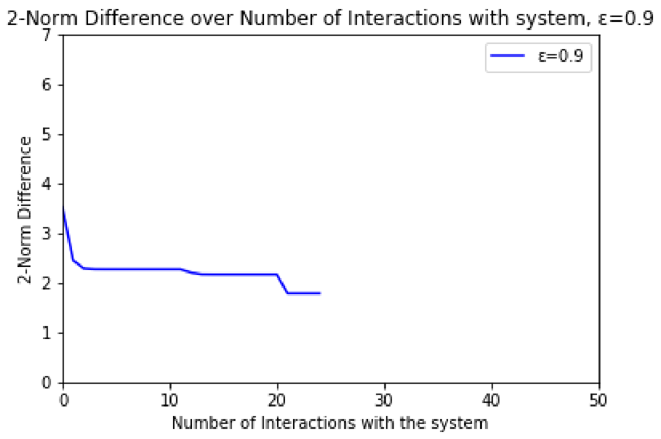

Figure 34: 2-Norm Difference over Number of Interactions with system

The main difference of Q-learning and SARSA is that Q-learning assumes that the next state is going to the optimal policy. However, this is not the case in SARSA which takes into account the control policy of the following state.
SARSA's main advantage is that it is able to overcome the states where a wrong choice of action results in a negative reward, even though choosing a correct action is the goal of the optimal policy. On the other side of the spectrum, Q-learning only takes into account the optimal policy. Thus, it always follows a strategy which leads to the highest reward despite the fact that there would be a possible high risk.
Nonetheless, the performance of the SARSA algorithm could be worse in a case that the policy of a system changes constantly. SARSA is going to always try to find the policy in contrast to the Q-learning strategy which is following the optimal policy that may lead probably in better actions. To sum up, Q-learning learns the optimal policy directly in contrast to SARSA which learns a near-optimal policy through exploring. In order for the SARSA to be able to learn an optimal policy a strategy to decrease the hyperparameter ϵ in ϵ-greedy action choice should be found.
Moreover, Q-learning algorithm as it is an off-policy learning has a higher variance than SARSA, something that leads to problems in convergence (especially in training Neural Networks). In addition SARSA tries to reach convergence and in parallel allows for possible penalties which are come from exploratory moves in contrast to Q-learning that ignores them. Hence, this has an effect that SARSA is in general more conservative. In case that there is a high risk of a negative reward which is close to the optimal path, Q-learning is going to follow this path in contrast to SARSA that tries to avoid a dangerous optimal path.
Hence, in practice it may be preferable to have a more conservative learning algorithm that avoids high risks (real life). As a conclusion, we could say that there is not the "best" algorithm and always it is problem-dependent which approach is going to have better results.

References

[1] *Sebastian Papierok, Anastasia Noglik and Josef Pauli. "Application of Reinforcement Learning in a Real Environment using RBF Networks". In Proceedings of the International Workshop on Evolutionary Learning for Autonomous Robot Systems, page 17-22. 2008*

[2] *Richard S. Sutton and Andrew G. Barto. "Reinforcement Learning: An Introduction". Bradford Book. The MIT Press Cambridge, Massachusetts London, England.*