Homework 6 Multi Agent Systems

Dirk Hoekstra, Philipp Lintl, Luisa Ebner

November 5, 2018

Code is found at: `https://github.com/Dirk94/mas-homework-6.git`

Problem 6.7.1. Let $X \sim \mathcal{N}(0,1)$. Use importance sampling to estimate $\mathbb{E}(X^2)$ by sampling from a uniform distribution $q \sim \mathcal{U}(-5,5)$. What value do you expect (based on your knowledge of the normal distribution)? (i) How accurate is your estimate based on importance sampling? (ii)

(i)Expected value $\mathbb{E}(X^2)$
Recalling the definition of Importance Sampling and Monte Carlo Estimation, for $X \sim p$ and $q \sim \mathcal{U}(-5,5)$ we arrive at :

$$\begin{aligned}\mathbb{E}_p(f(X)) &= \int f(x)p(x)dx \\ &= \int f(x)\frac{p(x)}{q(x)}dx = \mathbb{E}_q\left[f(X)\frac{p(X)}{q(X)}\right] \\ &\approx \frac{1}{n}\sum_{i=1}^{n} f(X_i)\frac{p(X_i)}{q(X_i)} \quad \text{for sample of independent } X_1,\ldots,X_n \sim q\end{aligned}$$

As given in the exercise sheet, $f(X) = X^2$. Therefore

$$\begin{aligned}\mathbb{E}(X^2) = \mathbb{E}((X-0)^2) &= \underbrace{\mathbb{E}((X-\mu)^2)}_{\mu=0,\text{ as } X\sim\mathcal{N}(\mathbf{0},1)} \\ &= Var(X) = \sigma^2 = 1\end{aligned}$$

The expected value thus emerges as $\mathbb{E}(X^2) = 1$.

(ii) Now, the value is approximated by Importance sampling, whereas samples $X_1,\ldots,X_n$ are obtained by the aforementioned Uniform distribution.
As the term *uncertainty* was not fully understood, we decided to once name the standard deviance of all

$$f(X_i)\frac{p(X_i)}{q(X_i)}$$

pairs and also once ran several simulations and gave the standard deviance of the $\mathbb{E}(X^2)$ Monte Carlo values. That way, we specify uncertainty within one run first and then assess,

whether the expected value varies among different runs.
Obviously due to the stochasticity of the sampling process, the accuracy depends highly on the number of samples, which is why we tested several n (number of samples) values first. They amount to the following estimates with respective standard deviation:

n	n_{100}	n_{1000}	n_{10000}	n_{100000}
$E(X^2)$	1.046503	0.9736529	1.011674	**1.000377**
$sd(X^2)$	1.124692	1.044809	1.058868	1.055724

Table 1: Importance Sampling estimates for several n values of problem 6.7.1

As seen in Table 1, for n=100000, the estimation of on MC simulation run is the closest to the value we expected according to the normal distribution ($1.000377 \approx 1$). For that n value, the sample pairs also differ from the expected value with a standard deviation of 1.055724, which is visualized in Figure 1 (a).

Now, 1000 simulations are run and the standard deviation among those MC estimates is looked at. Therefore, we chose n to be 100000 for each Importance sampling. The mean over those 1000 runs amounts to

$$\frac{1}{n_{runs}} \sum \mathbb{E}(X^2) = 1.000058, \quad \text{with a sd of } 0.003373392.$$

This results can also be seen according to the histogram of those 1000 simulated expected values (Figure 1 (b)). So uncertainty in this case means that over 1000 runs, the obtained expected values deviate with 0.003373392 from the mean of 1.000058.

Problem 6.7.2. Suppose some random process produces output $(-1 \leq X \leq 1)$ that is distributed according to the following continous density function:

$$p(x) = \frac{1 + cos(\pi \cdot x)}{2} \qquad (for\,(-1 \leq X \leq 1))$$

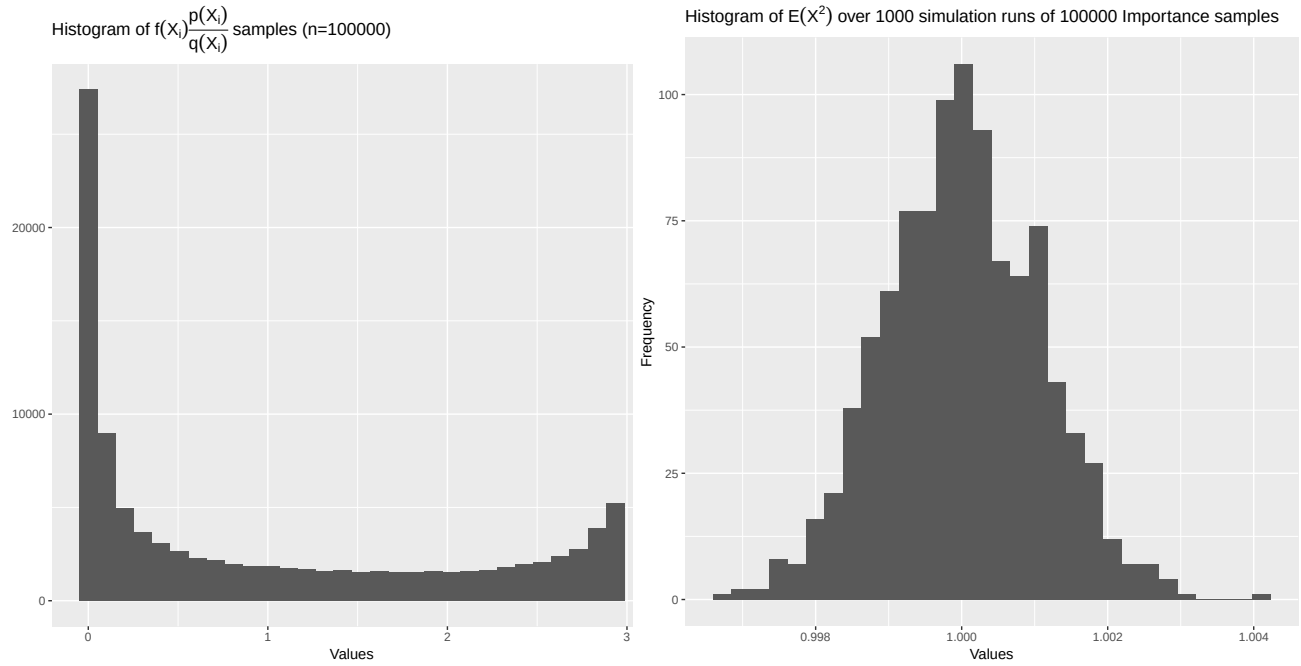
Again, we are interested in estimation of $\mathbb{E}(X^2)$. Use importance sampling to estimate this value. Quantify the uncertainty on your result.

This problem differs to the previous one in the p(x) distribution, as well as the sample generating distribution $q \sim \mathcal{U}(-1, 1)$. Similarly, we assess uncertainty in two ways.
First, the expected value is given for several n values, as seen in Table 2. The distribution

n	n_{100}	n_{1000}	n_{10000}	n_{100000}
$E(X^2)$	0.1254172	0.1275439	0.1303723	0.1302756
sd	0.0923248	0.08950353	0.08943098	0.09000635

Table 2: Importance Sampling estimate for several n values of problem 6.7.2.

of X^2 pairs for n=100000 is seen in Figure 2 (a).



(a) (b)

Figure 1

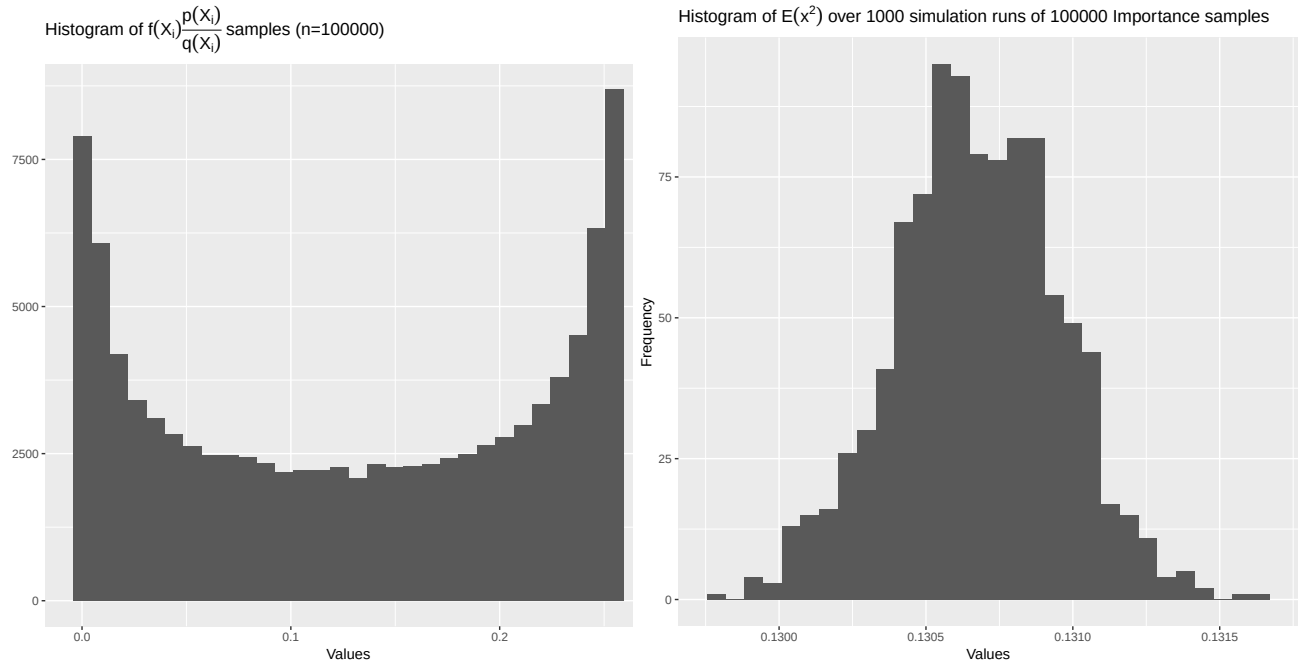
Now, 1000 simulations are run and the standard deviation among those MC estimates is looked at. This results can also be seen according to the histogram of those 1000 simulated expected values (Figure 2 (b)). So uncertainty in this case means that over 1000 runs, the obtained expected values deviate with 0.000281865 from the mean of 0.1306723.

Problem 6.8. SARSA and Q-Learning for the Gridworld example to derive the optimal policy

As given in the assignment sheet, the Gridworld is depicted as a 8×8 grid with black gridcells representing walls, and two terminal states, in particular one red field called *snakepit* and the *treasure* field coloured in green. The snakepit imposes a reward of -20, the treasure of 10 and each step into a non terminal step one of -1. A step into a black field or the wall is possible and implies a reward of -1, though leaving the state unchanged. The goal was to use both SARSA and Q-Learning to find optimal policies.

The code environment is described more closely for problem 6.9, which is why we refrain from elaborating on the details at this point. It only differs in the size of the gridworld which is 8×8, the different terminal states (*snakepit, treasure*), the feature, that walls and borders can be bumped into without changing the state but implying -1 reward and that there is no given starting state but sampling is required. The code for this problem is found in the `Gridworld` folder.

The characteristics of our algorithms can be described as follows. Starting with Q-Learning,



(a) (b)

Figure 2

we chose a starting ϵ to be 0.1, meaning that there is a 10% chance to take the non-optimal action. So the agent is moving according to a ϵ-*greedy* policy. The Learning rate α was chosen to be 0.1, the discount factor γ was set to 1 and the starting position of each episode was initialized randomly, which is required to ensure each state action pair is theoretically met infinitely often. That way, each Q-Learning episode updates its q-value as follows:

$$q(S, A) \leftarrow q(S, a) + \alpha(R + \gamma \max_{a'} Q(S', a') - Q(S, A))$$

Following the notion presented in the lecture that Q-Learning converges to the optimal value function $Q(S, A) \rightarrow q_*(S, A)$, we obtained the result given in Figure 3. In it, we filled each cell with its maximum q-value, implying that each terminal or wall state is 0. We chose an episode number of 100000 which already was sufficient, as the algorithm converged, which is noticeable, as the maximum q values per state are integers. That way, in each state the optimal action is clear, leading to the values seen in Figure 3. So the maximum q-value simply reflects the final reward - the number of steps it takes to get there. Noticeably the number of episodes to convergence is quite high (10000), which does not hold for the policy itself (It has to be remarked that we havent actually tested when convergence is reached, but rather in the area around 10000 episodes). That can be seen, when looking at the max q-values which arise after 1000 episodes (Figure 4 (b)). According to it, the policy is already almost the same, meaning that the agent will follow according to the optimal policy, which was exactly the task. So for Q-Learning, around 1000 episodes are enough to lead to the optimal policy visualized in Figure 4 (b). The according optimal policy visualized with arrows for each cell, which action should be taken can be seen in Figure 4 (a).

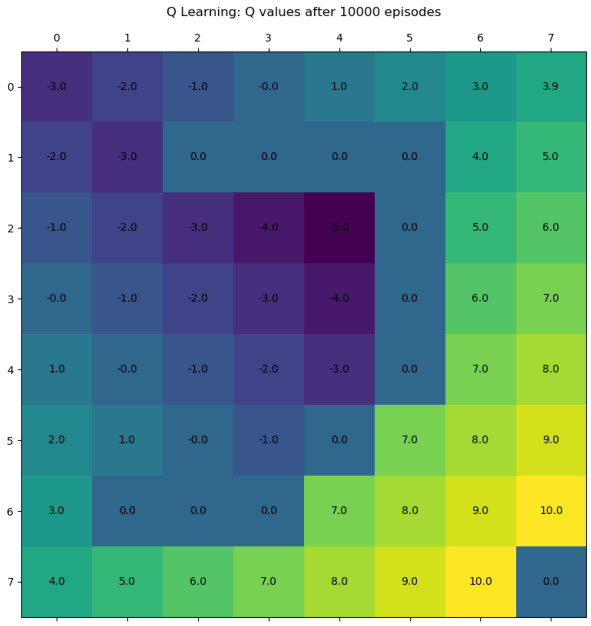
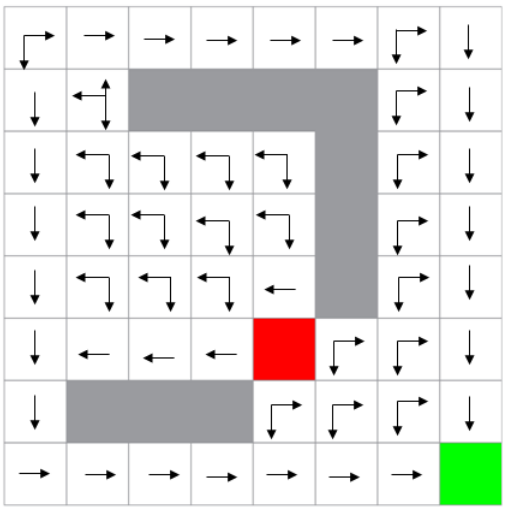
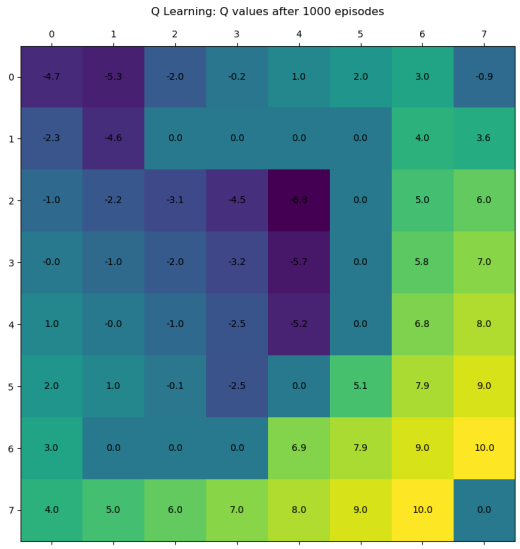


Figure 3: Maximum Q-values of each state on the 8×8 grid for 10000 episodes

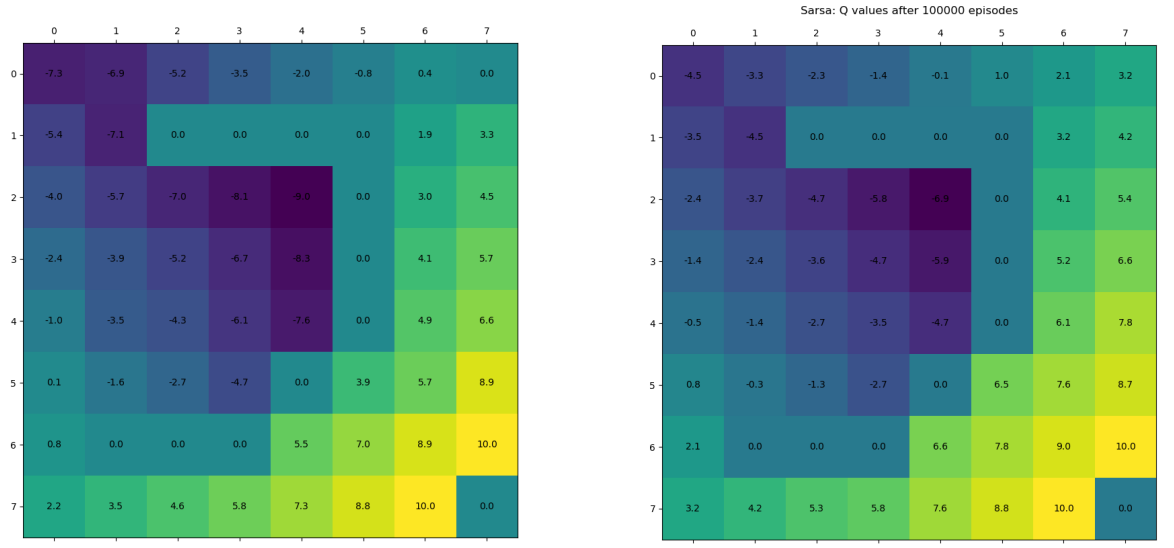


(a) Arrows representing optimal policy



(b) Maximum Q-values of each state on the 8×8 grid for 1000 episodes

Figure 4



(a) Maximum Q values 1000 episode SARSA

(b) Maximum Q values 100000 episode SARSA

Figure 5

Following the same approach for SARSA leads to the same optimal policy but requires far more episodes to converge (can not provide the exact converged output). For Sarsa to converge, GLIE and Robbins-Monro sequence for α needs to be fulfilled. ϵ decreasing in the number of episodes and $\alpha = 0.1$ fulfill this requirement and thus should provide convergence for enough episodes taken. To gradually compare it to Q-Learning, we provide the following figures. It is clearly noticeable, that SARSA does not reach convergence, even if 100000 episodes have been sampled (Figure 5 (b)), which means that more episodes are required to fulfil convergence. On the other hand, already 1000 episodes yield maximum qvalues according to which an optimal policy can be followed.

Problem 6.9.1. Now, SARSA and Q-Learning are implemented and scrutinized with respect to the Cliffwalking gridworld.

The code for this problem is found in the `cliff-walking` folder.

We represent the world with the `GridWorld` class. This gridworld has $n * m$ gridcells, a.k.a. states.

We represent a gridcell (state) with the `GridCell` class. Each gridcell has a `reward` that is defined in the cliff walking example.

Each gridcell has possible actions, the `GridCell.get_possible_actions()` returns all the possible actions of a gridcell. I.e. for the gridcell at location $(0, 0)$ this would be the actions `RIGHT` and `DOWN`. For a terminal gridcell no actions are returned.

In both the `QLearningAgent` and the `SarsaAgent` we store the Q values using a python dictionary with a $(gridcell, action)$ tuple as key. Initially all Q values are 0.

In the `QLearningAgent.run(number_of_episodes)` and the `SarsaAgent.run(number_of_episodes)` the actual algorithms are done.

We ran both algorithms with the following values:

- $\alpha = 0.1$, $\gamma = 1$, and $\epsilon = 0.1$ (decreasing per episode).

First we ran the Q-Learning agent. In Figure 9 the max Q value of each gridcell is shown after 50000 episodes.

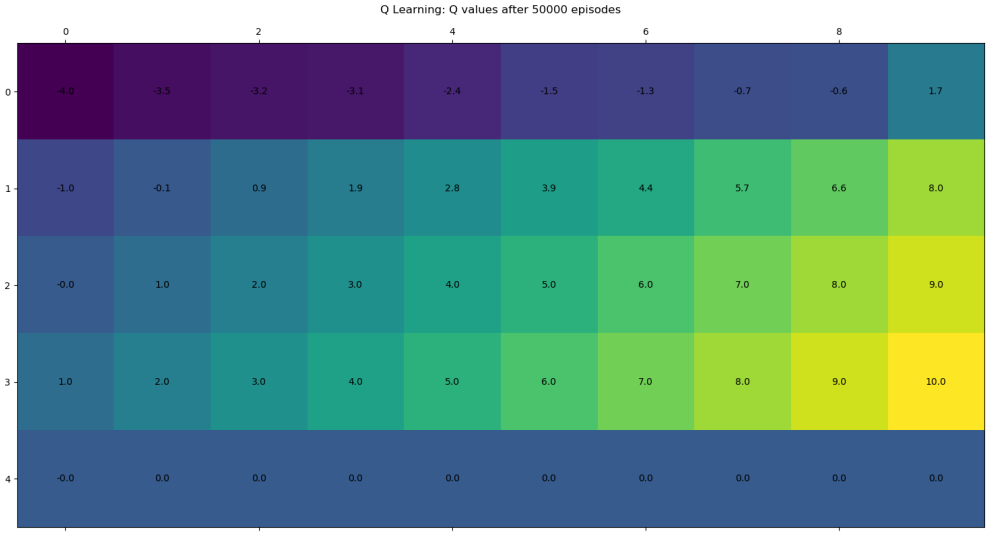


Figure 6: Q-Learning Q values after 50000 episodes.

When observing the values we can see that the Q-Learning agent will take the optimal path, which is also visualized similar to 6.8 with arrows on the grid (Figure 8 (b)).

Next, we ran the SARSA agent. In Figure 7 the Q max value of each gridcell is shown after 50000 episodes.

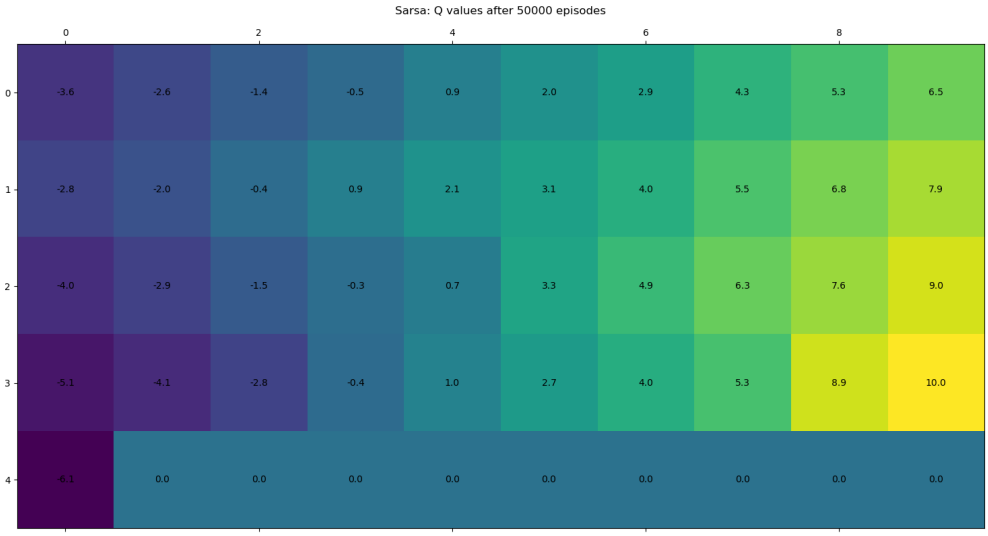
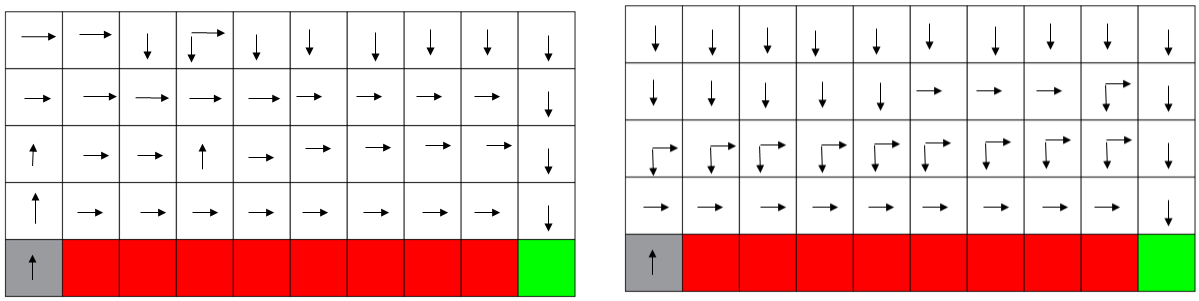


Figure 7: Sarsa-Learning Q values after 50000 episodes.

We observe that the plot is not as symmetrical as the Q-Learning plot. This makes sense because where Q-learning always takes the next best action to update its value, the SARSA takes the next action from the policy. So if the SARSA's next action would be to fall from the cliff, the Q value of that cell will go down drastically.

We do see that Q-Learning converges on the optimum path, and that SARSA almost converges on the safest path but still is pretty safe (Figure 8 (a)).

Next, in Figure 9 we plot the average total reward per 100 episodes of Q-Learning and Sarsa.



(a) Policy obtained by Sarsa

(b) Policy obtained by Q-Learning

Figure 8

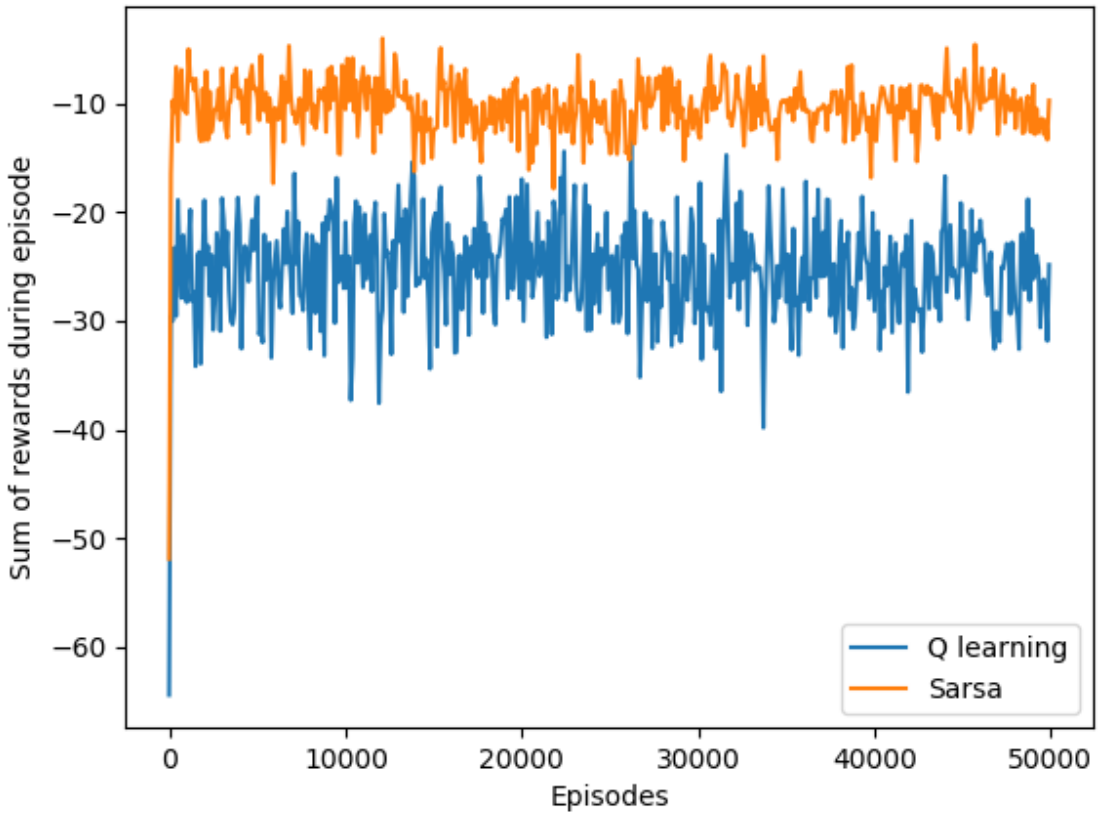


Figure 9: Q-Learning and Sarsa average total reward per 100 episodes

We see that, as in the book, SARSA performs better than Q learning. Q learning takes the optimal path. However, because of the ϵ greedy policy it will sometimes wander of the cliff. This explains why the SARSA algorithm, that takes the safe route, performs better.

Problem 6.9.2. How do different ϵ values influence the result

When running the algorithms with $\epsilon = 0.01$ we get the following results:

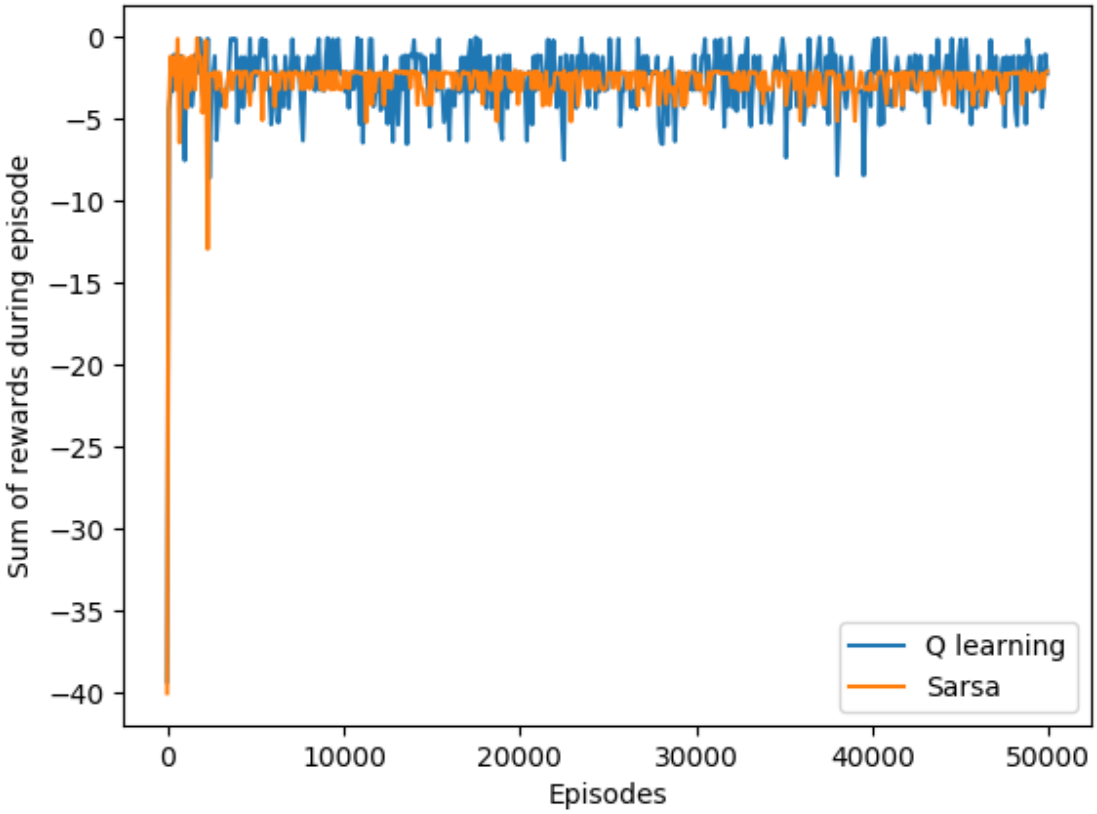


Figure 10: $\epsilon = 0.01$, Q-Learning and Sarsa average total reward per 100 episodes

Both algorithms perform almost identical now. This makes sense because as our ϵ decreases the best action is taken almost all of the time. Thus, making the 2 algorithms more identical.

When we run the algorithms with a large $\epsilon = 0.4$ we get the following results.

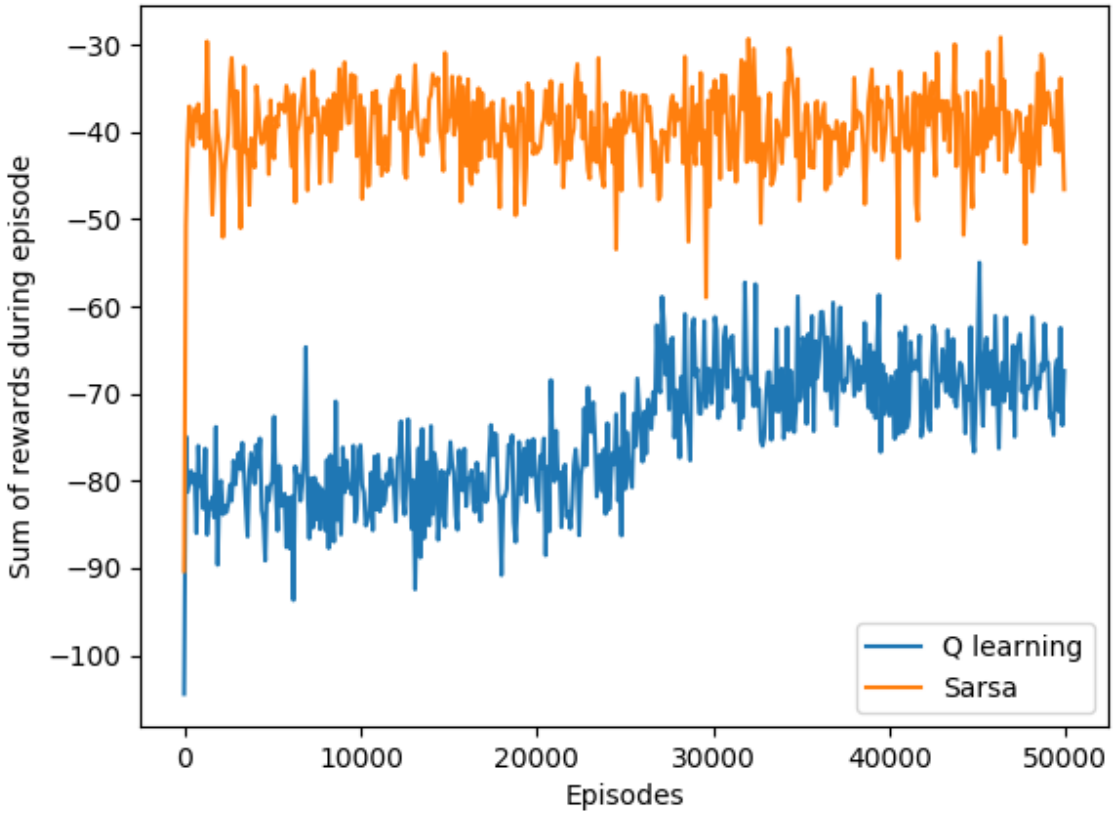


Figure 11: $\epsilon = 0.4$, Q-Learning and Sarsa average total reward per 100 episodes

A bigger ϵ means that the algorithms will explore more. This explains the lower average total sum in the graph and the big jumps. It makes sense that Q learning performs worse, as it has a higher probability of wandering of the cliff than SARSA.