Advanced ML - Tabular RL - Assignment

Albin Jansfelt and Amanda Allgurén

May 2022

1 Introduction

Sarsa

Sarsa stands for "state action reward state action" which describes every component of the algorithm. The algorithm is on-policy, meaning that it has the same policy for exploration and exploitation (target and behaviour policy).

The main part of the Sarsa-algorithm is that it updates the Q-value given the current state and action with the help of the reward and the next state and next action. The update is as follows:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A)) \tag{1}$$

Here α is the learning-rate and γ is discount factor. The next action, A', is chosen with for example ϵ-greedy.

Q-learning

Q-learning is an off-policy algorithm, where both a target and a behaviour policy are decided. The target policy is the one which decides the optimal moves for the agent, whereas the behavior policy is used in training to include exploration.The target policy is choosing the action with the maximum Q value, based on the current state. The behaviour policy is in this case using an epsilon greedy policy.

The update of Q is as follow:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma max_a[Q(S', a)] - Q(S, A)) \tag{2}$$

α is the learning-rate and γ is discount factor.

To compare with Sarsa-algorithm, Q-learning takes more "risky" actions while learning. This is because we use the maximium-operation to update the Q-values. This might lead to bad results in the beginning but will converge to optimum policy.

Double Q-learning

The double Q-learning is similar to standard Q-learning, except that two Q-matrices are used, Q_1 and Q_2. With a 0.5 probability we use Q_1 to determine the best action and Q_2 to yield the value of the state-action pair. Otherwise, we switch the role of Q_1 and Q_2. The update follows Equation 3 with a probability 0.5 and equation 4 with a probability 0.5.

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha(R + \gamma Q_2(S', argmax_a[Q_1(S',a)]) - Q_1(S,A)) \quad (3)$$

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha(R + \gamma Q_1(S', argmax_a[Q_2(S',a)]) - Q_2(S,A)) \quad (4)$$

The main benefit of double Q-learning compared to regular Q-learning is that we spend less time in suboptimal states during exploration. By using double Q-learning, maximization bias can be avoided.

When to use regular Q-learning or double Q-learning can be discussed with examples. Say for example that you have a one-dimensional state space with rewards in both ends (the agent starts in the middle). One of the reward is small and one is large and it is easier to get to the small reward than the larger. In the long run, we have tuned the rewards making it best to try to go to the larger rewards (to compare with riverswim environment).

With regular Q-learning while exploring, the agent might tend to prioritize the smaller reward, creating a maximization bias, but will eventually find out that the larger reward is to prefer. With double Q-learning we will start to prefer the larger reward much faster.

On the other hand, if there would only be one reward in the state-space (FrozenLake for example), then there is no risk that a regular Q-learning agent starts to prefer the wrong reward. Thus making Q-learning more efficient since we only have to update one Q-matrix.

2 Results and Discussion

For all the environments and agents, the number of runs was 5, number of episodes was 2500 for FrozenLake and 500 for Riverswim. Maximum step per episode was 100 (only needed for riverswim, since FrozenLake is terminated by terminal states). Further, the window used in moving average was 125 and the error probability in the error plot was 95%.

For the FrozenLake environment, the learning rate used was 0.1. To lower the fluctuation on the Riverswim rewards, the learning rate was lowered to 0.01. The discount rate used was 0.95.

The optimal policy was obtained by taking the argmax value for each row (states) in the Q-matrix.

Further, the initialization of Q does not matter in theory but might be slower to converge in practice. We have tested initializing the Q-matrices with random

numbers and found it coverging. However, when setting Q-matrix to zeros it will take a very long time to find a reward (for FrozenLake). But, when increasing the exploration rate (increasing ϵ), we eventually found a reward. Therefore, we have initialized all Q-matrices to ones (but zeros for terminal states).

2.1 FrozenLake-v1

The FrozenLake environment was run for Sarsa, Q-learning and double Q-learning algorithms. The randomness in FrozenLake is that when you try to take an action you have 1/3 chance to actually take that step, 2/3 chance to go either left or right of the tried direction.

Looking at the policies derived for Frozen Lake (see Table 1, 2 and 3, all three algorithms seem to have reached an optimal policy. All states where the agent have the opportunity to avoid the risk of fall into a hole completely, are taken correctly. In the states where the there will always be some risk of falling in, the agent minimizes this risk. This is done by going directly at the hole if taking another direction means being surrounded by two holes (thereby yielding a 1/3 risk of falling in compared to 1/3 + 1/3 risk).

The policy found by the algorithms seems to be the optimal policy. With the randomness in mind, the best path is: down, down, right, down, right, right. In our case we tend to prefer going into the wall. This is because we then have no way of falling in to a hole.

2.1.1 Sarsa

The moving average reward for the sarsa agent in the FrozenLake environment is shown in Figure 1. The optimal policy is visualized in Table 1.

As we can see in Figure 1, the average rewards increases with the number of episodes. This means that the algorithm learns and performs better with each episode. It seems to have learnt the optimal path at around 1000 episodes and the remaining fluctuation may be explained by the randomness existing in the environment.

One question to ask is why we only have an average reward of 0.5 as the highest limit. This can be due to the fact that we do not decrease the exploration probability with respect to the number of episodes. Even though the algorithm have found the optimal path, there still is a probability of exploring a random path which might lead to falling down.

Another answer is that even if the agent always exploit after finding the optimal policy, the randomness in the environment makes it possible to fall into a hole. For example if the agent is in state (2,1) (starting with 0), it might end up going to the right. And in the next state, taking the best action (to the left), it has 1/3 probability of going up instead and lastly 1/3 probability of going in the hole.

Finally, since Aarsa is an on-policy algorithm we exploit on the same policy that we train on. This is partly the reason we can see the result described above.

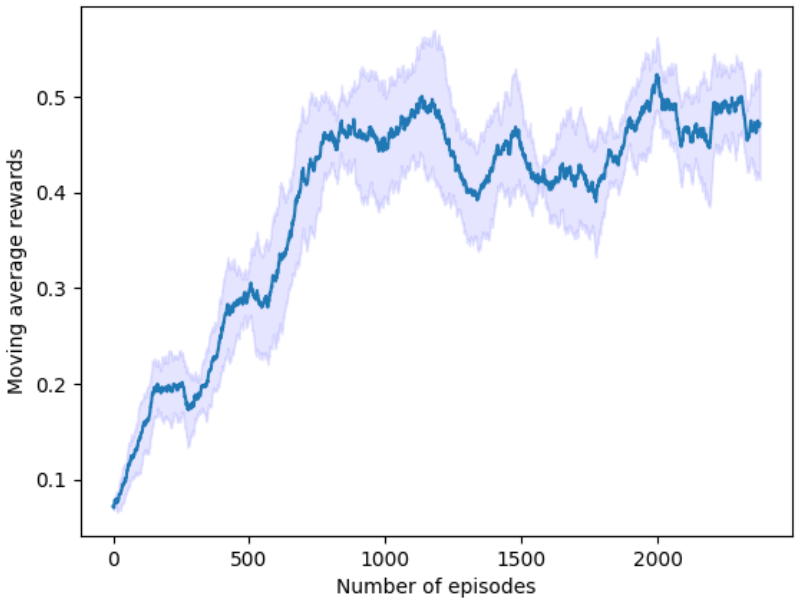

Figure 1: Moving average reward with sarsa-agent on FrozenLake environment with 2500 episodes averaged over 5 runs

←	↑	←	↑
←	-	→	-
↑	↓	←	-
-	→	↑	-

Table 1: Optimal policy with sarsa-agent on FrozenLake environment

Q-learning

The optimal policy found by the Q learning algorithm is shown in Table 2. It is very much alike the policy Sarsa found, where only the first and second last state differs.

As can be seen for Q-learning in Figure 2, it looks very much alike the Figure 1 describing Sarsa. The difference that can be noted is that Q learning seems to reach rewards of about 0.5 faster, after about 500 episodes, and also stabilizes faster.

This is somewhat different from what the theory would tell us, as Q learning is an off-policy algorithm and always explore during the learning phase. By this approach, Q learning would take more risky actions and thereby fall into the holes more often than Sarsa, possibly yielding lower rewards in the training

←	↑	←	↑
←	-	←	-
↑	↓	↓	-
-	→	→	-

Table 2: Optimal policy with Q-agent on FrozenLake environment

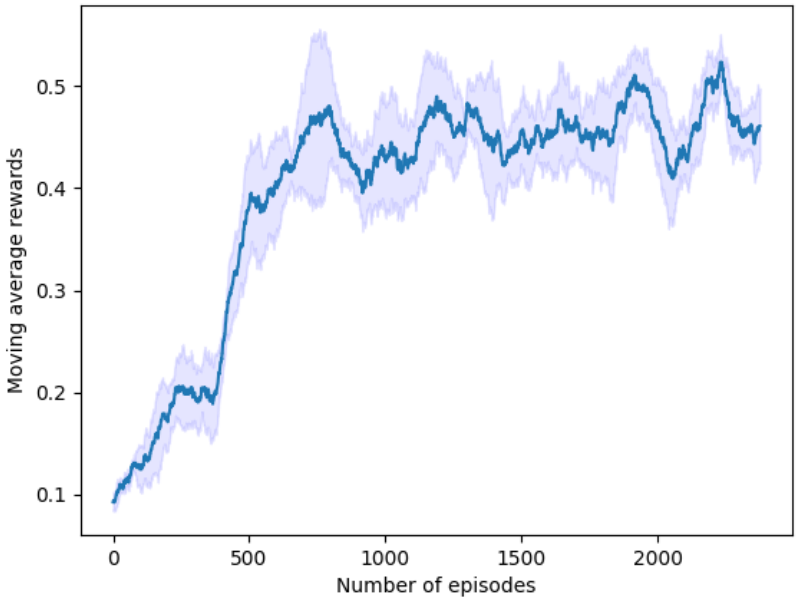

Figure 2: Moving average reward with Q-agent on FrozenLake environment with 2500 episodes averaged over 5 runs

phase. Also, since it takes more risky actions, it could make it find a better target policy than Sarsa since Sarsa is more careful in its exploration. In this case, both Q learning and Sarsa seems to have found the optimal policy if consider Table 1 and 2.

The plots shows that both Sarsa and Q learning reaches the highest moving average rewards of 0.5. To be noted is that the curve of moving average rewards for Q learning is presenting the rewards received during the learning process, and thereby by its behaviour policy. When the agent will later use the target policy, it could be receiving higher rewards than what is shown in this plot.

Double Q-learning

The policy found by the double Q-learning agent is shown in Table 3. Similar to Sarsa and Q-learning, it is the optimal policy. Figure 3 visualizes the moving average rewards for the agent over 2500 episodes. In comparison to Sarsa and Q-learning we see that double Q-learning stabilizes a bit later (around 1500 episodes compared to around 1000 episodes). This might be because we have 2 Q-matrices which we update with a 50% probability. It will thus take longer time to converge to optimal policy. Moreover, we do not see the upside of double Q-learning here either, since we have no risk of maximization bias (as described earlier) for this environment.

←	↑	↓	↑
←	-	←	-
↑	↓	←	-
-	→	↓	-

Table 3: Optimal policy with the double Q-learning agent on FrozenLake environment

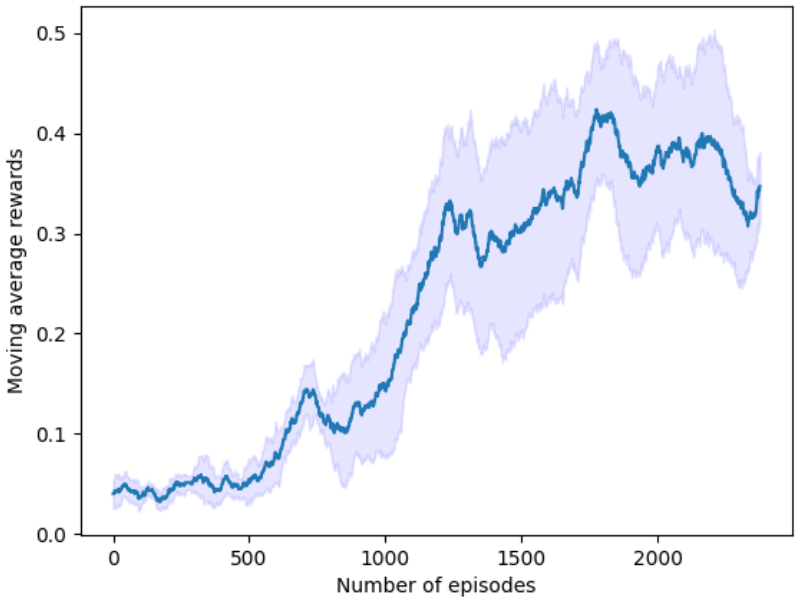

Figure 3: Moving average reward with the double Q-learning agent on FrozenLake environment with 2500 episodes averaged over 5 runs

2.2 Riverswim

The riverswim environment is a one-dimensional state-space consisting of two rewards on either side. You start at the side with the small reward (5/1000) and can either go backwards (stay in the first case) or forward towards the larger reward (1).

For all agents the optimal policy is described in Table 4 and was to always try to go forward. This is because the small reward is so small in comparison with the large. You have to fail to go to the large reward 200 times to justify staying at the smaller reward.

→	→	→	→	→	→

Table 4: Optimal policy for all agents on the Riverswim environment

The moving average reward over 500 episodes for Sarsa, Q-learning and Double Q-learning is presented in Figure 4, 5 and 6, respectively.

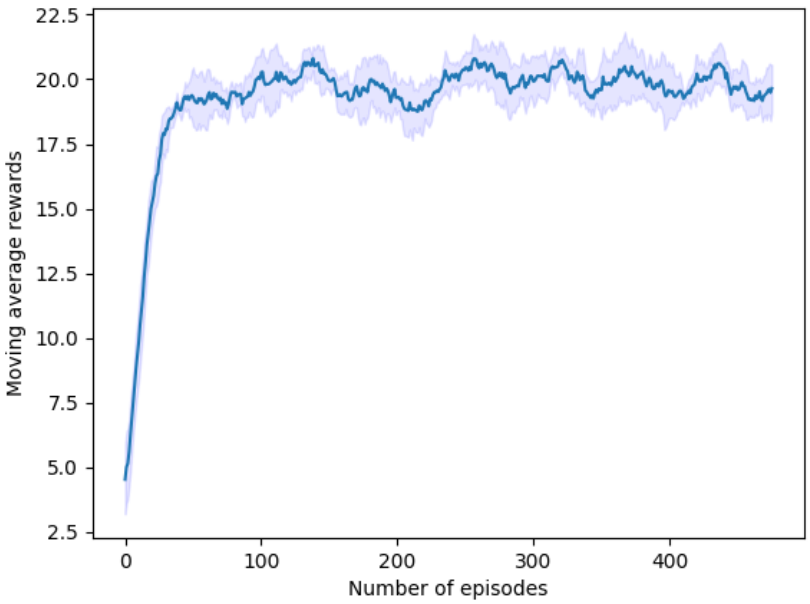

Figure 4: Moving average reward with the sarsa agent on Riverswim environment with 500 episodes averaged over 5 runs

As discussed earlier, due to maximization bias, it would be reasonble to guess that the average reward early for double Q-learning would be greater than for regular Q-learning. However, what we can see is that double Q-learning

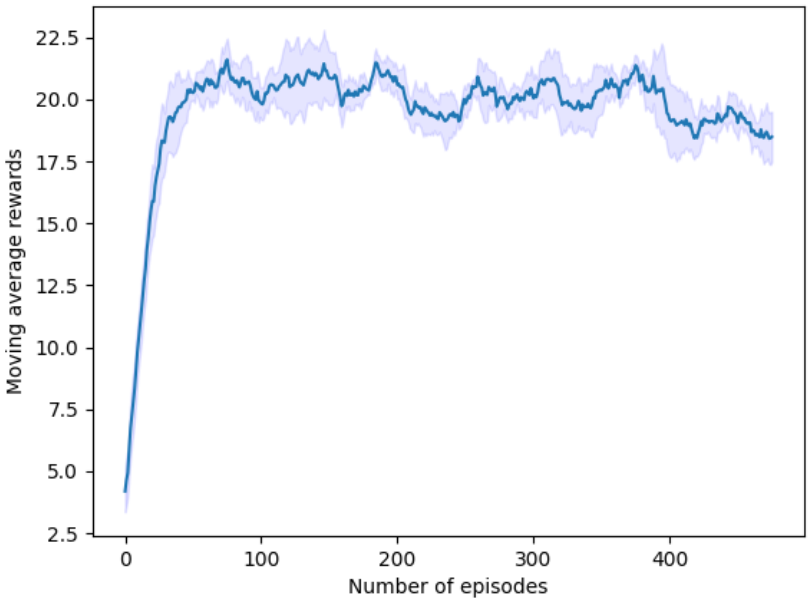

Figure 5: Moving average reward with the Q-learning agent on Riverswim environment with 500 episodes averaged over 5 runs

converges more slowly (similar to with FrozenLake). This could be due to the fact that the small reward is so small that not even Q-learning bother with it. One could try to tweak the small reward to see maximization bias in action.

For sarsa and Q-learning the moving average looks very similar and may only differ because of randomness in the environment. As discussed previous, we would expect to see somewhat different plots for these since one is an on-policy algorithm and the other off-policy.

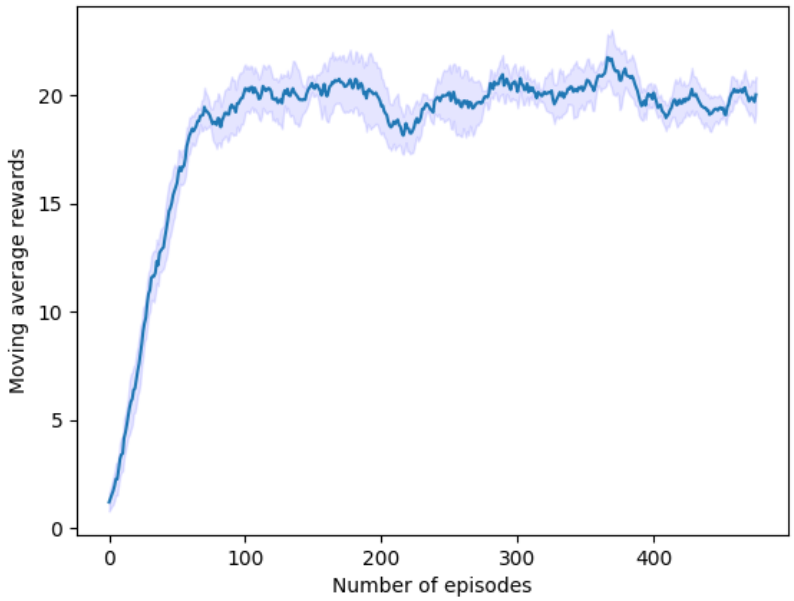

Figure 6: Moving average reward with the double Q-learning agent on Riverswim environment with 500 episodes averaged over 5 runs