DAT441 Assignment 2 - Tabular RL

Mirco Ghadri & Nazif Kadiroglu

October 20, 2024

Task 1 - Describe and compare algorithms

All these policies have in common that they don't require a model of the environment, i.e. its reward function and transition probabilities. This is in contrast to DP methods that use the Bellmann equation which do require a model of the environment(equation). Q-learning and Double-Q learning are so called off-policy methods while SARSA and Expected SARSA are on-policy methods.

Q-learning: is a form of off-policy learning that learns the optimal policy of an environment without necessarily knowing the model of the environment, i.e its reward function and transition probabilities. It's target policy is the greedy policy that selects the action that maximizes the state-action for a specific state(equation). It's behavior policy is epsilon-greedy, which means that with high probability it selects the greedy action, i.e. the one described above, but with a smaller probability of eps for each action that is not greedy, it explores it.

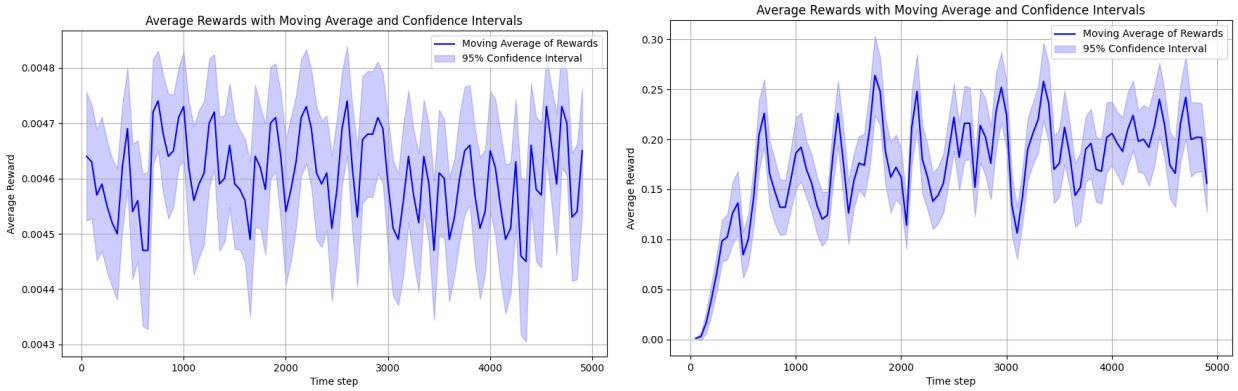
Double Q-learning: This algorithm works like Q-learning. It is also an off-policy method with target policy being the greedy policy and behavior policy being epsilon greedy. The difference is that Double-Q learning has 2 separate estimates of the Q-value, called Q1 and Q2. In Q learning when it updates the Q-value for a state-action pair, it uses the current Q value estimate for the state-action pair, the reward of the action, and the Q-value of the state transitioned to according to the greedy policy. In Double Q learning, this is similar. With 50% chance, it will update the Q1-value and with 50% chance it will update the Q2-values. When it updates the Q1-values, it uses the current estimate of Q1-value, the reward, and the Q2 value of the action that maximizes the Q1-value in the state transitioned to, i.e S'. When it updates the Q2-value, it uses the current Q2 estimate, the reward of the action taken, and the Q1-value of the action that maximizes the Q2-value in the state transitioned to.

SARSA: This is an on-policy method, meaning the policy it's target and behavior policy are the same. It uses e-greedy policy to explore. Hence, when it updates the Q value for a state-action pair, it uses the current Q-value estimate, the reward, and the Q-value estimate of the state transitioned to according to the e-greedy policy. This means it calls the e-greedy policy from state S' also to get the action A'. It might then seem counterintuitive how it finds the optimal policy as it never uses the optimal policy, but it converges to the optimal policy if epsilon becomes smaller with time so the e-greedy policy becomes the greedy policy.

Expected SARSA: This is similar to standard SARSA, except that when it calculates the Q value for an state-action pair, it uses the current estimate, the reward and the expected Q-value for the state transitioned to. This expectation is taken over the e-greedy algorithm used in exploration.

Task 2 - Initialization of Q-values

For RiverSwim, we noticed a clear difference in performance when initializing Q-values optimistically(in this case by setting them all to 5) and when we initialized them all to 0.



(a) RiverSwim Moving Average with Q-values initialized to 0

(b) RiverSwim Moving Average with Q-values initialized to 5

Figure 1: Comparison of RiverSwim Moving Averages with different Q-value initializations

From the 2 figures, we can see that when the Q-values were initialized to 0, the moving average never seemed to improve and was also stuck at a very low value, approximately 0.0046. The error bars show 95% confidence interval since we ran the experiment for 5 times and plotted the average moving average. However, when we initialized all Q-values to 5, we see that the moving average improved and reached a much larger stable value of approximately 0.2. So why did this happen then? The reason can be discovered if we study the rewards for both cases and also the Q-values. We got the following Q-values:

State	Action Left	Action Right
0	2.8174	**2.8212**
1	2.8653	**2.8998**
2	3.1945	**3.3026**
3	3.3582	**3.9165**
4	3.9369	**4.9112**
5	4.5762	**5.9100**

(a) Q-Learning RiverSwim Optimistically

State	Action Left	Action Right
0	**0.1000**	0.0924
1	**0.0950**	0.0290
2	**0.0171**	0.0000
3	0.0000	0.0000
4	0.0000	0.0000
5	0.0000	0.0000

(b) Q-Learning RiverSwim Zeros

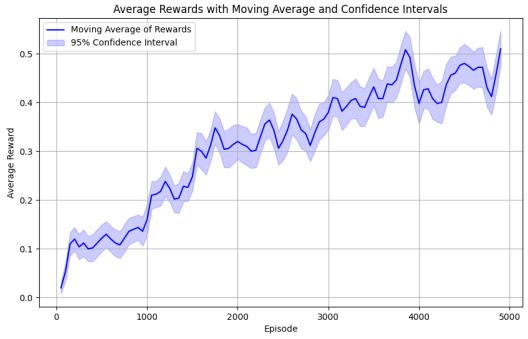
Figure 2: Comparison of Q-values for RiverSwim Environment

From the table of Q-values, we can see that initializing all the Q-values optimistically to 5, the Q-values converge such that the right action always has the largest Q-value. This can be confirmed to also be the optimal policy. However, when Q-values are all initialized to 0, the left action will have the largest Q-value in the 3 first states. This means the agent will always select moving left according to the greedy policy. When looking at the rewards, we can see why this happens. When all the Q-values are initialized to 0, once the agent gets the small reward from moving left in state 0, that action will have the largest Q-value of all states. Since the policy is greedy, it will keep selecting this action. From the rewards, we can see that the agent will get a consistent reward of 0.005. However, in the case all Q-values are initialized to 5, it will encourage the agent to explore other states also. It will eventually explore the rightmost state and learn that it is the best state to be in. If we look at the rewards here, we see a repeated reward of 1, which is the large reward you can receive in the last state.

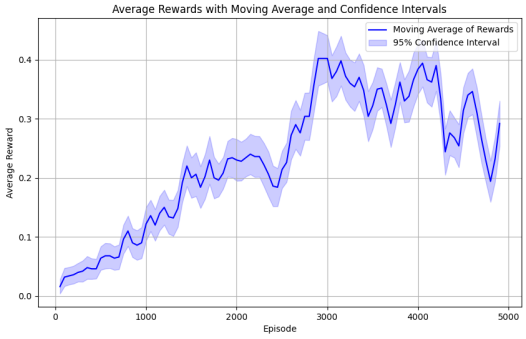
The same result is true for all the other algorithms on RiverSwim, i.e Double Q-Learning, SARSA and Expcected SARSA.

For FrozenLake, it did not matter whether we initialized all Q-values to 0 or to 5. This can also be understood by studying the FrozenLake environment. It only gives 1 positive reward and that is if you reach the goal state. It does not give any smaller positive rewards for taking suboptimal actions. Hence, it will not learn some suboptimal policy and stick to it, it will keep exploring until it reaches the goal even if we initialize all Q-values to 0. This was true for all 4 TD-learning algorithms.

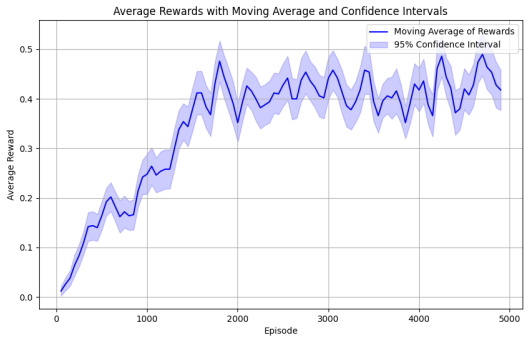
Task 3 - Run and plot rewards



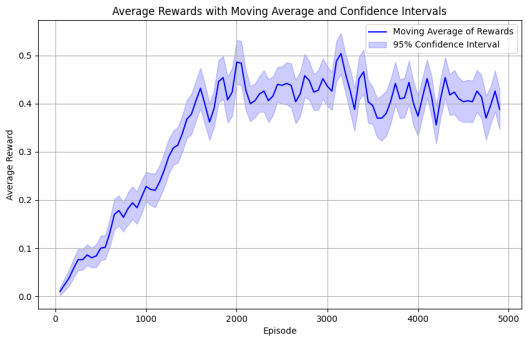
(a) Q learning moving average FrozenLake



(b) Double Q learning moving average FrozenLake



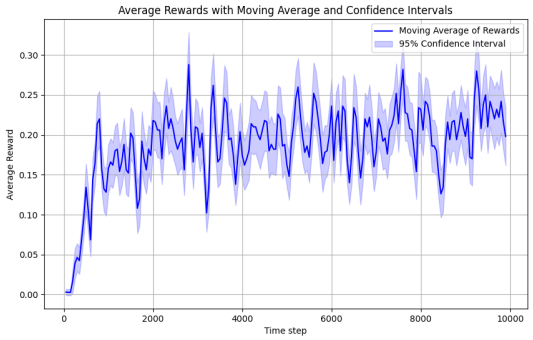
(c) SARSA moving average FrozenLake



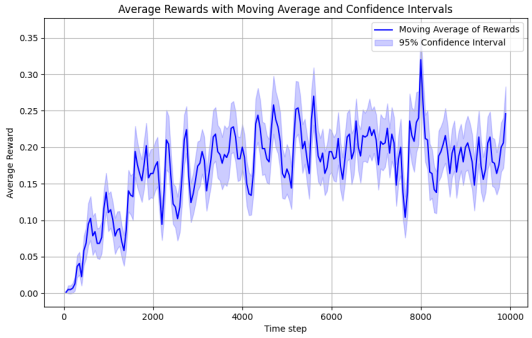
(d) Expected SARSA moving average FrozenLake

Figure 3: Comparative moving averages for different TD-algorithms on FrozenLake environment

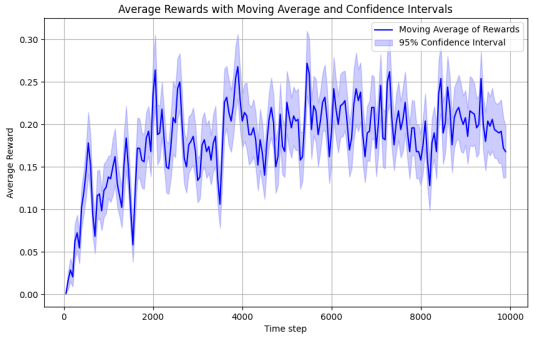
For the FrozenLake environment, we used 5000 episodes and initialized all Q-values to 0.



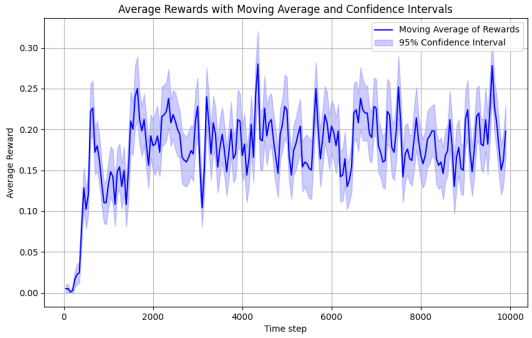
(a) Q Learning Moving Average RiverSwim



(b) Double Q Learning Moving Average RiverSwim



(c) SARSA Moving Average RiverSwim



(d) Expected SARSA Moving Average RiverSwim

Figure 4: Moving Averages of Different Learning Algorithms in River Swim

Since RiverSwim has no terminal state, we first thought we had to introduce a terminal condition. However, we learned that the appropriate method was instead to use time-steps instead of episodes when collecting data from different runs of RiverSwim. So we used 10000 time steps for each run. The results we got were first very poor. We then learned that this is because we initialized all Q-values to 0 and for RiverSwim this was a very bad idea. The reason is that it would quickly learn that moving left is the optimal action and would not explore the action of moving right. After we initialized all Q-values optimistically to 5, it always learned the optimal policy. However, as we can see from the reward, the moving average stops improving at about 0.25. The reason is partly due to the nature of the epsilon greedy algorithm, it selects optimal action with 95% chance and selects non-greedy action with 5% chance. However, it is also because moving right in final state only has 60% chance of giving you the reward and 40% chance of moving you back to the state before where you must move forward again. Hence, it makes sense why the moving average does not become so large.

Task 4 - Visualize Q-values and greedy policy

RiverSwim Value Iteration

When we performed value iteration on RiverSwim, the policy we got was indeed the optimal policy. It consisted of always moving right in every state. We know that this gives the highest possible reward.

Optimal Value Function

State	Value
State 0	2.80
State 1	3.05
State 2	3.54
State 3	4.14
State 4	4.85
State 5	5.68

Optimal Policy

State	Action
State 0	Action 1
State 1	Action 1
State 2	Action 1
State 3	Action 1
State 4	Action 1
State 5	Action 1

	State 0	State 1	State 2	State 3	State 4	State 5
Action	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$

Table 1: Optimal Policy with Numerical Actions and Arrow Representation

Q-values

State	Q(0)	Q(1)
State 0	2.67	2.80
State 1	2.66	3.05
State 2	2.89	3.54
State 3	3.36	4.14
State 4	3.94	4.85
State 5	4.61	5.68

FrozenLake Value Iteration

For FrozenLake, we got the following results.

Optimal Value Function

0.18	0.15	0.15	0.13
0.21	0.00	0.18	0.00
0.27	0.37	0.40	0.00
0.00	0.51	0.72	0.00

Table 2: Optimal Value Function

Optimal Policy (0=Left, 1=Down, 2=Right, 3=Up)

0	3	0	3
0	0	0	0
3	1	0	0
0	2	1	0

$\leftarrow$	$\uparrow$	$\leftarrow$	$\uparrow$
$\leftarrow$	-	$\leftarrow$	-
$\uparrow$	$\downarrow$	$\leftarrow$	-
-	$\rightarrow$	$\downarrow$	-

Table 3: Optimal Policy

Q-values

$[0.18, 0.17, 0.17, 0.16]$	$[0.11, 0.11, 0.10, 0.15]$	$[0.15, 0.15, 0.15, 0.14]$	$[0.09, 0.09, 0.08, 0.13]$
$[0.21, 0.15, 0.14, 0.12]$	$[0.00, 0.00, 0.00, 0.00]$	$[0.18, 0.13, 0.18, 0.05]$	$[0.00, 0.00, 0.00, 0.00]$
$[0.15, 0.20, 0.18, 0.27]$	$[0.25, 0.37, 0.29, 0.21]$	$[0.40, 0.35, 0.28, 0.17]$	$[0.00, 0.00, 0.00, 0.00]$
$[0.00, 0.00, 0.00, 0.00]$	$[0.28, 0.39, 0.51, 0.35]$	$[0.52, 0.72, 0.69, 0.62]$	$[0.00, 0.00, 0.00, 0.00]$

RiverSwim TD-Learning

We tried our 4 TD-learning algorithms to see what policy they gave us on RiverSwim. All 4 algorithms gave us the optimal policy of moving right in every state. We observed the following Q-values and policy

Q-Learning

State	Action Left	Action Right
State 0	2.6763	2.6902
State 1	2.7219	2.9364
State 2	2.8076	3.5244
State 3	3.2978	4.0658
State 4	3.8842	4.5735
State 5	4.6025	5.4612

Table 4: Q-values for Actions Left and Right

State	**0**	**1**	**2**	**3**	**4**	**5**
Action	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$

Table 5: Selected Actions for Each State

Double-Q-Learning

State	Action Left	Action Right
State 0	5.47	5.50
State 1	5.65	5.90
State 2	6.07	7.03
State 3	6.72	8.20
State 4	7.76	9.86
State 5	9.23	11.55

Table 6: Q-Values for Double-Q-Learning

State	**0**	**1**	**2**	**3**	**4**	**5**
Action	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$

Table 7: Selected Actions for Each State - Double-Q-Learning

SARSA

State	Action Left	Action Right
State 0	2.05	2.06
State 1	2.12	2.18
State 2	2.29	2.57
State 3	2.52	3.17
State 4	2.99	3.81
State 5	3.56	4.54

Table 8: Q-Values for SARSA

State	**0**	**1**	**2**	**3**	**4**	**5**
Action	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$

Table 9: Selected Actions for Each State - SARSA

Expected SARSA

State	Action Left	Action Right
State 0	2.24	2.24
State 1	2.29	2.37
State 2	2.35	2.65
State 3	2.61	3.18
State 4	3.03	3.79
State 5	3.63	4.62

Table 10: Q-Values for Expected SARSA

State	**0**	**1**	**2**	**3**	**4**	**5**
Action	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$

Table 11: Selected Actions for Each State - Expected SARSA

FrozenLake TD-Learning

Q-Learning FrozenLake

0.17, 0.15, 0.15, 0.15	0.10, 0.10, 0.09, **0.14**	**0.13**, 0.11, 0.12, 0.12	0.08, 0.08, 0.08, **0.11**
0.19, 0.14, 0.13, 0.13	**5.00**, **5.00**, **5.00**, **5.00**	0.12, 0.08, **0.12**, 0.05	**5.00**, **5.00**, **5.00**, **5.00**
0.15, 0.18, 0.18, **0.25**	0.26, **0.34**, 0.27, 0.23	**0.34**, 0.26, 0.25, 0.19	**5.00**, **5.00**, **5.00**, **5.00**
5.00, **5.00**, **5.00**, **5.00**	0.31, 0.34, **0.47**, 0.33	0.53, **0.72**, 0.59, 0.59	**5.00**, **5.00**, **5.00**, **5.00**

Table 12: Q-values for the given states

←	↑	←	↑
←	—	→	—
↑	↓	←	—
—	→	↓	—

Table 13: Optimal Policy for Q-Learning (0=Left, 1=Down, 2=Right, 3=Up)

Double-Q-Learning FrozenLake

0.22, 0.21, 0.20, 0.18	0.12, 0.11, 0.11, **0.21**	**0.22**, 0.17, 0.19, 0.16	0.10, 0.09, 0.10, **0.18**
0.31, 0.19, 0.17, 0.16	**10.0**, **10.0**, **10.0**, **10.0**	0.24, 0.13, **0.28**, 0.08	**10.0**, **10.0**, **10.0**, **10.0**
0.22, 0.27, 0.25, **0.42**	0.37, **0.64**, 0.45, 0.36	**0.75**, 0.58, 0.43, 0.29	**10.0**, **10.0**, **10.0**, **10.0**
10.0, **10.0**, **10.0**, **10.0**	0.47, 0.63, **0.92**, 0.58	0.94, **1.49**, 1.24, 1.09	**10.0**, **10.0**, **10.0**, **10.0**

Table 14: Double-Q-Learning Q-values for the FrozenLake states

←	↑	←	↑
←	—	→	—
↑	↓	←	—
—	→	↓	—

Table 15: Optimal Policy for Double-Q-Learning (0=Left, 1=Down, 2=Right, 3=Up)

SARSA

0.14, 0.12, 0.12, 0.11	0.08, 0.07, 0.06, **0.11**	**0.11**, 0.09, 0.08, 0.08	0.06, 0.06, 0.06, **0.08**
0.18, 0.13, 0.11, 0.09	**5.00**, **5.00**, **5.00**, **5.00**	**0.11**, 0.07, 0.10, 0.04	**5.00**, **5.00**, **5.00**, **5.00**
0.13, 0.18, 0.14, **0.24**	0.20, **0.34**, 0.19, 0.18	**0.36**, 0.29, 0.20, 0.13	**5.00**, **5.00**, **5.00**, **5.00**
5.00, **5.00**, **5.00**, **5.00**	0.27, 0.36, **0.45**, 0.29	0.49, **0.74**, 0.60, 0.52	**5.00**, **5.00**, **5.00**, **5.00**

Table 16: SARSA Q-values for the given states

←	↑	←	↑
←	—	←	—
↑	↓	←	—
—	→	↓	—

Table 17: Optimal Policy for SARSA (0=Left, 1=Down, 2=Right, 3=Up)

Expected SARSA

0.12, 0.12, 0.11, 0.11	0.07, 0.08, 0.07, **0.11**	**0.12**, 0.09, 0.09, 0.09	0.06, 0.06, 0.06, **0.09**
0.15, 0.11, 0.11, 0.08	**5.00**, **5.00**, **5.00**, **5.00**	**0.13**, 0.06, 0.11, 0.04	**5.00**, **5.00**, **5.00**, **5.00**
0.12, 0.15, 0.13, **0.22**	0.20, **0.32**, 0.24, 0.20	**0.29**, 0.25, 0.19, 0.15	**5.00**, **5.00**, **5.00**, **5.00**
5.00, **5.00**, **5.00**, **5.00**	0.26, 0.31, **0.46**, 0.31	0.48, **0.64**, 0.58, 0.50	**5.00**, **5.00**, **5.00**, **5.00**

Table 18: Expected SARSA Q-values for the given states

←	↑	←	↑
←	—	←	—
↑	↓	←	—
—	→	↓	—

Table 19: Optimal Policy for Expected SARSA (0=Left, 1=Down, 2=Right, 3=Up)

Task 5 - Discussion

From looking at the results in Task-4, we see that the policy that the agents get from the 4 different TD-learning algorithms Q-learning, Double-Q-Learning, SARSA, and Expected SARSA are not just similar, but identical. This means that the Q-table that they get have the same relative differences between the Q-values for each state. When looking at the value iteration algorithm, we notice that it is also the exact same policy that it gave us. Hence, value iteration using Bellmann equation gives the same Q-values/policy as TD-methods such as Q-learning. This is impressive because Q-learning does not have any knowledge of the environment, yet it manages to find out the same policy as value iteration which should give the optimal policy under convergence. A counterintuitive action is in state 7 where it looks like it goes left into the terminal state with reward 0. However, we must remember that the environment is slippery which means that is is more likely that it will move up/down when it takes this action. If it instead went straight down, there would be a probability of 2/3 that it went into a terminal state with reward 0. Hence, it makes sense why this is the optimal action in state 7.