# Tabular RL

Simon Svensson, Lukas Martinsson

May 2022

## 1 Introduction

Briefly explains the idea behind the different algorithms.

### 1.1 Q-learning

Q-learning is a method for an agent to learn from practice what to do in an unknown environment. The basis is similar to humans in that rewards are given for good actions and penalties can be given for bad. The agent will then try different actions in different states, partly by exploring and partly by exploiting, to build itself a Q table. The Q table consists of state-action pairs and is updated by the bellman optimality equation when the agent moves.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

The terms needed to calculate Q is the action we take in a state, then we observe the reward it gave and what new state we came to. This series of events is the standard way of interacting with the environment in the gym package. We firstly make an action in a state, see what reward and new state the environment gives and then we update the Q table. The term taking max Q of the next state enables the rewards in the environment to propagate through the Q table one square away at a time. After some time of propagating the rewards, the Q table will have the layout of a path that the agent then follows to optimally gain as much reward as possible.

### 1.2 Double Q-learning

A problem with Q-learning is that due to the max term it will overestimate Q values. This may lead it on the wrong path. A solution to that is double Q-learning which uses two Q tables simultaneously. A coin is flipped to select which one is updated on after each action. Whenever one is updated it gets the max Q of the next state's action from the other Q table. To select an action we sum the Q values from the two tables and select the max Q.

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q_2(S_{t+1}, arg\max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right]$$

$$Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q_1(S_{t+1}, arg\max_a Q_2(S_{t+1}, a)) - Q_2(S_t, A_t) \right]$$

### 1.3 SARSA

SARSA updates the agent's Q table by choosing the action in the next state based on the Q table its own policy. This means that with a probability 1 - $\epsilon$ it will choose the best next state and with a probability $\epsilon$ it will choose a randomly selected next possible state when it updates its table. In comparison, Q-learning updates its Q table based on the max Q of the next state. The difference is small in our case because usually SARSA would also select the best action based on the max Q, but we have a 5% chance of exploring.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

# 2 Results

In this chapter the Q-tables values and their greedy policy choices visualized will be presented as well as the plots of each algorithm for riversvim and FrozenLake-v1 with their average reward over 5000 episodes, where every 20 episodes are aggregate over to get a smoother curve that is easier to analyze/understand.

## 2.1 Code Explanation

Following is a brief explanation of the code: It was made fully automatic by making *for* loops for the 5 runs and the episodes. The rewards are saved in lists and the statistics for the plots are done after the 5 runs. The Q-tables are printed after each run. A reset method was added to the agents to reset their Q-table after each run. Timesteps were set to 100 since that is the limit for FrozenLake-v1. Riverswim was changed slightly so that it is episode based, this enables the run_environment file to run on both environments without any special changes. The changes made was that Riverswim will return done and reset state to 0 after reward was reached.

We have a learning rate of 0.1, discount of 0.95 and 95% greedy. The initial action is randomized over the action space and the Q-tables are initialized to ones.

## 2.2 Riverswim

The results of running the three algorithms on the edited Riverswim environment will be presented here.

| Algorithm | State 0 | State 1 | State 2 | State 3 | State 4 | State 5 |
|---|---|---|---|---|---|---|
| Q-learning | → | → | → | → | → | → |
| Double Q-learning | → | → | → | → | → | → |
| SARSA | → | → | → | → | → | → |

Table 1: Greedy-Policy for algorithms for Riverswim

| State | Move left | Move right |
|---|---|---|
| 0 | 0.70850986 | 0.85868229 |
| 1 | 0.75820011 | 0.93344906 |
| 2 | 0.84042253 | 1.08859325 |
| 3 | 0.98588718 | 1.24367364 |
| 4 | 1.15642274 | 1.51791394 |
| 5 | 1.32170385 | 1.66611617 |

(a) Q-learning

| State | Move left | Move right |
|---|---|---|
| 0 | 0.66846682 | 0.73683313 |
| 1 | 0.68056948 | 0.78362099 |
| 2 | 0.72887305 | 0.94213556 |
| 3 | 0.88025209 | 1.05598964 |
| 4 | 1.01682894 | 1.27170814 |
| 5 | 1.18039068 | 1.47007132 |

(b) SARSA

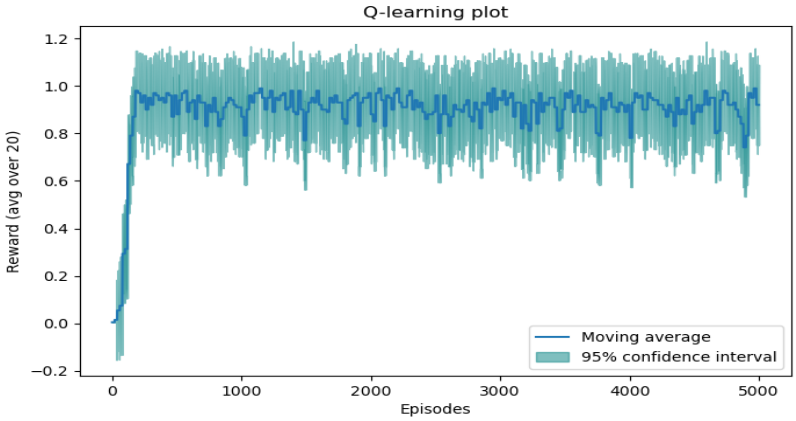
| State | Move left | Move right |
|---|---|---|
| 0 | 0.73939805 | 0.77592476 |
| 1 | 0.74580069 | 0.8568834 |
| 2 | 0.81159694 | 0.95842622 |
| 3 | 0.93267444 | 1.18295362 |
| 4 | 1.09076735 | 1.37874928 |
| 5 | 1.2896986 | 1.65336719 |

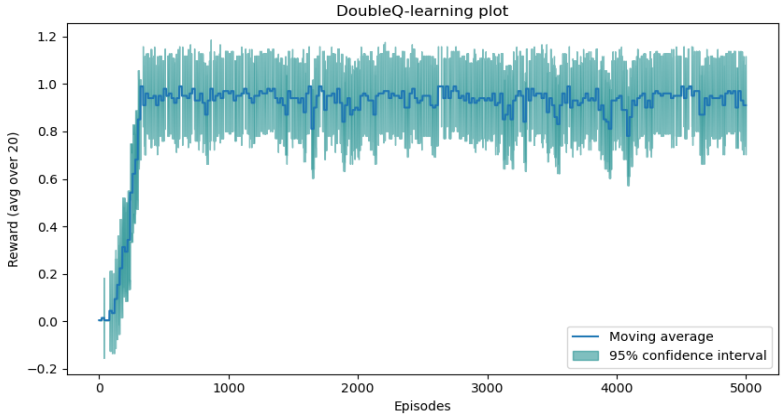
(c) Double Q-learning Q1

| State | Move left | Move right |
|---|---|---|
| 0 | 0.7350237 | 0.78594419 |
| 1 | 0.74508464 | 0.84312175 |
| 2 | 0.80540118 | 0.99319581 |
| 3 | 0.92911881 | 1.1915213 |
| 4 | 1.10477775 | 1.33953358 |
| 5 | 1.27871186 | 1.55179209 |

(d) Double Q-learning Q2

Table 2: Q-tables for Riverswim



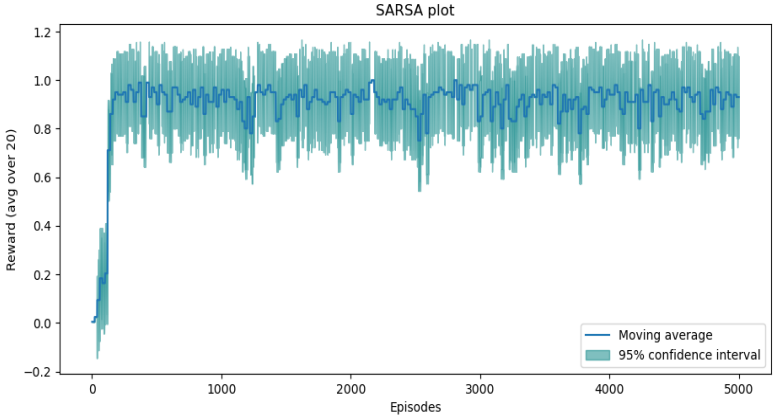




Figure 1: The algorithms average reward on Riverswim

## 2.3 FrozenLake-v1

The results of running the three algorithms on the edited FrozenLake-v1 environment will be presented here.

| | | | |
|---|---|---|---|
| ← | ↑ | ← | ↑ |
| ← | N/A | ← | N/A |
| ↑ | ↓ | ← | N/A |
| N/A | → | → | N/A |

(a) Q-learning

| | | | |
|---|---|---|---|
| ← | ↑ | ← | ↑ |
| ← | N/A | ← | N/A |
| ↑ | → | ← | N/A |
| N/A | ↑ | → | N/A |

(b) SARSA

| | | | |
|---|---|---|---|
| ↓ | ↑ | ← | ↑ |
| ← | N/A | → | N/A |
| ↑ | ↓ | ← | N/A |
| N/A | → | ↓ | N/A |

(c) Double Q-learning Q1

| | | | |
|---|---|---|---|
| ↓ | ↑ | ← | ↑ |
| ← | N/A | ← | N/A |
| ↑ | ↓ | ← | N/A |
| N/A | → | ↓ | N/A |

(d) Double Q-learning Q2

Table 3: Greedy-Policy on FrozenLake-v1

| State | Move left | Move down | Move right | Move up |
|---|---|---|---|---|
| 0 | 0.40064914 | 0.32153688 | 0.31471942 | 0.30848825 |
| 1 | 0.14678771 | 0.15938334 | 0.1628668 | 0.28472515 |
| 2 | 0.32235475 | 0.2147941 | 0.18750563 | 0.20605112 |
| 3 | 0.15267538 | 0.1497825 | 0.08479395 | 0.19626882 |
| 4 | 0.47488145 | 0.35032358 | 0.34630784 | 0.21628076 |
| 5 | 1. | 1. | 1. | 1. |
| 6 | 0.34451965 | 0.22405817 | 0.25176458 | 0.04882525 |
| 7 | 1. | 1. | 1. | 1. |
| 8 | 0.19646297 | 0.44913885 | 0.29974737 | 0.58128441 |
| 9 | 0.57086769 | 0.7849971 | 0.36077462 | 0.43138898 |
| 10 | 0.85696315 | 0.59339487 | 0.28623714 | 0.31444464 |
| 11 | 1. | 1. | 1. | 1. |
| 12 | 1. | 1. | 1. | 1. |
| 13 | 0.30158705 | 0.64618991 | 1.08148757 | 0.69020265 |
| 14 | 0.88469467 | 1.132865 | 1.5787972 | 1.20763321 |
| 15 | 1. | 1. | 1. | 1. |

Table 4: Q-learnings Q-table on FrozenLake-v1

| State | Move left | Move down | Move right | Move up |
|---|---|---|---|---|
| 0 | 0.26657574 | 0.1985088 | 0.19351746 | 0.19636348 |
| 1 | 0.1286632 | 0.17828919 | 0.17977573 | 0.19061179 |
| 2 | 0.19164319 | 0.18194091 | 0.18230748 | 0.18103213 |
| 3 | 0.13209541 | 0.12342435 | 0.11017713 | 0.16364316 |
| 4 | 0.35655241 | 0.19987785 | 0.20524393 | 0.18682605 |
| 5 | 1. | 1. | 1. | 1. |
| 6 | 0.16133692 | 0.11746055 | 0.15720222 | 0.08461779 |
| 7 | 1. | 1. | 1. | 1. |
| 8 | 0.19840711 | 0.27872206 | 0.26818086 | 0.44467392 |
| 9 | 0.39059757 | 0.70671921 | 0.45781757 | 0.2176704 |
| 10 | 0.58467182 | 0.44709891 | 0.3888016 | 0.31171636 |
| 11 | 1. | 1. | 1. | 1. |
| 12 | 1. | 1. | 1. | 1. |
| 13 | 0.57158053 | 0.50985852 | 1.20859962 | 0.53200986 |
| 14 | 0.88070312 | 1.54047368 | 1.04877193 | 1.06400617 |
| 15 | 1. | 1. | 1. | 1. |

Table 5: SARSAs Q-table on FrozenLake-v1

| State | Move left | Move down | Move right | Move up |
|---|---|---|---|---|
| 0 | 0.19204713 | 0.20889903 | 0.20033279 | 0.20218052 |
| 1 | 0.13461089 | 0.11474169 | 0.06853976 | 0.1837443 |
| 2 | 0.19001652 | 0.1236537 | 0.14112642 | 0.13404276 |
| 3 | 0.05492171 | 0.08157825 | 0.10383132 | 0.11834421 |
| 4 | 0.29149653 | 0.18563647 | 0.18379767 | 0.14084941 |
| 5 | 1. | 1. | 1. | 1. |
| 6 | 0.23110552 | 0.11817133 | 0.26377852 | 0.06023529 |
| 7 | 1. | 1. | 1. | 1. |
| 8 | 0.19752395 | 0.29392933 | 0.26239146 | 0.41104936 |
| 9 | 0.36873079 | 0.64120717 | 0.39635102 | 0.31371677 |
| 10 | 0.86394173 | 0.4365607 | 0.28649166 | 0.18029007 |
| 11 | 1. | 1. | 1. | 1. |
| 12 | 1. | 1. | 1. | 1. |
| 13 | 0.31099177 | 0.62287539 | 0.89637171 | 0.4014577 |
| 14 | 0.83518741 | 1.55910244 | 1.03660928 | 0.94773901 |
| 15 | 1. | 1. | 1. | 1. |

Table 6: Double Q-learnings Q1-table on FrozenLake-v1

| State | Move left | Move down | Move right | Move up |
|---|---|---|---|---|
| 0 | 0.21091611 | 0.22336654 | 0.20253211 | 0.1995353 |
| 1 | 0.10562618 | 0.13482299 | 0.11053185 | 0.1822751 |
| 2 | 0.20229992 | 0.15069463 | 0.14453624 | 0.13370323 |
| 3 | 0.06257944 | 0.09560445 | 0.06063838 | 0.11498082 |
| 4 | 0.23753118 | 0.22404412 | 0.16943955 | 0.12847348 |
| 5 | 1. | 1. | 1. | 1. |
| 6 | 0.26402663 | 0.10687392 | 0.18663469 | 0.0895184 |
| 7 | 1. | 1. | 1. | 1. |
| 8 | 0.22164889 | 0.19262856 | 0.25904805 | 0.40866262 |
| 9 | 0.45227013 | 0.60361126 | 0.37053335 | 0.30495329 |
| 10 | 0.86961137 | 0.69539121 | 0.2982985 | 0.31304794 |
| 11 | 1. | 1. | 1. | 1. |
| 12 | 1. | 1. | 1. | 1. |
| 13 | 0.4156985 | 0.5558849 | 0.9239483 | 0.43389915 |
| 14 | 0.81391335 | 1.42854654 | 0.86774565 | 1.03469893 |
| 15 | 1. | 1. | 1. | 1. |

Table 7: Double Q-learnings Q2-table on FrozenLake-v1

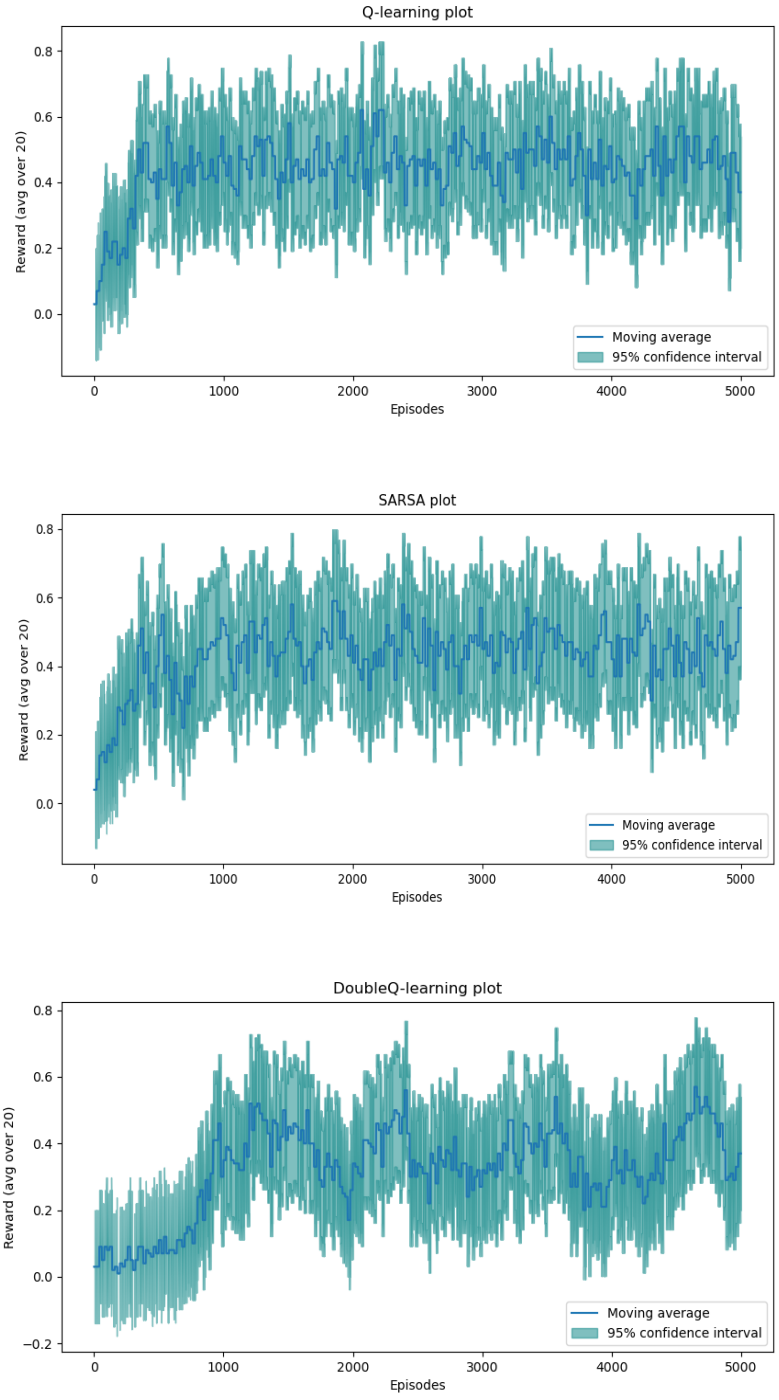



Figure 2: The algorithms average reward on FrozenLake-v1

# 3 Conclusion

The optimal policy for Riverswim is to always go right. This is due to the fact the the large reward is significantly higher then the small reward. All algorithms reaches the optimal policy.

The optimal policy for FrozenLake-v1 is more complicated due to the sliding. According to its documentation, there is a 2/3 chance to slide perpendicular to the given direction. To deal with this problem, the algorithms has learned to move directly away or towards the holes and walls. For example, in state 1 we see an upwards arrow indicating a move into the upper wall, but in fact it is the optimal move in that state since it guarantees it won't fall into the hole beneath. The optimal policy for FrozenLake-v1 can be seen below.

| | | | |
|---|---|---|---|
| $\leftarrow$ | $\uparrow$ | $\leftarrow$ | $\uparrow$ |
| $\leftarrow$ | N/A | $\leftrightarrow$ | N/A |
| $\uparrow$ | $\downarrow$ | $\leftarrow$ | N/A |
| N/A | $\rightarrow$ | $\rightarrow$ | N/A |

Table 8: Optimal Greedy-Policy on FrozenLake-v1

By comparing the optimal greedy policy to the algorithms one can see that Q-learning reaches the optimal policy while SARSA and Double Q-learning almost reaches the optimal policy (whether this is always the case or if this varies deepening on runs is hard to say). Hence all algorithms stabilize with an average reward of slightly over 0.4. However, Double Q-learning take a longer time to stabilize then Q-learning and SARSA. This is probably since double Q-learning has a harder time comprehending random actions. Hence probably double Q-learning suffers more if the $\epsilon$ value was higher compared to Q-learning or SARSA.

Double Q-learning stabilizes last out of the three algorithms. Something that is mentioned from the lecture is that it learns much faster, which is contrary to the figures illustrated above (the reason for this is briefly explained above). Regardless, the difference is negligible in the case of these simple environments where the learning only takes a matter of seconds. For environments used in this paper there is not really any perks to use the unbiased and quick learning of double Q-learning compared to Q-learning and SARSA. Perhaps if the small reward in Riverswim was higher it would trick normal Q-learning to go for the small reward for a longer period of time in the training sessions.

When it comes to initialization values zeros, ones, and random between 0 and 1 were tested. The paper won't showcase all the results since it would result in a large number of figures/tables that will only illustrate a simple principle. Zeros

leads to zero reward and only zeros in Q-tables. This is because the values will never go negative in the Q-table because there are not any negative rewards. Therefore, the max Q action is the first one in the list which is always to move left which makes it stuck at the start. It will only explore by chance of 5% which is extremely slow so it will eventually work but it will take more steps and/or episodes. Since the environment resets after 100 steps, it is extremely unlikely to even find the goal once. In general, initializing the Q-table to zeros will not encourage exploration, at least in environments where there is only rewards and no penalties. If however Q-learning could use argmax and then randomize the action chosen if there are several actions that are equal and if there would be a penalty for going into the ice, an initialization of zeroes for the Q-table could work. Random initialization works and ones work similarly well since it emphasizes exploration of each state which will result in the Q-table reaching the reward after a certain number of steps and being to back-propagate until an optimum has been reached if given enough episodes.