UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR ROBOTICS
AND COGNITIVE SYSTEMS

Course: Reinforcement Learning (SS2020)
Dr. Christoph Metzner & Honghu Xue, M.Sc.
Assignment Report Nr. II
Date: 26th May 2021

Team Nr. 1
Rakesh Reddy Kondeti
Christopher Schmale
Tim-Henrik Traving
Contact: timhenrik.traving@student.uni-luebeck.de

1 MC & TD Theory

1.1 Difference between DP and MC/TD

MC & TD sample an expectation, while DP does a full backup instead. Furthermore, MC & TD are methods for model-free prediction and can therefore be used when the underlying MDP is unknown. For DP the underlying model needs to be known.

Dynamic Programming: Breaking a large problem down into incremental steps so optimal solutions to sub-problems can be found at any given stage. A model (mathematical representation of real world) is known. So, Bellman equation can be applied to calculate the optimal values of each state.

Monte-Carlo or Temporal Difference: However, we almost never have all of this information of the real world and hence the model is unknown. In this case, our agent must learn from the environment by interacting with it and collecting experiences, or samples. This is where MC/TD is applied.

1.2 On- & Off-Policy Learning

On-policy methods learn about the policy π from experience sampled from this policy π. In other words, it is trying to evaluate π while following the same policy π.
In on-policy learning the Q(s,a) function is learned from actions that we take using our current policy $\pi(a|s)$.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)) \tag{1}$$

where s is the current state, a is the current action, α is the step size, r is the immediate reward, γ is the discount factor, s' is the next state, a' is the next action depending upon the state s'.
An example for on-policy learning is SARSA.

Off-policy methods learn about policy π from experience sampled from another policy μ. In other words, it is trying to evaluate π while following another policy μ. The policy used to sample experience from is called the *behavior policy* (in this case μ) and the policy that is learned is called the *target policy* (in this case π).
In off-policy learning, the Q-function is updated depending upon the best future state-action pair, which do not necessarily have to follow the agents current policy.

$$Q(S_t,A_t) \leftarrow A(S_t,A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1},A') - Q(S_t,A_t)) \tag{2}$$

In off-policy, we don't have to follow any specific policy. The agent can act randomly to explore. In on-policy methods depends on the policy used.
One simple example for Off-Policy learning algorithm is Q learning

1.3 Stationary- & Non-Stationary Policy

A stationary policy is time-independent, which means that the same action will be chosen for a state, regardless of the time:

$$A_t \sim \pi(\cdot|S_t), \forall t > 0. \tag{3}$$

However, when dealing with non-stationary problems, it might be favorable to have a non-stationary policy, which adapts the action for a state with regard to the time. One example of such an non-stationary policy could be a running mean

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t). \tag{4}$$

In this scenario, the step size α describes how much a new return G_t influences the V or Q value on an update. In other words, α describes how confident we are that the new return accurately represents the true, underlying value. Therefore, in the stationary case, the α can be set e.g. to 1, while in the non-stationary case, α should be rather low, because the „true, underlying value" will probably be different than the recently collected experience.

1.4 Conditions for Convergence

Besides proper choice of step size α and discount factor γ, each state action pairs should be visited an infinite amount of times, as time goes towards infinity (see GLIE). Furthermore, an infinite number of episodes has to be assumed for the policy evaluation, as the Q-function is estimated based on the averages of the MC returns.

1.5 Other Exploration Strategy

We selected the *upper confidence bound* (UCB) action selection exploration strategy for further investigation. Instead of randomly selecting an action to explore the state-action-space, UCB selects the action that has the highest potential of being optimal. This is done as follows:

$$A_t = \arg\max_a \left(Q_t(s,a) + c\sqrt{\frac{\ln t}{N_t(s,a)}} \right) \tag{5}$$

This equation takes into account how close the the estimated action value $Q_t(s,a)$ is to the maximal action value, combined with the uncertainty of the estimation (expressed by the square-root term). The c parameter can be used to control the degree of exploration and acts as a confidence level. N_t denotes how many times the action a was chosen up until timestep t.
Each time an action a is selected, $N_t(s,a)$ is incremented and the uncertainty of a is thus reduced. Each time an other action a' is selected, t is incremented, but not $N_t(s,a)$, thus the uncertainty of a is increased. As time progresses, it becomes more and more unlikely that a non-maximal action is selected. However, as the increase of the natural logarithm only gets smaller over time, but is still unbounded, it is still possible that non-maximal actions are explored even as time progresses (or, in other words: it is *not* impossible that exploration occcurs even in the later learning phases). The term that is maximized over thus acts as a upper bound for the value of action a.

2 Programming

2.1 On-policy first-visit MC Control

The *on-policy first-visit Monte Carlo control with ϵ-greedy exploration* strategy is implemented in the "`MC_frozen_lake.py`" file.
The final learned policy is shown in figure 1. It took $\sim$ 38.76 seconds to compute this policy. It takes an average of 38 steps to get to the goal while the agent falls into a hole 25.2% of the time, based on 100 episodes.

←	↑	←	↑
←	←	→	←
↑	↓	←	→
↓	→	↓	↓

Figure 1 Monte Carlo Result

2.2 SARSA

The figure 2a shows the results of the *SARSA* learning. There are three different results stated in three different colors, one color per group member. It takes around 405 seconds to train the policy and it converges to a average reward of -150.

2.3 Q-Learning

The figure 2b shows the results of the *Q-Learning*. There are three different results stated in three different colors, one color per group member. It takes around 279 seconds to train the policy and it converges to a average reward of -150.

3 Going Deeper

3.1 Comparison with Policy-/Value-Iteration

A comparison of the learned policies can be seen in table 1. The general layout of the frozen lake can be seen in table 1a (S = Start, G = Goal, F = Frozen, H = Hole). Besides from some of the terminal states, the MC policy differs from the VI/PI policy in states 2 and 6 (first & second row of the third column). However, in practice this is less of a difference than one might think.
The policy for state 6 (second row, third column) recommends to go left in case of the VI/PI approach, while MC recommends to go right. As both states 5 and 7 to the left or right of state 6 are holes, it does not matter for the overall success of the policy in which direction the agent goes. In both cases, the probability of falling into a hole is 1/3.
The other difference is in state 2 (first row, thrid column). While the VI/PI policy recommends to move upwards, the MC policy urges to move left. This is due to the ordering of the different actions:

0. Left
1. Down
2. Right
3. Up

When the VI/PI greedely determines the best action in the initial state, it goes through the different actions from 0 to 3. The first option, 0/left provides the greatest return, therefore this option is chosen. From there on, the only way to proceed is downwards. MC on the other hand chooses the actions with some random influence. Therefore, instead of moving downwards along the optimal path, an agent might move to the right. By chance the agent might end up in state 2, and from here on it is better to move to the left (with a 1/3 chance of ending up one row below, in state 6), than to move upwards, because moving up might cause the agent to actually move to the right, which is even further away from the goal than the current state 2.

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

(a) Map of the frozen lake

←	↑	↑	↑
←	←	←	←
↑	↓	←	←
←	→	↓	←

(b) VI/PI Result

←	↑	←	↑
←	←	→	←
↑	↓	←	→
↓	→	↓	↓

(c) Monte Carlo Result

Table 1 Comparison of the different policies. Differences are marked in red.

3.2 Different ϵ

A comparison of the learned policies can be found in table 2. With the new policy, it took 45 steps on average to reach the goal, failing 48.6% of the time, based on 100 measurements. Even though this only takes 7 more steps than compared to the original policy with $\epsilon = 40\%$, it is almost only half as reliable. Furthermore, the training time was more than doubled, taking ~ 81.24 seconds instead of ~ 38.76 seconds. Therefore the new policy is obviously worse than the original policy that was derived with $\epsilon = 40\%$
This is also easy to see in the policy itself. When starting, the 5% version recommends to move right. From here on however, it is impossible to move any further down, towards the goal, due to the transition dynamics of the environment. An easy improvement would be to move to the left, when starting in the initial state. This would prevent the agent from getting lost in the upper row. Yet, with an ϵ of only 5%, the algorithm did not discover this, which illustrates that with a too small ϵ, the algorithm might get caught in a local optimum, or will at least take longer (on average) to converge towards an optimal policy.

→	↑	↑	↑
←	←	→	←
↑	↓	←	→
↓	→	↓	↓

(a) $\epsilon = 5\%$

←	↑	←	↑
←	←	→	←
↑	↓	←	→
↓	→	↓	↓

(b) $\epsilon = 40\%$

Table 2 Comparison of different epsilons. Differences are marked in red.

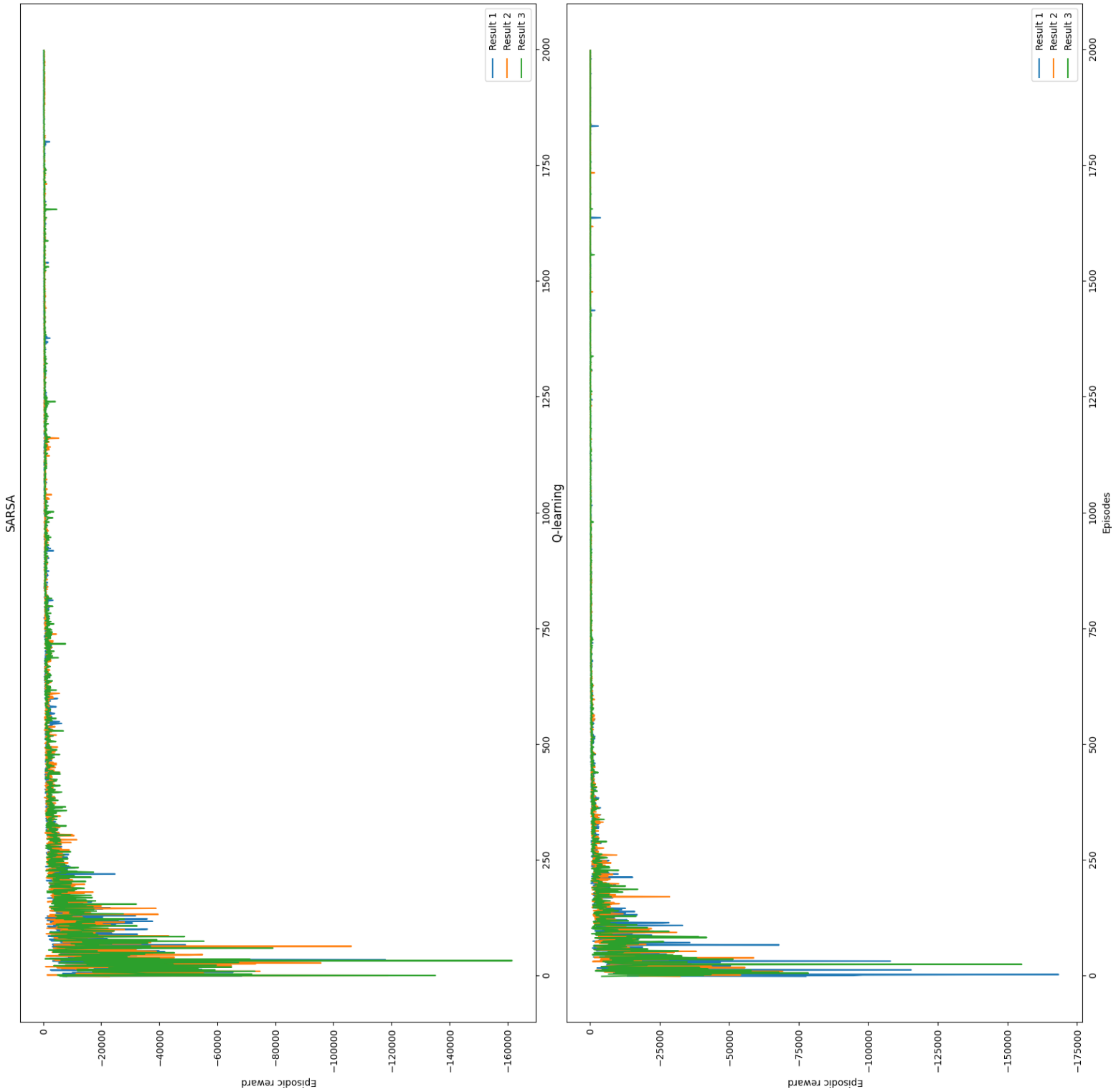
3.3 Different Step Size

The final learned policy has more stable rewards with a different step size of $\alpha = 1/N_{visit}$ so that the vehicle behaviour is nearly the same every test. Moreover, you can observe that the policy is more effective to reach the goal since the rewards are better with a different step size.
Theoretically, a different step size of $\alpha = 1/N_{visit}$ will converge to a near optimal policy, because the step size will be more detailed. Therefore, the policy can be optimized to a better result.
Theoretically, Q learning algorithm always converge to a near-optimal action. The initial Q-value has an effect on the time it will converge to the near-optimal policy. When the Q-value is initialised closely to the near-optimal value, it will take less time (and episodes) to converge to the near-optimal policy. When a poor initial Q-value is chosen it needs to be trained longer to converge to near-optimal policy.

3.4 Optimistic Values

The purpose of optimistic initial values is, that the agent will be encouraged to explore in the beginning. When the initial value is higher than the reward, it is wildly optimistic. Whichever action the agent performs, the received reward is less than the value estimates and as a result, it will perform other actions. This is mostly noted at the beginning. Negative initial values will be better because in the end we get a negative reward. If we would get positive rewards it would be the other way around.

- Initial value 0: This has no effect on the result.
- Initial value 50: The rewards will be worse in the beginning and the finial learned policy is not as good. Not valid.
- Initial value -50: The policy will converge sooner to the near-optimal action, so that the policy after 2000 episodes will be better. Valid.
- Initial value -5: Same effect as -50, but not as strong, but still valid.
- Initial value 5: Same effect as 50, but not as strong, but still not valid.



(a) Team results of SARSA learning

(b) Team results of Q-learning

Figure 2 The team results of the mountain car tests.