UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR ROBOTICS
AND COGNITIVE SYSTEMS

Course: Reinforcement Learning (SS2020)
Dr. Christoph Metzner & Honghu Xue, M.Sc.
Assignment Report Nr. III
Date: 16th June 2021

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

1 Theoretical understanding

1.1 Differences

1.1.1 one-step & multi-step TD

In the one-step on-policy TD case, the q-value for SARSA is updated as follows:

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

where s and a are the current state and current action respectively. Then next step state and action are s and a and the immediate reward is r.The update of one-step TD method, is based on just the one next reward, bootstrapping from the value of the state one step later as a proxy for the remaining rewards.

In the n-step on policy TD case, the q-value is updated as follows:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left(q_t^{(n)} - Q(S_t,A_t)\right), \tag{2}$$

where $q_t^{(n)}$ is the n-step Q-return:

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}). \tag{3}$$

It is easy to see that the one-step on-policy TD algorithm q-value update rule is just a special case of the n-step one where $n = 1$.

1.1.2 n-step & λ-step return

The n-step return is defined as the sum of the first n rewards plus the estimated value of the state reached in n steps, each appropriately discounted by the discounted factor γ. The n-step return is defined as:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}), \qquad 0 \le t \le T - n, \tag{4}$$

where T is the time of episode termination.

The λ-*return* is a weighted combination of all n-step returns up to the nth-step. Each weighted term is proportional to λ^{n-1}, and is normalized by the factor of $(1-\lambda)$ to ensure the weights sum to 1. The resulting update is called λ-return:

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}, \qquad \lambda \in [0,1]. \tag{5}$$

1.2 Importance Sampling

1.2.1 Definition

Importance Sampling ratio ($\rho_{t:h}$) is the relative probability under the two policies π and b, of taking n actions from A_t to A_{t+n-1}.

$$\rho_{t:h} = \prod_{k=t}^{min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{6}$$

In case of on-learning policy, the two policies (π and b) are same and *importance sampling ratio* is always 1. To make a simple off-policy version of n-step TD, the update for time t can simply be weighted by using $\rho_{t:h}$. The value at time $t+n$ is:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha\rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)], \tag{7}$$

1.2.2 Relation of IS Ratio and Q-Value Update

Importance Sampling is a method to perform off-policy learning, where the expected return of samples drawn from a policy P is estimated through the expected return of samples drawn from another policy Q: The n-step SARSA update in off-policy form:

$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1:T+n-1}(G_{t:t+n} - Q_{t+n-1}(S_t, A_t)). \tag{8}$$

1.2.3 Potential Problems

Importance Sampling increases the variance, which already high when using MC/Off-Policy methods. This renders the method unusuable, as the high variance needs to be compensated with a small step size. As a result, the overall speed of the algorithm is reduced.

1.3 Bias-Variance Tradeoff

In most Reinforcement Learning methods, the goal of the agent is to estimate the state value function $v_\pi(S_t)$ or the action value function $q_\pi(S_t, A_t)$. Considering state value function, $v_\pi(S_t)$, it is defined the expected total amount of rewards received by the agent from that state S_t until the end of the episode. Mostly, the agent cannot predict this value accurately and hence we need an estimate of these values. During this estimate, a *bias* and *variance* is introduced.
Let $V_\pi(S_t)$ be the estimate of true value of $v_\pi(S_t)$. Then it is updated with the formula:

$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha(Target - V_\pi(S_t)) \tag{9}$$

On every update, $V_\pi(S_t)$ takes a step towards the *Target* with the learning rate α.

Monte Carlo RL algorithms estimate returns by running full trajectories and literally averaging the return achieved for each state. In Monte-Carlo method, the return G is used as the target($G_t = R_{t+1} + \gamma R_{t+2} + ...$). i,e., $v_\pi(S_t) = \mathbb{E}\ (G_t)$. This method has a very few assumptions and thus have a low bias. However variance is high since each estimate depends on the full literal trajectories that we observe. So one needs many trajectories to get a good estimate of the value function.

In TD method, instead of waiting until the end of the episode, the Target is the sum of immediate reward and the estimate of future rewards.($Target = R_{t+1} + \gamma V_\pi(S_{t+1})$). We are updating $V_\pi(S_t)$ using the same function $R_{t+1} + \gamma V_\pi(S_{t+1})$. Using V this imposes some assumptions and hence some bias (initialization of V at the beginning of the training effects the next estimates). But this method has low variance.

Generally, to reduce the variance, the *bias* can be increased by introducing assumptions. If, however, the assumptions are false, the estimated values may well have a low variance, but still be imprecise, for the true values (that are to be estimated), may be in a completely different range which has been cast aside due to the introduced assumptions. So, to achieve good results, one has to trade the variance of the estimated values off

Method	Bias	Variance
One-step TD Learning	Medium	Low
10-step TD Learning	Low	Medium
Monte-Carlo Learning	None	High

Table 1 Comparison between the bias-variance tradeoff of different methods

against the bias of initial assumptions.

Monte-Carlo Methods have no bias, as they rely on the return G_t, which is a unbiased estimate of $v_\pi(S_t)$. The TD approaches on the other hand use the TD target, which is a biased estimate of $v_\pi(S_t)$. The 10-step TD learning approach is less biased than the One-step approach, because the 10-step lookahead uses the true, unbiased values of those states before estimating the value of the remaining states, while the one-step approach not only estimates the value of the remaining states, but of the previous nine steps, too.
But because the TD target of the one-step approach depends only on one random action, transition and reward, it has a comparatively low variance. The more steps are considered, the greater the variance becomes, as more and more random actions, transitions and rewards influence the final estimate. The return used for the Monte-Carlo approach depends on all random actions, transitions and rewards until the end of an episode. MC approaches therefore have the greatest variance.

1.4 Relation of λ-Value and Target Update

Monte-Carlo and SARSA are special cases of n-step TD learning, where as Monte-Carlo is similar to infinite-step, and SARSA to one-step TD learning. Equations 2, 3, 5 state how to update Q-values in n-step TD learning. As a result, if $\lambda = 1$ than the equations will result to Monte-Carlo (n-step TD learning) learning and if $\lambda = 0$ the equations will result SARSA (one-step TD learning).
Eligibility traces have arithmetically advantages over n-step concepts, but not on one-step concepts. Following advantages are offered by eligibility traces to n-step:

- There is no need to wait for n-steps, learning has a direct effect on all values.
- The storage of the last n feature vector is not necessary. Just a single trace vector is stored. i,e., smaller memory requirements
- Continuous learning, not at the end of an episode.

With tuning the hyper-parameters λ and α can result in nearly the same estimates. Equation 10 defines accumulating traces and equation 11 defines replacing traces.

$$E_t(s,a) = \begin{cases} \gamma \cdot \lambda \cdot E_{t-1}(s,a) + 1, & if(S_t = s, A_t = a) \\ \gamma \cdot \lambda \cdot E_{t-1}(s,a), & \text{otherwise} \end{cases} \tag{10}$$

$$E_t(s,a) = \begin{cases} 1, & if(S_t = s, A_t = a) \\ \gamma \cdot \lambda \cdot E_{t-1}(s,a), & \text{otherwise} \end{cases} \tag{11}$$

The above equations includes the trace ($E_t(s,a)$) in time step t and the associated state s and action a, current state s_t, discount factor γ and bootstrap factor λ. In a accumulating trace, the eligibility trace get increased by 1 per visit, which means that the eligibility increases constantly with frequent visits. In contrast, the value is only set to 1 in the replacing trace.

1.5 Forward- & Backward-View

1.5.1 Equivalence of Forward- & Backward-View

The forward view of TD(λ)/SARSA(λ) is equivalent to the backward view for off-line updating. For on-line updating, forward view is not equivalent to backward view. But for smaller α, the forward view is similar to backward view.
In forward view algorithm it looks forward from updated state.It depends on all future rewards till the end of an episode Update of the q value with the forward view algorithm: q value at current time step n is $q_n(s_0, a_0)$

$$q_{n+1}(s_0, a_0) = q_n(s_0, a_0) + \alpha[G_n^\lambda - q_n(s_0, a_0)] \tag{12}$$

In backward view algorithm it goes back to the last visited states and the current states gets updated. Update of q value with the backward view algorithm:

$$q_{n+1}(s_0, a_0) = q_n(s_0, a_0) + \lambda[R_{n+1} + \gamma q_n(s_{n+1}, a_{n+1}) - q_n(s_n, a_n)]E_n(s_0, a_0) \tag{13}$$

1.5.2 Estimation of $q(s_0, a_0)$

See formular 12 and 13.

1.5.3 Dutch Eligibility Trace

The dutch eligibility trace is defined as

$$E_t(S_t) = (1 - \alpha)\gamma\lambda E_{t-1}(S_t) + 1. \tag{14}$$

The dutch eligibilty trace becomes the accumulation trace if $\alpha = 0$ and turns into the replacing trace if $\alpha = 1$.

1.6 Explanation of Terms

1.6.1 Sample Update

Sample update requires several transitions or even a full episode in order to perform the update.
E.g.: SARSA, TD(0) and Q-learning

1.6.2 Expected Update

Expected update is mostly based on distributions of all possible states.
E.g: Dynamic programming (policy and value iteration) and n-step tree backup.

1.6.3 Bootstrapping

A method is *bootstrapping*, if it does not learn from complete episodes, but from incomplete ones. Examples of such methods are Dynamic Programming (Value- & Policy-Iteration) or Temporal Difference Learning (SARSA, Q-Learning) and TD(0).

1.6.4 Non-Bootstrapping

A method is *non-bootstrapping*, if it learns only from complete episodes. In order to use such methods, the episodes obviously need to terminate at some point. An example for such a method is Monte-Carlo Learning (e.g. with ϵ-greedy exploration) and TD(1).

2 Programming

2.1 SARSA(λ) with Replacing Trace

The figure 1 shows the episodic returns using SARSA(λ) with $\lambda = 0.95$. For better visibility all values are smoothed.

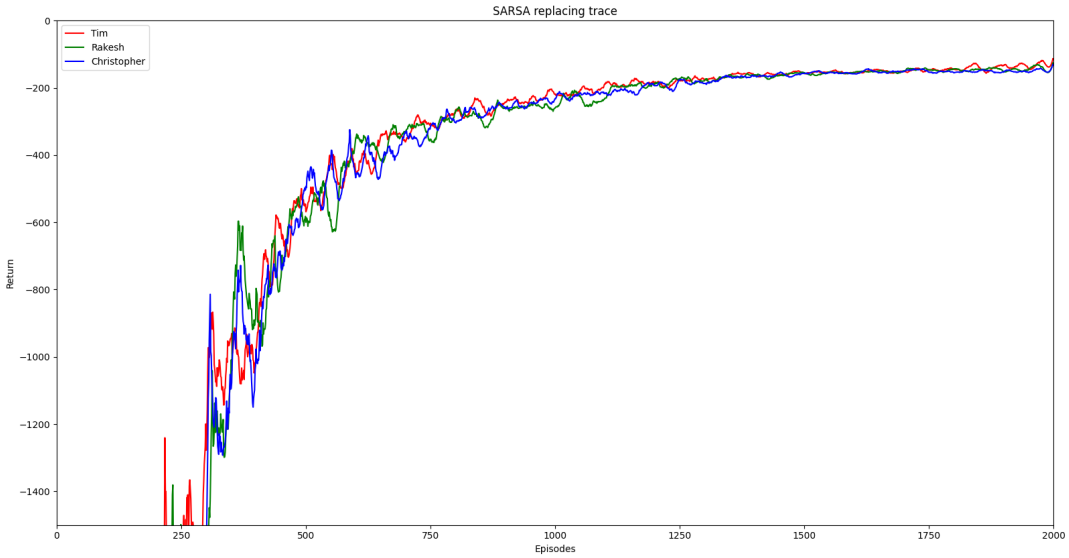


Figure 1 Different returns of episodes. Made with SARSA with replacing trace

2.2 Effect of λ-Values

The different Q matrices are shown in figure 2. The Q-values are a little higher than the Q*-values generally higher, because of the initialization of 10000. Futhermore, higher λ-values return more differentiated Q values. This can be seen in the color differences in figure 2. Smaller λ-values return neighborhoods of Q-values, which are closer together.

3 Bonus

3.1 Categorization of Algorithms

See table 2 for a categorization of the different algorithms.

3.2 Watkin's Q(λ) algorithm

In figure 3a the episodic returns of the Watkin´s Q(λ) during training are shown. For better visibility all values are smoothed.

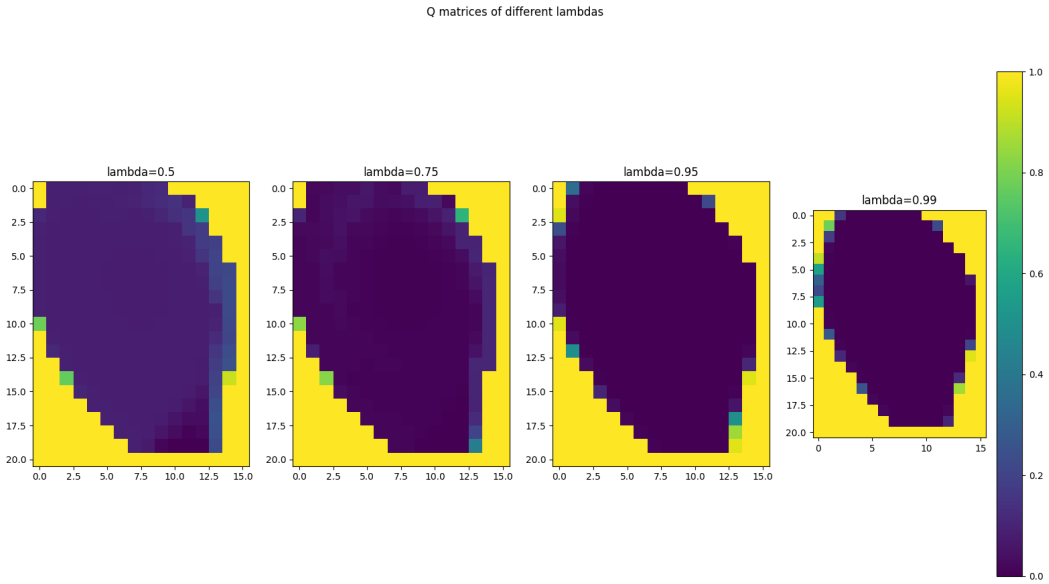


Figure 2 Different Q-values. Made with SARSA with replacing trace and different λ

Algorithm	Update	Bootstrapping	On-/Off-Policy	Model
Expected SARSA	Sample until last step, then Expected	Bootstrapping	Both	Model Free
n-step Tree Backup	Expected	Bootstrapping	Both	Model Free
Watkin's Q(λ)	Sample until last step, then Expected	Bootstraps on first non-greedy action	Off-Policy	Model Free
Q(σ)	Both (depends on σ)	Bootstrapping	Both	Model Free

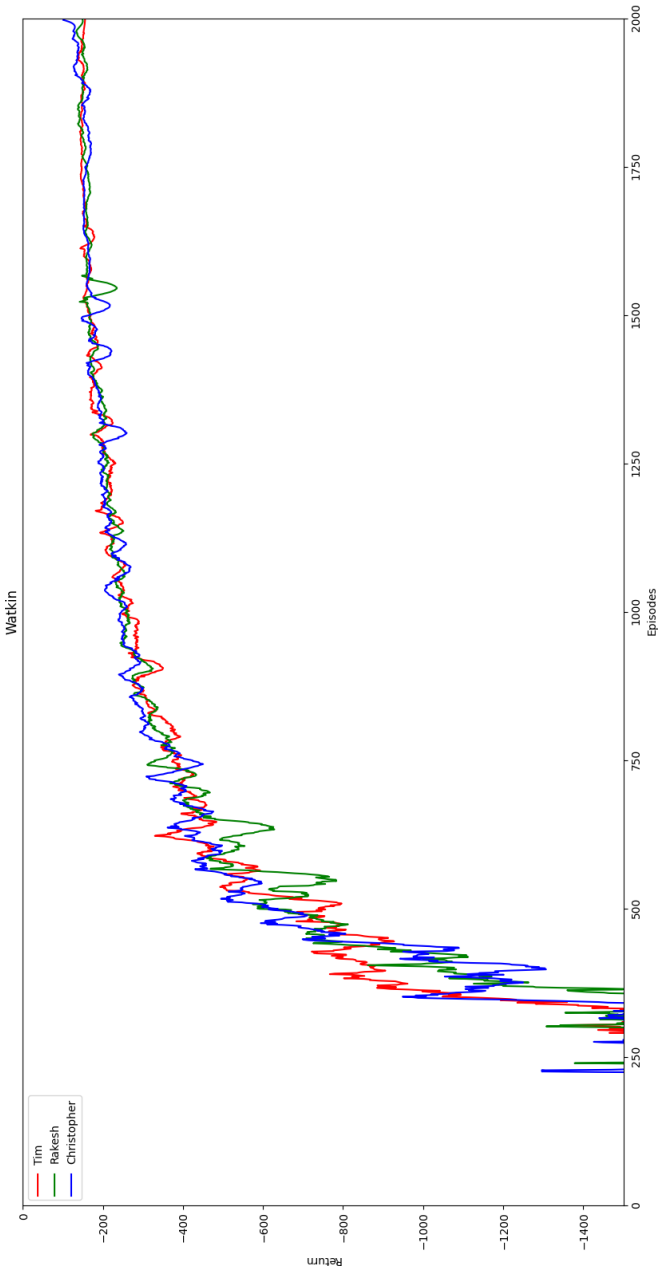
Table 2 Categorization of different algorithms

3.3 True-Online SARSA(λ)

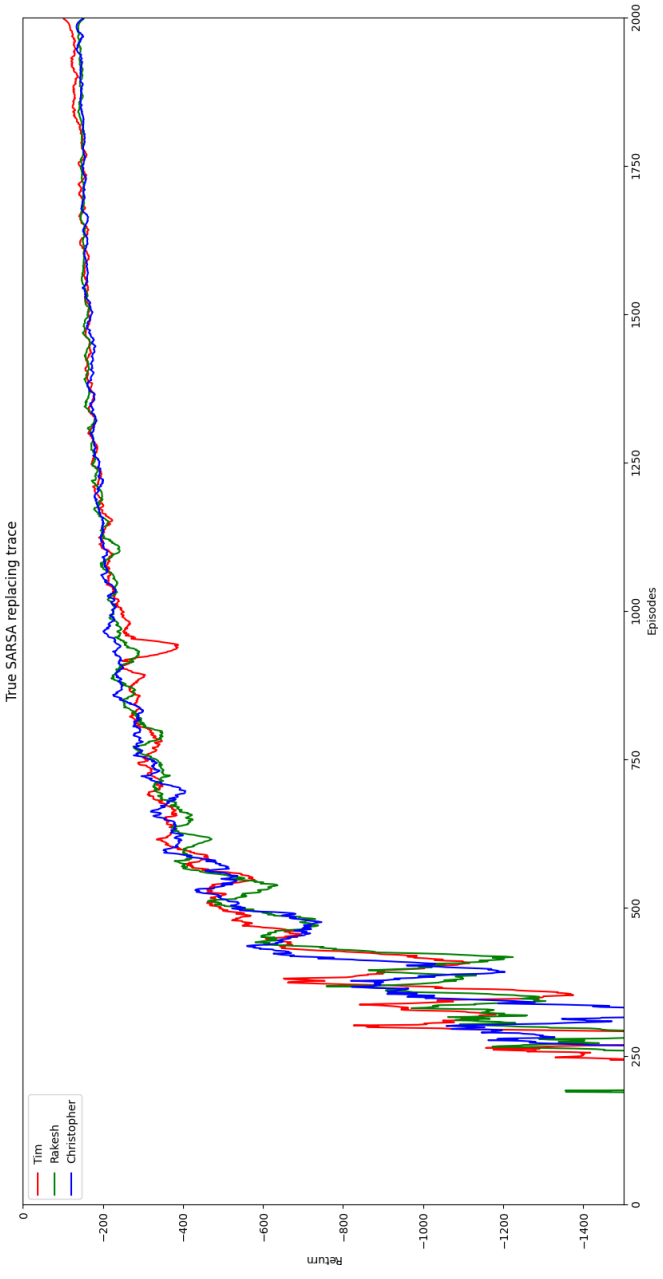
In figure 3b the episodic returns of the true online SARSA(λ) during training are shown. For better visibility all values are smoothed. The Q-values are calculated with equation:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \cdot (\delta_t + Q_t(S_t,A_t) - Q_{t-1}(S_t,A_t)) \cdot E_t(s,a) - \alpha I_{sSt} \cdot (Q_t(S_t,A_t) - Qt-1(S_t,A_t)) \quad (15)$$

$$I_{sS_t} = \begin{cases} 1, & s = S_t \\ 0, & \text{otherwise} \end{cases} \quad (16)$$



(a) Different returns of episodes. Made with Watkin´s Q(λ)



(b) Different returns of episodes. Made with true online SARSA)

Figure 3 The team results for the different algorithms.