UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR ROBOTICS
AND COGNITIVE SYSTEMS

Course: Reinforcement Learning (SS2020)
Prof. Dr. Elmar Rueckert & Honghu Xue, M.Sc.
Assignment Report Nr. III
Date: 11. Juni 2020

Team Nr. 1
Matthias Klatt
Abby Kandathil Abraham
Suneesh Omanakuttan Sumesh Nivas
Sameed Quais
Lakshith Nagarur Lakshminarayana Reddy
Contact: matthias.klatt@student.uni-luebeck.de

Task I, q.1:

(1:) The difference in the update rule for the q-values is the definition of the return (also called target). For one-step TD we have the return defined as:

$$G_{t:t+1} = \mathcal{R}_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}),$$

for the n-step return:

$$G_{t:t+n} = \mathcal{R}_{t+1} + \gamma\mathcal{R}_{t+2} + \cdots + \gamma^{n-1}\mathcal{Q}_{t+n-1}(S_{t+n}, A_{t+n}), n \geq 1, 0 \leq t < T - n$$

with $G_{t:t+n} = G_t$, if $t + n \geq T$

The Q-matrix is updated with:

$$\mathcal{Q}_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - \mathcal{Q}_{t+n-1(S_t,A_t)},$$

with: $\mathcal{Q}_{t+n}(s, a) = \mathcal{Q}_{t+n-1}(s, a), \forall s, a$ such that $s \neq S_t \vee a \neq A_t$

so if we choose: $n = 1$we get the 1-step TD update.

So as you can see the different is the definition of the return.
(2:) We have defined the n-step return in the equation above, now we want to clarify the difference between the n-step return and the λ-return, to do this we define first the λ-return, with $\lambda \in [0, 1]$:

$$\begin{aligned} G_t^\lambda &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n} \\ &= (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}\cdot G_{t:t+n} + \lambda^{T-t-1}G_t \end{aligned}$$

So if $\lambda = 1$ we have a MC return and if $\lambda = 0$ we will get a 1-step TD return, but more important is we can average with the λ-return n-step returns with different values for n for one return. So we don't have to work with a fixed n-value for one return we can have different values for n and average them to get one return. We weight the nth return with λ^{n-1}, and we want all of these weights to sum to one, so we need to normalize them with the normalization constant $(1 - \lambda)$. Different values of λ affect the initial value of a return and the way this value decays over time. Bigger values of λ lead to slower decay (information from the past is given a non-negligible importance).

Task I, q.2:

In Reinforcement learning variance refers to a noisy but an average accurate value estimate, whereas bias refers to a stable, but inaccurate value estimate, so we are talking about the bias and the variance of the target values. Bias and variance can slow down the agent's learning.
One-step TD learning has a low variance but because of the estimation at each time step, we have a lot of bias in the beginning of the training phase and as the estimate improves, the bias subsides.
MC approaches have no bias but a high variance.
A 10-step TD learning does a trade-off between this extremes, it has some variance and some bias. In general we can say if the step size low we estimate more and the estimator is because of that more biased but have a low variance, while a high n-value TD learning approach will have a lower bias but a higher variance.

Task I, q.3:

For a simple off-policy n-step TD update rule for the V-value is defined as:

$$V_{t+n} = V_{t+n-1}(S_t) + \alpha\rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)], 0 \leq t \leq T,$$

with the so called importance sampling ration

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

For the off-policy SARSA the update rule for the Q-value (state-action pair value) is defined as:

$$\mathcal{Q}_{t+n}(S_t, A_t) = \mathcal{Q}_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1:t+n}[G_{t:t+n} - \mathcal{Q}_{t+n-1}(S_t, A_t)], 0 \leq t < T$$

Important is that the off-policy one step TD learning starts and ends one step earlier than the off-policy SARSA algorithm.
The *importance sampling ration*(ρ) is is helpful to increase or decrease the weighting of for example the choose of an action. If the behaviour policy b would take an action not often but our target policy π choose this action often we want to increase the weighting of this action for the behaviour policy so the agent can exploit the action. If our target policy wouldn't take an action the return will be ignored because ρ is equal to zero for this case. As you can see in the definition of the *importance sampling ration* if we choose $\pi = b$ (so we have an on-policy case) this would lead to $\rho = 1$ so it wouldn't affect the update function, and we would get the same result as with the on-policy update.

Task I, q.4:

With the eligibility trace we can generalize and unify TD and MC approaches, our agent learns continually and uniformly in time and the learning isn't delayed by n-steps (our update can occur after each new state). With the so called *short term memory* (or eligibility trace) $\mathbf{z}_t \in \mathbb{R}^d$ we get the advantage that we use this single vector instead of saving the last n feature vectors. With the parameter λ (also called trace-decay parameter) we define the rate at which the trace falls. Methods that uses eligibility trace under repeated presentations can converge to the same predictions as *forward view* algorithms. If we choose $\lambda = 1$ we would get an MC approach, and if we would set $\lambda = 0$ we would get an one-step TD approach, so if we use an on-policy approach we would get a one-step SARSA algorithm, in the most cases it is more efficient to use a λ-value somewhere in between this both values an don't use one of this extremes.
With the eligibility trace we can use so called *backward views* algorithms. With the so called *dutch-eligibility trace* we can update every time step so we don't need to wait n-steps for an update like in for example n-step SARSA.
Definition of the *replacing trace*:

$$z_{i,t} = \begin{cases} 1 & \text{if } x_{i,t} = 1 \\ \gamma\lambda z_{i,t-1} & \text{else} \end{cases},$$

Definition of the *accumulating trace*

$$\mathbf{z}_{-1} = 0$$
$$\mathbf{z}_t = \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{v}(S_t, \omega_t), \quad \text{for } 0 \leq t \leq T$$

Task I, q.5:

So called *backward views* algorithm looking back to the lately visited states and updates with the current so called TD error for that we use the so called *eligibility trace*. It possible to get the (nearly) same update as for so called *forward view* algorithm.

Forward view algorithm are looking forward from the updated state, and not backwards. So with the *backward view* uses the TD error to learn while *forward view* algorithm used a target, so the update depends on things that happens later (in the future) forward the learning process.
Imagine we have the state trajectory $\tau = \{s_0, a_0, s_1, a_1, \cdots, s_{n-1}, a_{n-1}\}$ the $\mathcal{Q}$-value of the state-action pair (s_0, a_0) is estimated for the forward view algorithms (1-step case) like:

$$\mathcal{Q}(s_0, a_0) = \mathcal{Q}(s_0, a_0) + \alpha[\mathcal{R}_{t+1} - \mathcal{Q}(s_1, a_1)],$$
$$\text{for the next state-action pair:}$$
$$\mathcal{Q}(s_1, a_1) = \mathcal{Q}(s_1, a_1) + \alpha[\mathcal{R}_{t+2} - \mathcal{Q}(s_2, a_2)].$$

For the backward view algorithms it would look like:

$$\mathcal{Q}(s_0, a_0) = \mathcal{Q}(s_0, a_0) + \alpha * \delta * \mathbf{z},$$
$$\text{with the eligibility trace } \mathbf{z} \text{ and } \delta \text{ is defined as(for SARSA)}$$
$$\delta = \mathcal{R}_{t+1} + \gamma * \mathcal{Q}(s_1, a_1) - \mathcal{Q}(s_0, a_0),$$
$$\text{for the next state it will be:}$$
$$\mathcal{Q}(s_1, a_1) = \mathcal{Q}(s_1, a_1) + \alpha * \delta * \mathbf{z},$$
$$\text{with}$$
$$\delta = \mathcal{R}_{t+2} + \gamma * \mathcal{Q}(s_2, a_2) - \mathcal{Q}(s_1, a_1)$$

Task I, q.6:

Sample updates The agent is looking ahead to a successor state (or state-action pair), using the value of this successor state (or state-action pair) and the reward along the way to compute a backed-up value of the original state (or state-action pair).
Expected update differ from the sample update in that they are based on a complete distribution of all possible successor state rather than on a simple sample successor.
Bootstrapping in Reinforcement Learning can be read as *using one or more estimated values in the update step for the same kind of estimated value.* TD is a bootstrap method because we are in part using a Q value to update another Q value. The main disadvantage of bootstrapping is that it is biased towards whatever your starting values are.Those are are most likely wrong, and the update system can be unstable as a whole because of too much self-reference and not enough real data - this is a problem with off-policy learning.
Non-bootstrapping is the opposite of bootstrapping. So in non bootstrapping algorithms we aren't using estimated values in the update step. MC approaches are non-bootstrapping algorithm.
value iteration: expected update, bootstrapping, on-policy , model based
policy iteration: expected update, bootstrapping, on-policy , model based
MC: sample update, non-bootstrapping, on-policy, off-policy , model free
SARSA: Sample update, bootstrapping, on-policy , model free
one-step Q-learning: sample update, bootstrapping, off-policy , model free
n-step SARSA: sample update, bootstrapping, on-policy, model free
n-step off-policy learning: sample update, bootstrapping, off-policy, model free
SARSA(λ): sample update, bootstrapping, on-policy, model free

Task II, q.1:

As you can see in figure 1, the resulting episodic reward converge as good as our SARSA - implementation from the last assignment. In the next question we will talk about the convergence spead and the influence of our initial $\mathcal{Q}$-matrix have into that

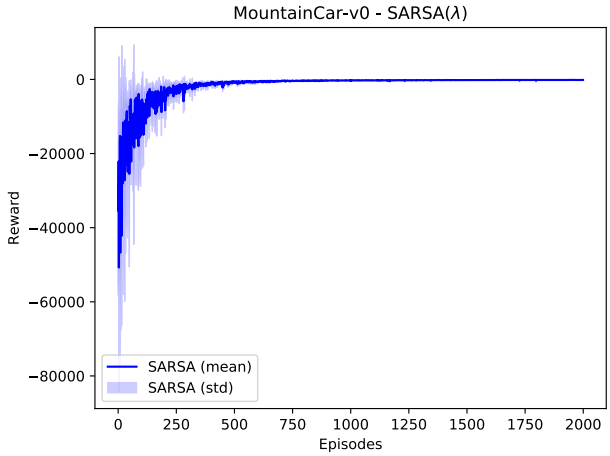

Figure 1 You can see the mean reward of SARSA(λ) with over 5 training evaluations, 2000 episodes each, the episode length wasn't limited, with a fixed $\alpha = 0.05$ and $\epsilon = 1 - \frac{E_c}{E_t}$

Task II, q.2:

Let us first Talk about the influence of the initial value of all entries of the $\mathcal{Q}$-matrix, if we choose $\mathcal{Q}(s,a) = 10000 \forall s, a$ we have optimistic initial values and this leads to a slower learning rate, also $\mathcal{Q}(s,a) = 0 \forall s, a$ are an optimistic initial value but not that optimistic as 10000, so the learning is faster for our tried λ - values. We made one run per chosen λ-value ($\lambda = 0$, $= 0.5$,$= 0.75$, $= 0.95$ and $\lambda = 0.99$) and stop the time the agent needs for 2000 Episodes where each episode just terminated after the agent reached the goal. For both initial $\mathcal{Q}$ - values the agent needs most of the time for $\lambda = 0$ and least time for $\lambda = 0.99$. So a higher λ-value takes less time to finish this 2000 episodes for learning.
All λ values learned a policy which will give a rewards around -150, what is also the value that the optimal policy will get (it was assumed that the optimal policy takes 150 steps from initial state to the goal).
If we look at the learned $\mathcal{Q}$ - matrix for 5000 training episodes with the initial value of $\mathcal{Q}(s,a) = 10000, \forall s, a$ we can see that for $\lambda = 0$ it isn't possible to reduce this initial bias that much, also for the other λ-values the agent got this problem but less then $\lambda = 0$, so if the λ-value increase, we got lower $\mathcal{Q}$ - matrix entries for the visited state-action pairs, but still they are very high, around 9500 - 10000, the values can be looked up at the 'SARSA_q2_Q10000_maxInf_5000eps.npy' file.
If we look at the possible λ values for the initial conditions $\mathcal{Q}(s,a) = 10000 \forall s, a$ and set the maximum episode length to 5000, it's not possible for the agent to reduce the visited state-action pair with any λ value, for example for $\lambda = 0$ we are around 9800 for 2000 episodes, also for the other λ values we got something similar.
If we look at the learned $\mathcal{Q}$ - values for the following initial $\mathcal{Q}$-matrix where are all entries are equal to 0 and 2000 episodes, we can see that for the visited states the values for $\lambda = 0.99$ are between -130 and -230, for $\lambda = 0.95$ it's between -160 and -270 for $\lambda = 0$ we got something between -190 to -300.

Bonus Tasks, q.1:

expected SARSA: sample update, bootstrapping, on-policy, off-policy (both possible), model free
n-step Tree Backup algorithm: sample update, bootstrapping, off-policy,model-free
Watkins Q(λ): sample update, bootstrapping, off-policy, model-free
n-step Q(σ): sample update, expected update, bootstrapping, off-policy, model free

Bonus Tasks, q.2:

Bonus Tasks, q.3:

Literatur