Course: Reinforcement Learning (SS2020)
Prof. Dr. Elmar Rueckert & Honghu Xue, M.Sc.
Assignment Report Nr. I
Date: 13. Mai 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:

In Reinforcement Learning we an agent which interacts with an environment by actions and received rewards based on these actions. The agent makes decisions and is the learners in RL.
A reward is a numerical values that the agent receives on performing some action at some state(s) in the environment , the reward can be positive or negative. We have a reward function which calculates the expected reward for this ($\mathcal{R}_s = \mathbb{E}(r_{t+1}|S_t = s)$).
The environment is the demonstration of the problem to be solved, it can be the real world or a simulated environment. The dynamics of the environment can be fully defined using states and a transition probability matrix $\mathcal{P}$, which we can also define as a matrix where the sum of each row is equal to 1.
A state s_t is the position of the agent at a specific time step t in the environment. So whenever the agent performs an action a he will get a reward on will reach a new state, the movement from one state to another state is called transition. The probability that the agent move from one state to another state is called transition probability and is defined as: $\mathcal{P}_{ss'} = P(S_{t+1} = s^{'}|S_t = s)$ so as you can see we assume the Markov property here.
In Reinforcement Learning we are about maximizing the cumulative reward instead of the reward the agent achieved from the last performed transition and the current state (which is also called the immediate reward), This total sum of reward is called return and is defined as: $\boldsymbol{G} = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1} \in \mathbb{R}^1, \gamma$ is called discount factor and is defined in the interval of $[0, 1]$ but more about that later.
The Value function determinds how good it is for the agent to be in a particular state what depends on some actions that it will take and thats the job of a policy. The policy defines which action($A_t = a$) is to perform in a particular state ($\pi(a|s) = P(A_t = a|S_t = s)$). So our value function for a specific policy π for a state s is defined as:

$$V_\pi(s) = \mathbb{E}_\pi(\boldsymbol{G}_t|S_t = s) = \mathbb{E}_\pi(\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1}|S_t = s), \forall s \in S,$$

where S is the set of all possible states.
So the Value function is the expected return starting from state s until we the terminal state (in case of an episodic task) with the policy π. Our goal is to find the best (optimal) policy for our agent in his environment with the following equation:

$$V(s) = \mathbb{E}(r_{t+1} + \gamma V(s_{t+1}|S_t = s), \tag{1}$$

this equation is also called the Bellmann Equation.
In the most cases our reward does not only depends on our states but also on the actions we take. The actions can influence our transition probability and so our transition probability matrix and also our reward function because of that they will also be attentioned as follows:

$$\mathcal{P}^a_{ss'} = P(S_{t+1} = s'|S_t = s, A_t = a)$$
$$\mathcal{R}^a_s = \mathbb{E}(r_t + 1|S_t = s, A_t = a)$$

So in short in Reinforcement Learning we have a set of States, a set of actions the agent can perform at each state to reach another state, a transition probability matrix (or function) a reward function, a discount factor and an agent. The four main sub elements for an agent are:

- policy (π)
- reward function or the return
- value function (the expected return)

- optionally : a model

Task I, q. 2:

For an episodic task the task comes to an end (e.g.tic-tac-toe), so if we assume $\gamma = 1$ and T is our final time step, our expected reward will be defined as:

$$\boldsymbol{G}_t = r_{t+1} + r_{t+2} \dots r_T.$$

If we have no finite number for T so we don't come to an end (e.g. tuning a heating system) we would have a continuing task.
With $\gamma \geq 1$ can be $\boldsymbol{G}_t$ easy go to infinity because of that we define the discount factor as follows $0 < \gamma < 1$ for continuing tasks which leads to:

$$\begin{aligned}\boldsymbol{G}_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{T} \gamma^k r_{t+k+1} \\ &= r_{t+1} + \gamma \cdot \boldsymbol{G}_{t+1}.\end{aligned}$$

So for an episodic task we try to maximise an finite sum and for the continuing task we try to maximise the infinite sum.

Task I, q. 3:

A stationary policy is time independent, this means an agent will take the same decision whenever certain conditions are met, so in general we would choose for state s every time the same action a or if we have deterministic policy the probability of choosing an action remains the same.

A non-stationary policy is updating by a learning algorithm, this means that we want to maximise the function $Q_t(a)$ which estimate the the value for the reward at time t if we choose action a.
We want to weight the recent rewards more then the long-past rewards, if the reward probabilities change, we need a step-size parameter which is denoted by α. If alpha is constant in the interval of $(0, 1]$ ($\alpha \in (0, 1]$), this leads to the following update rule for the Q_k function:

$$\begin{aligned}Q_{k+1} &= Q_k + \alpha[r_k - Q_k] \\ &= (1-\alpha)^k Q_1 + \sum_{j=1}^{k} \alpha(1-\alpha)^{k-j} r_j.\end{aligned} \tag{2}$$

To explain the influence of α we need to look in equation [2], let $\alpha(a)$ denote the step-size parameter used to process the reward received after the nth selection of action a.
So if we choose $\alpha(a) = \frac{1}{k}$,this would result in the sample-average method, so we will converge to the true action values but it isn't guaranteed for all choices of the sequence. We need to fulfill the following conditions to assure convergence:

$$\begin{aligned}&\sum_{k=1}^{\infty} \alpha_k(a) = \infty \\ &\sum_{k=1}^{\infty} \alpha_k^2(a) > \infty.\end{aligned} \tag{3}$$

So if $\alpha(a)$ isn't fulfill 3 we can't guarantee the convergence and if it fulfill the conditions it is often the case that is converge very slowly.

Task II, q. 1:

$V_\pi(s)$ returns the estimated value of the state s under the policy π, the optimal state-value function $V_{\pi^*(s)}$ returns the value of the state s under the optimal policy π^*.
In formulas we can write this as follows:

$$V_{\pi^*(s)} = \max_\pi V_{\pi(s)},$$

this need to hold for all possible states s. Also optimal policies have an optimal action-value function $Q_{\pi^*}(s,a)$ which is defined as the value of taken action a in state s under the optimal policy π^* The action-state value function $Q_\pi(s,a)$ returns the expected value of taking action a in state s under a policy π. To get an optimal action-value function we need to maximise the action-value function for the policy π:

$$\begin{aligned} Q^*_\pi(s,a) &= \max_\pi Q_\pi(s,a) \\ &= \mathbb{E}[r_{t+1} + \gamma V_{pi^*}(s+1)|S_t = s, A_t = a], \end{aligned}$$

So the function $Q_\pi(s,a)$ follows the policy π and $Q^*_\pi(s,a)$ follows the policy π^*.

Task II, q. 2:

We have two interacting, simultaneous processes in policy iteration, the policy evaluation and the policy improvement. In GPI the two processes interact, independent of the granularity and other details of the two processes. If both processes stabilize (no change anymore in the produce) the value function and the policy must be optimal. The improvement and evaluation processes in GPI can be viewed as both competing and cooperating. So for each cycle we get a better policy π^* ($V_{\pi^*(s)} > V_\pi(s)$) or if the GPI can't find a better policy it will stay with the same policy (so its stabilize), in general: we will always find a better policy until we reach the optimal policy.

Task III, q. 1:

The Goal is that our agent reached the goal tile in the grid world starting from his start tile. It's a 4 × 4 grid map so we have 16 possible states. Challenging is that if you go in a certain direction, there is only 0.333% chance that the agent will really go in that direction (e.g. $P(s_{t+1} = (2,1)|s_t = (1,1), a = \mathit{down}) = 1/3$). So the movement of the agent is uncertain and only partially depends on the chosen direction. So the episode ends when you reach the goal or fall in a hole, we have an episodic task here.

Task III, q. 2:

In Table 1 you can see the learned optimal policy π^* by our implemantation of the policy Iteration.

Tabelle 1 Value iteration optimal policy (π^*)

Pos	0	1	2	3
0	left	up	up	up
1	left	left	left	left
2	up	down	left	left
3	left	right	down	left

In Table 2 you can see the state value for each possible state rounded to 3 digits.

Tabelle 2 State Value Value iteration ($V_{\pi^*}(s)$)

Pos	0	1	2	3
0	0.823	0.822	0.822	0.823
1	0.823	0.0	0.529	0.0
2	0.823	0.823	0.764	0.0
3	0.0	0.882	0.941	0.0

Tabelle 3 Policy iteration optimal policy π^*

Pos	0	1	2	3
0	left	up	up	up
1	left	left	left	left
2	up	down	left	left
3	left	right	down	left

Task III, q. 3:

In Table 3 you can see the learned optimal policy π^* by our implemantation of the policy Iteration.
In Table 4 you can see the state value for each possible state rounded to 3 digits.

Tabelle 4 State Value Policy iteration $V_{\pi^*}(s)$

Pos	0	1	2	3
0	0.823	0.823	0.823	0.823
1	0.823	0.0	0.529	0.0
2	0.823	0.823	0.764	0.0
3	0.0	0.882	0.941	0.0

Task III, q. 4:

Both iteration methods work around the Bellman Equation where we find the optimal utility. For both we pick something randomly at first, for the Value iteration we pick a value function randomly and find new value function in an iterative process, until we reaching the optimal value function and after that we derive the optimal policy from it. For the policy iteration we select a policy randomly and find a corresponding value function to it, after that we find a new policy based on the previous value function and so until we reach the optimal policy. So the algorithms for policy evaluation and finding optimal value function are highly similar except for a max operation. We can see that both approaches achieve the same goal (table 1 and table 3), but policy iteration being more computationally efficient.
In our environment is the optimal policy also the optimal trajectory for our agent to reach the goal, but because of the slippery tiles it is possible that our agent can't reach the goal with it. In 75 out of 100 runs the agent reached the goal but in 25 he failed and fall in the hole.