Reinforcement Learning (2022/2023)
Assignment 2

Student Name: Gabriele G. Di Marzo
Student ID: 2012633

---

1. **Theory**

   (a) **Demonstrate** $\sum_{t=0}^{T-1} \alpha\delta_t e_t(s) = \alpha \sum_{t=0}^{T-1} (\gamma\lambda)^{t-k}\delta_t = \alpha(G_k^\lambda - V(s_t))$:
   SOLUTION:
   Considering:

$$\sum_{t=0}^{T-1} \alpha\delta_t e_t(s) = \text{total updates for TD}(\lambda) \tag{1}$$

$$G_k^\lambda = (1-\lambda)\sum_{n=1}^{h} \lambda^{n-1} G_k^{(n)} \tag{2}$$

$$\delta_t = [r_t + V(s_{t+1}) - V(s_t)] \tag{3}$$

   First we split the definition of G_k, so we get 2 different summation:
   $G_k^\lambda = (1-\lambda)\sum_{n=1}^{T-k-1} \lambda^{n-1} G_k^{(n)} + (1-\lambda)\sum_{n=T-k-1}^{h} \lambda^{n-1} G_k$

   The second part it's a geometric series that end up in $\lambda^{T-k-1} G_k$
   By expanding the summation of G_k^λ we get:

$$\begin{aligned}
&(1-\lambda)(r_k + \gamma V(s_{k+1})) + \\
&+ (1-\lambda)\lambda(r_k + \gamma r_{k+1} + \gamma^2 V(s_{k+2})) + \\
&+ ... + \\
&(1-\lambda)\lambda^{T-k-1}(r_k + \lambda r_{k+1} + ... + \gamma^{T-k-1} r_{T-1})
\end{aligned}$$

   Considering the original formula of G_k^λ and keep to expand we get:

$$\begin{aligned}
&(1-\lambda) r_k + (1-\lambda)\gamma V(s_{k+1}) + \\
&+ (\lambda - \lambda^2) r_k + (\lambda - \lambda^2)\gamma^2 V(s_{k+2}) + \\
&+ ... + \\
&+ (\lambda^{T-k-2} - \lambda^{T-k-1}) r_k + (\lambda^{T-k-2} - \lambda^{T-k-1})\gamma^{T-k-1} V(s_{T-1}) + \\
&\lambda^{T-k-1}(r_k + \gamma r_{k+1} + ... + \gamma^{T-k-1} r_{(T-1)})
\end{aligned}$$

   we can notice that the λr_k is simplified by the opposite value in the next element, so we remain with:

$$\begin{aligned}
&r_k + \gamma V(s_{k-1}) + \gamma V(s_{k+1}) - \lambda\gamma V(s_{k+1}) + \\
&+ \lambda\gamma^2 V(s_{k+2}) + (\lambda\gamma)^2 V(s_{k+2}) + \\
&+ \lambda^{T-k-2}\gamma^{T-k-1} V(s_{T-k-1}) + (\lambda\gamma)^{T-k-1} V(s_{T-1}) + (\lambda\gamma)^{T-k-1} r_{(T-1)}
\end{aligned}$$

   Adding the final term $V(s_k)$ and re-arrange we can find a pattern that look like:

$$\begin{aligned}
&r_k + \gamma V(s_{k-1}) - V(s_k) + \\
&+ (\gamma\lambda) r_{k+2} + \gamma^2 \lambda V(s_{k+1}) - \gamma\lambda V(s_{k+2}) + \\
&+ ... +
\end{aligned}$$

$+ (\gamma\lambda)^{T-1-k} r_{t+1} + \gamma^{T-k-1}\lambda^{T-k-2} V(s_{T-1}) - (\gamma\lambda)^{T-k-1} V(s_T)$
$= \sum_{t=0}^{T-1} (\gamma\lambda)^{t-k} [r_t + \gamma V(s_{t+1}) - V(s_t)] =$
$= \sum_{t=0}^{T-1} (\gamma\lambda)^{t-k} \delta_t$

multiply all by a α factor, we find our target relation, that result proved

(b) **Compute the update of Q-learning and Sarsa for given value**:

SOLUTION:
the update rule for the Sarsa algorithm are define as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{4}$$

where the next action a_{t+1} is chosen following the policy.
With the numerical value we get:

$Q(1,1) \leftarrow Q(1,1) + 0.1[5 + 0.5(Q(2,1)) - Q(1,1)]$
$Q(1,1) \leftarrow 1 + 0.1[5 + 0.5(3) - 1]$
$Q(1,1) \leftarrow 1 + 0.55 = 1.55$

The difference between Sarsa and Q-learning is in the selection of the next action a_{t+1}. In the Q-learning algorithm we select the new action with a greedy approach, so the update rule became:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma \max_{a_i} Q(s_{t+1}, a_i) - Q(s_t, a_t)] \tag{5}$$

Considering the state $s_{t+1} = 2$ we took the maximum value from the previous $Q - table$:

$Q(1,1) \leftarrow Q(1,1) + 0.1[5 + 0.5(\max_{a_i} Q(2, a_i)) - Q(1,1)]$
$Q(1,1) \leftarrow Q(1,1) + 0.1[5 + 0.5(Q(2,2) - Q(1,1)]$
$Q(1,1) \leftarrow 1 + 0.1[5 + 0.5(4) - 1]$
$Q(1,1) \leftarrow 1 + 0.6 = 1.6$

2. **Practical**

(a) **Sarsa(λ)**
For my Sarsa(λ) implementation i have choose the backward-view of the algorithm. Using a eligibility trace, it's possible update the $Q-table$ value without wait for the terminal condition. Each pair (s, a) get a "credit", stored in the eligibility trace. This credit will be update following the formula :

$$e_t(s) = \gamma\lambda e_{t-1}(s) + I(s_t = s) \tag{6}$$

so, the two part of the update rule correspond to a frequency term ($I(s_t = s)$) and a temporal term, which will be scaled down in the future run.

In my code i chose to add a new condition for the ϵ decay:
After the first reward the ϵ will decrease for each step. I found that when the agent finds the treasure (get the reward) at the firsts run, the performance in the final evaluation will be lower that usual. In this way the agent stop the exploration to soon. Therefore, i add a fixed number of episodes to be performed before starting the epsilon decay process.

(b) **RBF and LFVA**:
For my implementation of `RadialBasisFunction` encoder i chose a random 100 centers, uniformly sampled in the range of the state. The sigma it's fixed across all the x_i, even though there is the possibility of setting a non-fixed sigma by the implementation of `set_non_constant_sigma`. I also added a normalization part inside the `encode` function. That's because otherwise i was getting a Overflow error during the calculation of the Q-function
The reference for the normalization part:
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

The `TD_LFVA` agent does not work. My solution cannot reach the final goal. I've try to implement both version: backward-view using as reference some implementation found online, and the forward view as did during the practical session. I'm pretty sure that the update rule is fine, but i was not able to find the error in my code. The encoder works, it has been tested also in the practical environment (instead of the vanillia encoder) and i didn't see decrease of performance. Another test was on the hyperparameter setting: i chosen a bigger learning rate (α) and a value of 0.99 for the α-decay.

Contributor: Syrine Enneifer