Reinforcement Learning

Assignment #02

SAPIENZA
UNIVERSITÀ DI ROMA

Info

- **Deadline:** Friday November 24th

- Students may discuss assignments, but the solutions must be typed and coded up **individually**

- Students must **indicate the names of colleagues they collaborated with**

Folder organization

- The assignment source code will be available on Classroom. You will find:
  - assignment2.pdf: with all the information
  - assignment2.zip that contains:
    - ilqr/
      - requirements.txt
      - main.py (do not touch!)
      - student.py
    - policy_iteration/
      - requirements.txt
      - main.py (do not touch!)
      - student.py

Theory submission

The theory solutions must be submitted in a pdf file named "XXXXXXX.pdf", where XXXXXXX is your matricula.

We encourage you to type equations on an editor, rather than uploading scanned files.

Use the pdf file also to communicate the **students you collaborated with** and to insert a **small report of the code exercises**.

Code submission

The code solutions must be submitted in a zip file named “XXXXXXX.zip”, where XXXXXXX is your matricula.

The zip file must be organized exactly as the original assignment.zip file. Wrongly submitted assignments will be penalized.

Only edit the “students.py” files.

Theory

1. Given the following Q table:

$$Q(s,a) = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} Q(1,1) & Q(1,2) \\ Q(2,1) & Q(2,2) \end{pmatrix}$$

   Assume that $\alpha = 0.1$, $\gamma = 0.5$, after an experience $(s, a, r, s') = (1, 2, 3, 2)$ compute the update of both Q-learning and SARSA. For the latter consider $a' = \pi_\epsilon(s') = 2$

2. Prove that the n-step error can also be written as a sum of TD errors if the value estimates don't change from step to step, i.e.:

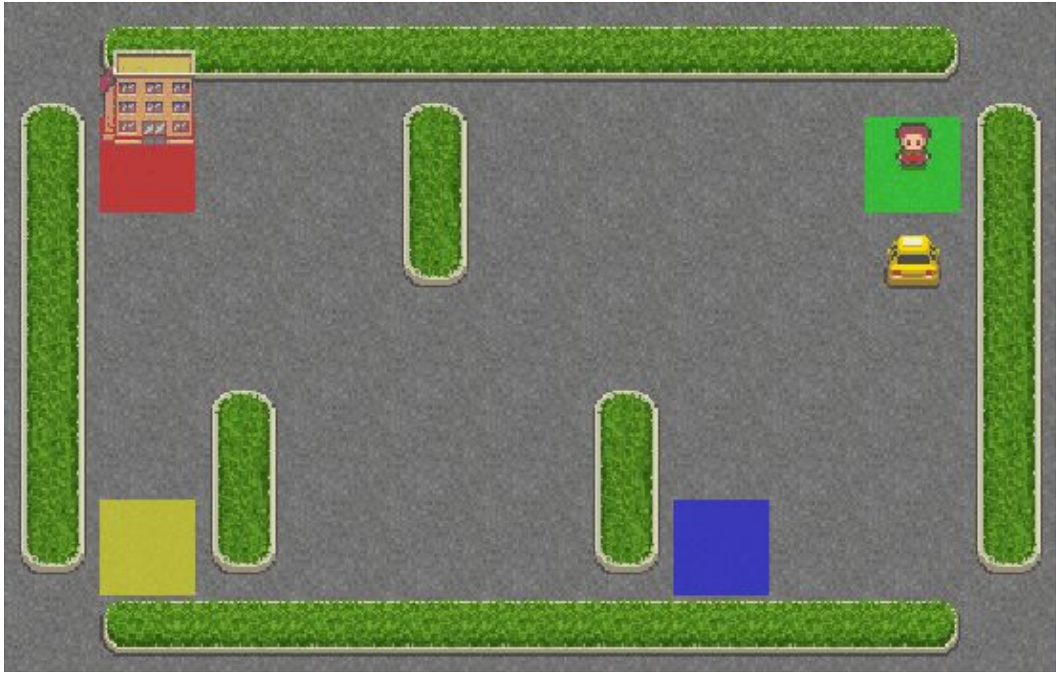
$$G_{t:t+n} - V_{t+n-1}(S_t) = \sum_{k=t}^{t+n-1} \gamma^{k-t} \delta_k$$

   where $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$

Code: SARSA-lambda (tabular)

“Taxi” environment from gymnasium

The goal is to pick up the passenger from one of the colored squares and drop it off at the hotel.



Code: SARSA-lambda (tabular)

1. Implement the epsilon-greedy policy

```python
def epsilon_greedy_action(env, Q, state, epsilon):
    # TODO choose the action with epsilon-greedy strategy
    action = ...
    return action
```

Code: SARSA-lambda (tabular)

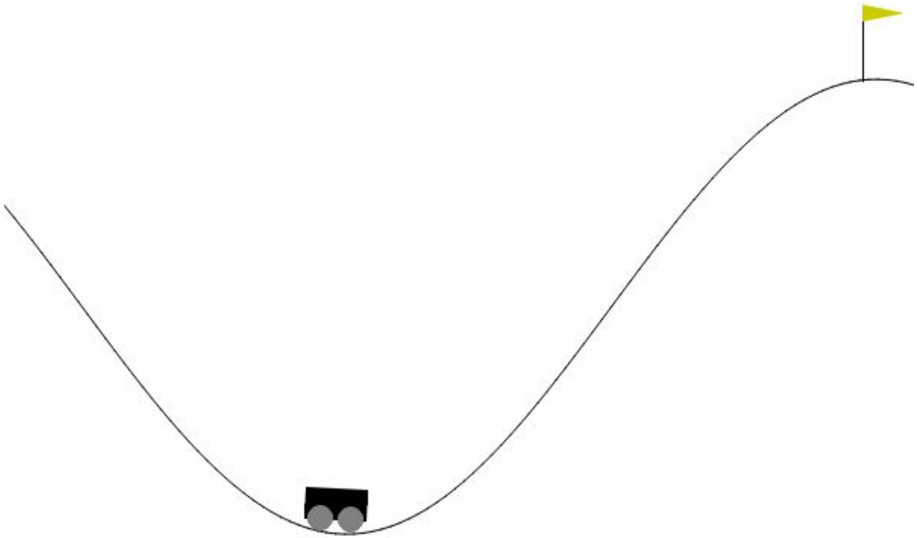
2. Implement SARSA-lambda:
    a. update Q table (Q)
    b. update eligibility traces (E)

```
############# define Q table and initialize to zero
Q = np.zeros((env.observation_space.n, env.action_space.n))
E = np.zeros((env.observation_space.n, env.action_space.n))
print("TRAINING STARTED")
print("...")
# init epsilon
epsilon = initial_epsilon

received_first_reward = False

for ep in tqdm(range(n_episodes)):
    ep_len = 0
    state, _ = env.reset()
    action = epsilon_greedy_action(env, Q, state, epsilon)
    done = False
    while not done:
        ############## simulate the action
        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated
        ep_len += 1
        # env.render()
        next_action = epsilon_greedy_action(env, Q, next_state, epsilon)

        # TODO update q table and eligibility
        ...

        if not received_first_reward and reward > 0:
            received_first_reward = True
            print("Received first reward at episode ", ep)
```

Code: Q-Learning TD(lambda) + RBF

“MountainCar” environment from gymnasium

The goal is to reach the flag on top of the mountain. Note: the car does not have enough power to climb the mountain with no initial velocity!



Code: Q-Learning TD(lambda) + RBF

1. Implement the RBF feature encoder
   a. initialization
   b. encoding
   c. number of features

```python
class RBFFeatureEncoder:
    def __init__(self, env): # modify
        self.env = env
        # TODO init rbf encoder
        ...

    def encode(self, state): # modify
        # TODO use the rbf encoder to return the features
        return ...

    @property
    def size(self): # modify
        # TODO return the number of features
        return ...
```

Code: Q-Learning TD(lambda) + RBF

2. Implement the Q-learning TD(lambda) update rule for LVFAs
   - you can implement either backwards or forwards view
   - for backwards, you can use the eligibility traces already initialized by us (but you still have to update them in update_transition!)

```python
def update_transition(self, s, action, s_prime, reward, done): # modify
    s_feats = self.feature_encoder.encode(s)
    s_prime_feats = self.feature_encoder.encode(s_prime)
    # TODO update the weights
    self.weights[action] += ...
```