Reinforcement Learning

Assignment #03

SAPIENZA
UNIVERSITÀ DI ROMA

Info

- Deadline: December 4th, 2022
- 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:
    - assignment3.pdf: with all the information
    - assignment3.zip that contains:
        - car_racing/
            - main.py (do not touch!)
            - requirements.txt (you may 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

Suppose you have an environment with 2 possible actions and a 2-d state representation ($x(s) \in \mathbb{R}^2$). Consider the REINFORCE Algorithm with the following Linear Function Approximator (LFA) policy (Logistic Regression) such that:

$$\pi(a = 1|s) = \sigma(w^T x(s)) \tag{1}$$

$$a = \mathbb{1}_{\pi(a=1|s)>0.5} \tag{2}$$

where $w = [0.8, 1]$ are the weights and $y = \sigma(t) = \frac{1}{1+e^{-t}}$ is the sigmoid function.

Suppose you are doing an iteration of the REINFORCE algorithm and you have just run an episode getting the following trajectory:

$$x(s_0) = [1, 0]^T, \quad a_0 = 0, \quad r_1 = 0 \tag{3}$$

$$x(s_1) = [1, 0]^T, \quad a_1 = 1, \quad r_2 = 1 \tag{4}$$

$$x(s_2) = [0, 1]^T \tag{5}$$

Show the weights w update according to the REINFORCE algorithm ($\alpha = 0.1$, $\gamma = 0.9$).

Code: CarRacing-v2

Solve the CarRacing-v2 gym environment using one of the following algorithms:

- Double DQN with proportional prioritization
- World Models
- Advantage Actor-Critic (A2C)
- TRPO
- PPO

Code: CarRacing-v2

The **grade will be assigned basing on the correctness** of the code.

3 additional points will be awarded to the 3 best students according to the **following criteria:**

- agent performance
- algorithm/implementation complexity

Code: CarRacing-v2

In the folder “car_racing” you will find:

- main.py (do not touch!)

- requirements.txt (you may touch)

- student.py (please, touch)

Code: CarRacing-v2

- numpy
- scipy
- gym
- gym[box2d]
- sklearn
- torch

You may add some requirements, but they need to be authorized on Classroom.

In order to request them, **place a comment** under the assignment post.

Stable-baselines and other **libraries that already implement the algorithm are banned**. You need to use **py-torch as deep learning framework**.

Code: CarRacing-v2

```python
import argparse
import random
import numpy as np
from student import Policy
import gym

def evaluate(env=None, n_episodes=1, render=False):
    agent = Policy.load()

    env = gym.make('CarRacing-v2', continuous=agent.continuous)
    if render:
        env = gym.make('CarRacing-v2', continuous=agent.continuous, render_mode='human')

    rewards = []
    for episode in range(n_episodes):
        total_reward = 0
        done = False
        s, _ = env.reset()
        for i in range(max_steps_per_episode):
            action = agent.act(s)

            s, reward, done, truncated, info = env.step(action)
            if render: env.render()
            total_reward += reward
            if done or truncated: break

        rewards.append(total_reward)

    print('Mean Reward:', np.mean(rewards))
```

```python
def train():
    agent = Policy()
    agent.train()
    agent.save()


def main():
    parser = argparse.ArgumentParser(description='Run training and evaluation')
    parser.add_argument('--render', action='store_true')
    parser.add_argument('-t', '--train', action='store_true')
    parser.add_argument('-e', '--evaluate', action='store_true')
    args = parser.parse_args()

    if args.train:
        train()

    if args.evaluate:
        evaluate(render=args.render)


if __name__ == '__main__':
    main()
```

Code: CarRacing-v2

```python
import gym
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np

class Policy(nn.Module):
    continuous = False # you can change this

    def __init__(self, device=torch.device('cpu')):
        super(Policy, self).__init__()
        self.device = device

    def forward(self, x):
        # TODO
        return x

    def act(self, state):
        # TODO
        return

    def train(self):
        # TODO

    def save(self):
        torch.save(self.state_dict(), path)

    def load(self):
        self.load_state_dict(torch.load(path), map_location=self.device)

    def to(self, device):
        ret = super().to(device)
        ret.device = device
        return ret
```