Robot Learning and Vision for Navigation
Exercise 3 – Reinforcement Learning

Release date: Monday, 20 March. 2023 - **Deadline for Homework: Monday, 3 April. 2023 23:59**

You will have **Two Options** in this homework: one using OpenAI-Gym environment, and the other using CARLA environment. **Completing either one of them will receive full credit.**
You need to submit a **.zip** folder containing your report as a **.pdf** file (up to 5 pages), your best-trained model **agent.pth**, and your **.py** code files.

As in the previous exercise, please use the provided code templates. Do not change the `evaluate` function in `deepq.py`. Comment your code clearly and structure it well, messy code will not be graded.

Note of caution: Training a reinforcement learning agent takes time. As this exercise requires you to train several agents, it may be helpful to start working on this exercise sheet reasonably early.

OPTION 1: OpenAI-Gym Environment

3.1 Local OpenAI-Gym Installation (0 Points)

We ask you to install OpenAI-Gym on your machine:

a) Please install Python 3.7+ if you don't have it.

b) Unzip the `sdc_gym.zip` you downloaded together with this Exercise Sheet and enter the folder `sdc_gym`, install the Box2D enviroment by the command

```
pip install -e '.[box2d]'
```

c) After installation, please take a screenshot of your **full** screen with the car racing game on top. Please pack the screenshot into the ZIP file and upload it to the Blackboard system. If you are in the `sdc_gym` folder, you should be able to start the car racing game with the command

```
cd gym/envs/box2d
python car_racing.py
```

3.2 Base Implementation (2+2+2+2+2 Points)

The aim of this exercise is to implement a Deep Q-Learning agent for the CarRacing environment.

a) Deep Q-Network: Implement a Deep Q-Network and its forward pass in the `DQN` class in `model.py`. Your network should take a single frame as input. Describe your architecture in your report.

  i) Would it be a problem if we only trained our network on a crop of the game pixels, which would not include the bottom status bar and would not use the extracted sensor values as an additional input to the network? *Hint: Think about the Markov assumption.*

  ii) Why do we not need to use a stack of frames as input to our network as proposed to play Atari games in the original Deep Q-Learning papers [1][2]?

b) Deep Q-Learning: Implement the functions `perform_qlearning_step` and `update_target_net` in `learning.py`. The first should run a single Deep Q-Learning update step while the second should update the target network with the weights of the policy network. *Hint: Make sure to clip your gradients* [1].

  i) Why do we utilize fixed targets with a separate policy and target network?

  ii) Why do we sample training data from a replay memory instead of using a batch of past consecutive frames?

c) **Action selection:** In `action.py`, implement the functions `select_exploratory_action` and `select_greedy_action` which select an action according to an exploratory ϵ-greedy strategy or a greedy strategy (i.e. the action which maximizes the Q-values of your policy network).

   i) Why do we need to balance exploration and exploitation in a reinforcement learning agent and how does the ϵ-greedy algorithm accomplish this?

d) **Training:** Train a Deep Q-Learning agent using the `train_racing.py` file with the provided default parameters. Describe your observations of the training process. In particular, show the generated loss and reward curves and describe how they develop over the course of training. Some points of interest to describe should be: How quickly is the agent able to consistently achieve positive rewards? What is the relationship between the ϵ-greedy exploration schedule and the development of the cumulative reward which the agent achieves over time? How does the loss curve compare to the loss curve that you would expect to see on a standard supervised learning problem?

e) **Evaluation:** Evaluate the trained Deep Q-Learning agent by running the `evaluate_racing.py` script. Observe the performance of the agent by running the script on your local machine. Where does the agent do well and where does it struggle?

Important: Make sure you have a working baseline implementation, in which the trained agent is often able to achieve positive scores as well as take some corners before moving on to work on Section 3.3.

3.3 Further Investigations and Extensions (2+2+2+2+2 Points)

Please note that all further investigations and extensions should be made w.r.t. the baseline agent which you have implemented in the previous section. Except for your best solution in part **e)**, you should not combine subsequent extensions but investigate the effect of each aspect separately on its own.

a) **Discount factor γ:** Investigate the influence of the discount factor γ. Show reward curves that demonstrate the effect of an increase/decrease of γ from its default of 0.99 on your agent.

   i) Why do we typically use a discount factor in reinforcement learning problems and in which cases would it be a problem not to use a discount factor (i.e. $\gamma = 1$)?

b) **Action repeat parameter:** Describe the reasoning behind the use of an `action_repeat` parameter. By default, this value is set to 4. What is the effect on the training progress (look at your training plots) and your evaluation performance if this parameter is increased or decreased? Discuss and interpret your findings.

   i) Why might it be helpful to repeat each action several times?

c) **Action space:** By default, the agent uses a 4-dimensional action set of left-turn, right-turn, brake and acceleration (see `get_action_set` function in `action.py`). Investigate the addition of a null action ([0,0,0]) as well as more fine-grained turning, braking or acceleration actions. What is the effect on the agent's driving style as well as its evaluation score? Which additional actions lead to an improvement in the agent's performance and which do not?

   i) Why might it not always be helpful to increase an agent's action space?

   ii) In general, why are Deep Q-Networks limited to a discrete set of actions, and what solutions exist to overcome this limitation?

d) **Double Q-learning:** One problem with the standard Deep Q-Learning approach is an overestimation of Q-values. A proposed solution to this problem is the double Q-learning algorithm [3]. Read this paper, implement the double Q-learning algorithm and evaluate its effect on the training and evaluation performance of your agent. *Important: Make sure to include your double Q-learning implementation in your submitted code.*

   i) Shortly summarize the reason for the overestimation of Q-values in a standard DQN.

   ii) How does double Q-learning algorithm solve this issue?

e) Best solution: Finally, putting together your previous findings as well as any more ideas you might have, fine-tune and evaluate your best-performing agent. How is this agent constructed and trained? In which aspects has its performance improved over your baseline agent from Section 3.2 and where does it still exhibit sub-optimal behavior? Briefly sketch out an idea for overcoming the main remaining issue with your agent's training or performance *(this does not need to be implemented)*. Save the weights of your best performing agent as `agent.pt` and include it in your submission.

OPTION 2: CARLA Environment

In CARLA, due to the difficulty in training, we are only asking to implement any reinforcement learning agent, with the following:

a) A designed reward function, e.g., based on CIRL [4] leveraging speed, collisions, staying in lane, jerkiness, etc.

b) A report containing your findings with example results (e.g., reward training curves)

c) Evaluate your model using our provided CARLA evaluation (with your agent network). Report the impact of training hyper-parameters on performance, including discount factor, reward definition, and exploration rate. You may adapt the provided OpenAI Gym code structure for your CARLA code.

Note that an RL agent on CARLA doesn't have to drive very well, and by providing your training code, how the reward function is defined, and a small demo, you can still get full credit.

3.4 Competition (0 Points)

Congratulations! You have just built a reinforcement learning agent for navigation.

For a final ranking, we will run the `evaluate` function on a secret set of tracks for every submission, and evaluate separately based on the two options.
For OpenAI-Gym track, the cumulative reward after 600 frames is used as the performance measure. Evaluation is stopped early if the agent is `done` (e.g. has left the track). The overall score is computed by taking the mean cumulative reward from all validation tracks. The reward function is defined as -0.1 for every frame and $+1000/N$ for every track tile visited, where N is the total number of tiles in the track.
For CARLA track, we keep the same evaluation as HW1 so you can use the same evaluation code provided on HW1. We will evaluate two settings: low and high density of dynamic objects. The routes will be kept hidden. For each route, we will compute the success rate (higher the better), route completion (higher the better), and the number of collisions (lower the better). The final score is the average rank of the three metrics.
The winner is the student with the highest score. Good luck!.

3.5 References

[1] https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
[2] https://arxiv.org/pdf/1312.5602.pdf
[3] https://arxiv.org/pdf/1509.06461.pdf
[4] https://www.sciencedirect.com/science/article/pii/S0004370222001692