Reinforcement Learning
Project work

November 17, 2022

1 Learning goals

In the project work, students move to a more independent working style compared with the exercises. After the project work students have learned:

- How to tune hyperparameters of reinforcement learning algorithms and present a performance comparison of algorithms.
- How to choose and review research paper ideas and how to implement the ideas in practice.

2 Introduction

During the project work, we will leverage the algorithms you learned in the course in order to solve various environments: from easy to more complex ones. You'll conduct a set of experiments using the algorithms you have implemented for the assignments during the course and investigation the improvements over these algorithms in the literature.
The project contains two parts. In the first part, you're given a set of RL environments and you will train and finetune the algorithms you've learned during the course. For the second part, you will read a research paper that aims to improve the performance of the algorithms from the first part using the ideas from the list of research RL papers we offer you.

You get points for training RL agents to achieve better episode reward in several RL environments. More complex environments weigh more points. Importantly, you need to pay extra attention to how the results are presented:

Aalto University
School of Electrical Engineering

Reinforcement Learning course staff
Aalto Robot Learning Lab
aalto.fi, rl.aalto.fi

- All the experiments should be run in 3 random seeds and the episode reward, i.e. sum of rewards the agent received within a single episode, should be averaged over these seeds.
- Episode reward plots should depict the standard deviation of these runs as well.
- Plots should show a comparison between algorithms.

An example plot can be seen in Fig.1.

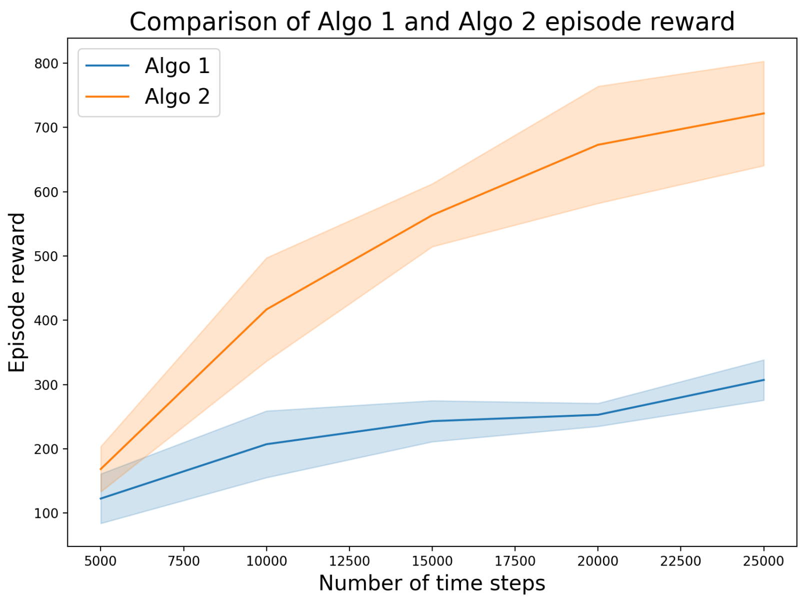

Figure 1: Example plot showing the comparison of Algo 1 and Algo 2 episode rewards, where the shaded area depicts the standard deviation when averaging over 3 random seeds.

For several decades of Reinforcement Learning (RL) research, there were many different environments implementations that were used to compare RL algorithms. To make the comparison fair for every approach, several benchmarks were developed. For example, OpenAI Gym [1] and DeepMind Control Suite [2], where the former we have selected as the main platform for conducting experiments.
This project work is supposed to be done in groups of 2 students. If you need to find a partner for the project, please join the **project** channel on Slack and advertise yourself. Furthermore, it's possible to choose an alternative project by proposing their own topic to combine RL with their research. The proposal of the alternative project should be submitted before 8th Nov (but please contact TAs asap to discuss your project) and **the deadline for both the standard project and the alternative project is 12.12.2022, 23:55**. We mainly recommend this for doctoral students who want to explore possible ways of integrating reinforcement learning in their research.

3 Part I

In this part, you are given a list of RL environments of different difficulty levels. Use the algorithms from the course code base, that is, those algorithms you've implemented during the exercises. Please, see below the list of the selected environments. Each environment has a target episode reward. Reaching target episode reward gives maximum points for the environment. Some of the environments have several levels of complexity. You need to choose **only 2 environments** from the list below. You need to choose one level per environment if there is more than one.

- **MountainCarContinuous** (Fig.2a): continuous control car should learn to climb the mountain within 999 timesteps.
  - Target episode reward: 85, reaching target episode reward gives 12 points.
  - Environment configuration file: *mountaincarcontinuous_easy.yaml*.
  - List of available algorithms: DQN with discretization of action space (Exercise 4), PG (Exercise 5), DDPG and A2C (Exercise 6), AlphaZero (with discretization of action space) and CEM (Exercise 7).
- **LunarLander** (Fig.2b): familiar with Exercise 3 lander should learn to land between flags.
  - Easy level: *lunarlander_discrete_easy.yaml*, target episode reward of 200. Reaching target episode reward gives 12 points.
  - Medium level: *lunarlander_discrete_medium.yaml*, target episode reward of 100. Reaching target episode reward gives 12 points plus 5 extra points.
  - List of available algorithms: DQN (Exercise 4), PG (Exercise 5), DDPG (use *lunarlander_continuous_easy.yaml* and *lunarlander_continuous_medium.yaml* config files for this algorithm) and A2C (Exercise 6), AlphaZero and CEM (Exercise 7).
- **BipedalWalker** (Fig.2c): two legs robot should learn to reach the end of the terrain.
  - Easy level: *bipedalwalker_easy.yaml*. Reaching target episode reward of 250 gives 15 points.
  - Medium level: *bipedalwalker_medium.yaml*. Reaching target episode reward of 200 gives 15 points plus 10 extra points.
  - List of available algorithms: PG (Exercise 5), DDPG and A2C (Exercise 6), CEM (Exercise 7).
- **Hopper** (Fig.2d): one-legged robot should learn to run forward without falls.
  - Easy level: *hopper_easy.yaml*. Reaching target episode reward of 2000 gives 15 points.

  - Medium level: *hopper_medium.yaml*. Target episode reward of 2000. Reaching target episode reward gives 15 points and 10 extra points.
  - List of available algorithms: PG (Exercise 5), DDPG and A2C (Exercise 6), CEM (Exercise 7).

We provide a function called *create_env* to initialize the correct version of the environment. You can find this function in *make_env.py*. In addition, we provide configuration files for each environment. In order to initialize the correct environment, you need to replace the code from exercises in the environment initialization with the following line:

```
    env = create_env(config_file_name=<name_of_config_file>, seed=<seed>)
```

Here <*name_of_config_file* should be replaced with the name of the configuration file given above for each environment and seed is up to you.

Please carefully check if an environment has discrete or continuous action space and change the code from the exercises accordingly.

Please note that the target episode reward should be reached by averaged episode reward using 3 random seeds. Within Part 1 if an algorithm failed to reach the stated target episode reward the number of points is exponentially using the following formula:

$$\text{points\_for\_env} = \text{points\_for\_solved\_env} \cdot \exp\left(-2\frac{\text{target\_ep\_rew} - \text{reached\_ep\_rew}}{\text{target\_ep\_rew}}\right) \tag{1}$$

Compare algorithms using sample-efficiency, that is, the number of episodes the algorithm needs to reach a given episode return, and clock-time (actual real-world time measured from the start of training) during training. The idea of the latter comparison is to show the real clock speed of the training of several algorithms. Results are assumed to be obtained using similar hardware, therefore making this comparison fair. The goal is to show which algorithm learns quicker in terms of real clock time compared to others.

3.1 Task 1

Using the code base from the course select **2** algorithms from the list of available algorithms for the selected environments. You may use the implementation from your submissions or copy the code from the released solutions. Train selected algorithms on the list of environments above. If more than one level of complexity is given, select one among *easy* and *medium*. Provide trained model parameters in the format of *.pt*.
Note: points per environment are rewarded to the best algorithm you chose using (1), i.e. if you selected two algorithms in this task, the best algorithm performance should reach the target episode reward. (Up to 30')

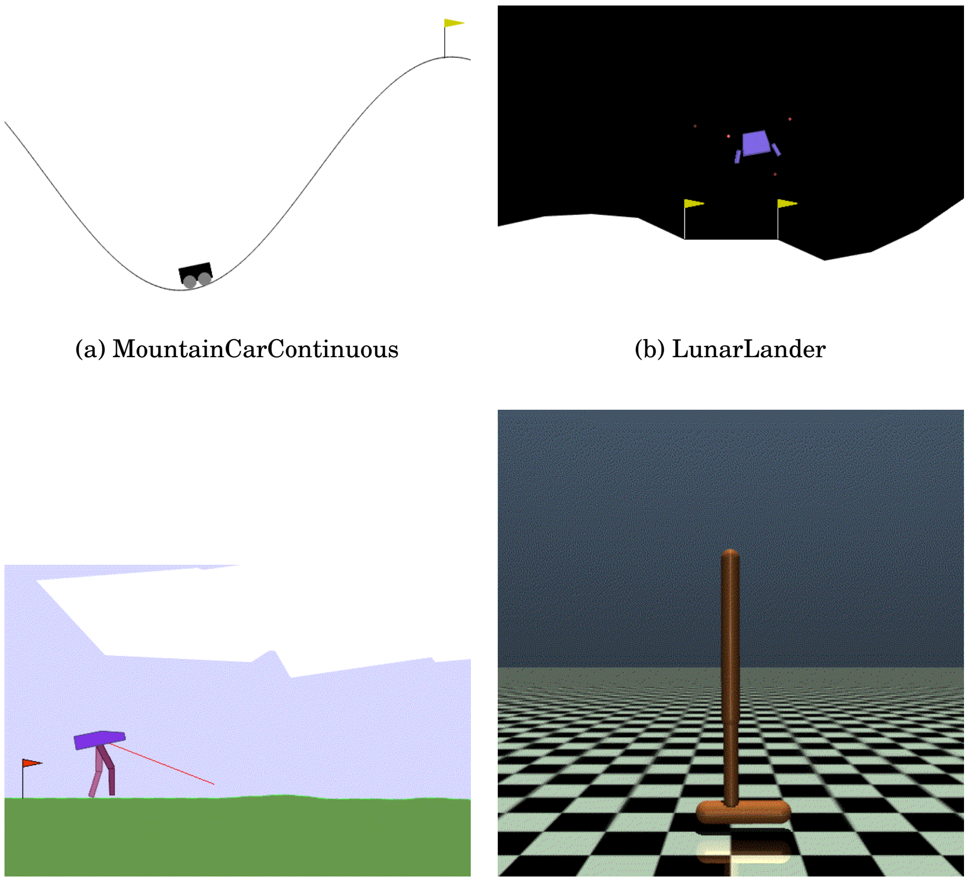
(a) MountainCarContinuous (b) LunarLander (c) BipedalWalker (d) Hopper

Figure 2: OpenAI Gym environments

3.2 Question 1

Which algorithm performed better in terms of averaged episode reward? What might be the reason in your opinion? Compare algorithms using sample-efficiency and clock-time. (5')

3.3 Question 2

Which hyperparameters did you change to make the algorithm work? Why? Justify your answer with graphical and algorithmic arguments. (5')

3.4 Grades for Part I

In order to obtain the reached episode reward by the algorithm you need to perform the following steps:

1. By the end of the training (after iterations exceed or after early-stopping) evaluate the agent for 50 episodes and average the episode reward over 50 episodes.
2. Since you're training 3 random seed agents, average each agent's mean episode reward from the previous step over 3 random seeds. Provide also standard deviation over 3 random seeds.
3. Obtained mean episode reward from the previous step will be used to compute the number of points using equation 1 from the document. So it should approach the target episode reward.

In Task 1 the best algorithm scores obtained using the procedure above for each environment are summed up. Questions provide 10 more points. In total, Part 1 contains 40 points with 20 extra points at max.

4 Part II

In this part, you will investigate ideas and approaches that seek to improve the performance of baseline algorithms from Part I. To do so, we suggest you a list of RL papers that provide remedies for the drawbacks of baseline algorithms.

4.1 List of research papers

The list of papers is presented below:

1. Prioritized Experience Replay [3].
2. Deep Reinforcement Learning with Double Q-learning [4].
3. Dueling Network Architectures for Deep Reinforcement Learning [5].
4. Exploration by Random Network Distillation [6].
5. Deep Exploration via Bootstrapped DQN [7].
6. Continuous Deep Q-Learning with Model-based Acceleration [8], learning model can be skipped in this paper.
7. High-Dimensional Continuous Control Using Generalized Advantage Estimation [9].
8. Proximal Policy Optimization Algorithms [10].
9. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor [11].
10. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model [12].

If you're interested in implementing the paper that wasn't listed above, please contact TAs team for confirmation before implementing the paper.

4.2 Question 1

Select a single research paper from the list of research papers above. Read the paper and explain the main ideas of the paper and the intuition behind it. (15')

4.3 Task 1

Implement the selected approach into the code of baseline algorithms. (10')

4.4 Question 2

Did the implemented ideas improve the baseline performance in terms of episode reward, sample-efficiency, or clock-time? Please justify your answer using graphical arguments and numerical information from your experiments, e.g. episode reward plot. (5')

4.5 Task 2

Train and fine-tune hyperparameters on the **same** environments as in Part I. Compare the performance using 3 random seed averaging. Remember to provide saved model weights using *.pt* format. (15')

4.6 Question 3

Can upgraded algorithms cope with increased environment complexity? Why / why not do you think the proposed approaches should improve the baseline algorithms? (5')

4.7 Extra task

If you're doing good and have the energy and passion for the more serious environment using algorithms from Part II, then try your algorithm to solve the environment from *hopper_hard.yaml*. For this task, you'll be awarded with 10 extra points.

4.8 Grades for Part II

In Task 1 the correctness of the implementation of the selected paper will be evaluated. In Task 2 there will be full points if the performance of the improved algorithm outperforms (or matches) the best baseline. The performance should be measured using the procedure from 3.4. In total, Part II contains 50 points.

5 Report and Grading

In this section, there are instructions on the submission will be graded. The report should include the following sections:

- Part I: models weights and comparison plots, 40'.
- Part II experimental results and question answers, 50'
- Conclusion, 2'

We will give **8 points** for the report quality:

- Structure, 2'
- Formatting, 2'
- Plots included and labeled, 2'
- Language, 2'

6 Submission

The **deadline** to submit the solutions through MyCourses is on **Monday, 12.12.2022** at 23:55.

Your submission should include:

```
<first_name>_<last_name>_<student_number>
 ├─ Report_<first_name>_<last_name>_<student_number>.pdf
 ├─ part1
 │   ├─ code, train.py, evaluate.py, etc
 │   ├─ <env_1_name>
 │   │   ├─ <algo_1_name>_<seed>_weights.pt
 │   │   └─ <algo_2_name>_<seed>_weights.pt
 │   └─ <env_2_name>
 │       ├─ <algo_1_name>_<seed>_weights.pt
 │       └─ <algo_2_name>_<seed>_weights.pt
 └─ part2
     ├─ code, train.py, evaluate.py, etc
     └─ Trained agents weights with folder pattern following part1
```

Please provide just a single *.pt* file for each agent per environment, i.e. **one run weights file** out of 3 random seeds agent initializations.

Only one student from the group should make a submission.

Please remember that not submitting a PDF report following the **Latex template** provided by us will lead to the subtraction of points.

For more formatting guidelines and general tips please refer to the submission instructions file on MyCourses.

If you need help or clarification solving the project, you are welcome to join the exercise sessions.

References

[1] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[2] Saran Tunyasuvunakool et al. “dm_control: Software and tasks for continuous control”. In: *Software Impacts* 6 (2020), p. 100022. ISSN: 2665-9638. DOI: `https://doi.org/10.1016/j.simpa.2020.100022`. URL: `https://www.sciencedirect.com/science/article/pii/S2665963820300099`.

[3] Tom Schaul et al. *Prioritized Experience Replay*. 2015. DOI: `10.48550/ARXIV.1511.05952`. URL: `https://arxiv.org/abs/1511.05952`.

[4] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. DOI: `10.48550/ARXIV.1509.06461`. URL: `https://arxiv.org/abs/1509.06461`.

[5] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning*. 2015. DOI: `10.48550/ARXIV.1511.06581`. URL: `https://arxiv.org/abs/1511.06581`.

[6] Yuri Burda et al. *Exploration by Random Network Distillation*. 2018. DOI: `10.48550/ARXIV.1810.12894`. URL: `https://arxiv.org/abs/1810.12894`.

[7] Ian Osband et al. *Deep Exploration via Bootstrapped DQN*. 2016. DOI: `10.48550/ARXIV.1602.04621`. URL: `https://arxiv.org/abs/1602.04621`.

[8] Shixiang Gu et al. *Continuous Deep Q-Learning with Model-based Acceleration*. 2016. DOI: `10.48550/ARXIV.1603.00748`. URL: `https://arxiv.org/abs/1603.00748`.

[9] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015. DOI: `10.48550/ARXIV.1506.02438`. URL: `https://arxiv.org/abs/1506.02438`.

[10] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. DOI: `10.48550/ARXIV.1707.06347`. URL: `https://arxiv.org/abs/1707.06347`.

[11] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. DOI: `10.48550/ARXIV.1801.01290`. URL: `https://arxiv.org/abs/1801.01290`.

[12] Xinyue Chen et al. *Randomized Ensembled Double Q-Learning: Learning Fast Without a Model*. 2021. DOI: `10.48550/ARXIV.2101.05982`. URL: `https://arxiv.org/abs/2101.05982`.