Reinforcement Learning SS 2023
Instructor: Georg Martius `<georg.martius@uni-tuebingen.de>`
TAs: Marco Bagatella, Sebastian Blaes, Marin Vlastelica, Pavel Kolev and Andrii Zadaianchuk
Final Project
Competition: 09.08.2023, In-person get-together 10.08.2023 (time TBA).
Report, Video and Code is due: 11.08.2023 23:55
Last Q&A session: July. 25th.
July 7, 2023, Version 1.1

1 Reinforcement Learning Project

The final project of the lecture will be on developing a Reinforcement learning agent for a small game. You can form teams of 2 or 3 people. You will implement different reinforcement learning algorithms and evaluate their performance on simpler tasks first and then apply them to a simulated `hockey` game. We are using the gymnasium API (former Open AI gym), see `https://gymnasium.farama.org/` with a custom environment. It uses the same packages as we had for the exercises. The code for the hockey game is at the git repository `https://github.com/martius-lab/laser-hockey-env` (*HockeyEnv* / `Hockey-v0` / `Hockey-One-v0`, *not LaserhockeyEnv*). Try out the notebook `Hockeyenv.ipynb`.
All teams will compete against each other in the game at the end. There are intermediate checkpoints, see below, that we will discuss in the remaining tutorial sessions.
Please register your team as soon as possible, but not later than 30th of June at this google form[1]. You will then get account information for a teaching cluster and for the tournament server (in due time).
For the final evaluation, you have to prepare:

(a) A report with:
  (a) an introduction, including a description of the game/problem **(1/2 page)**
  (b) a methods section, including relevant implementation details, modifications and math, e.g. the objective functions of the implemented algorithms **(min 1 page per algorithm/person)**
  (c) an experimental evaluation for all the methods and environments, including the performance against the basic opponent (weak) **(min 1 page per algorithm/person)**

[1] `https://forms.gle/tRzxWyCZF9GzQRkT7`

(d) a final discussion, including a comparison of the different algorithms **(≈ 1 page)**

For a single person, the report should not be longer than **5** pages excluding references.

For a team of two people, the report should not be longer than **8** pages excluding references.

For a team of three people, the report should not be longer than **11** pages excluding references.

Each **team member** should have **one algorithm** implemented and his/her independent contribution should be clearly marked in the **report and the source code. Significant modifications of the same base algorithm are possible, for instance, based on DQN, two modifications mentioned in the *Rainbow* paper [3] would be significant per person. For instance, just adding some Gaussian noise to the actions would not be significant.**

Update in Version 1.1: Regarding the usage of existing code you have the following options:

- **you implement an algorithm yourself (except the ones we had in the exercises), or**
- **you base your implementation on some existing code (needs to be specified where it comes from) and you make a non-trivial modification. A non-trivial modification is something that is or might be a main contribution in a paper.**

If in doubt, contact the tutors during the exercise sessions in advance.

(b) A video-presentation:

- **3 min for a single person**
- **4 min for two-person team**
- **6 min for three-person team**

(c) The source code

The submission deadline for the report, video, and code is on **11.08.2023 23:55** via ILIAS.
The code needs to be running in the tournament starting on the 09.08. from 10am onward. A test run will happen earlier and will be announced.

Requirements:

(a) Teams of two should implement 2 algorithms

(b) Teams of three should implement 3 algorithms

(c) In order to pass the exam report, presentation and code have to be handed in on time.

(d) The code has to run and if the hockey does not work at least a simple environment must be solved.

Usage of GPT Large language models are great tools for various tasks and can help you be more efficient, however, you need to know how to do things yourself properly first, before relying on the LLMs. Since this is part of YOUR education, the usage of chatGPT, or similar, for writing the report is not allowed, we will run software to detect its usage.

Grading

(a) The mark will be determined based on all parts. You are expected to deliver a nicely written report, a clear presentation, and a good performance.

(b) The final mark will be computed from the individual scores with the following weighting: 60% report, 20% presentation, 20% performance

(c) The performance is measures against the basic opponent (in weak mode). You should not be stressed about being better than the other teams to get top score. However, the top 3 best performing teams in the competition get a bonus (0.35 marks) (and the performance score will be 1.0 anyway).

1.1 Checkpoint 1: Get your algorithms up and running

Start with the Pendulum-v0 from exercise 8 and 9 or with other simple environments. Be aware that some environments (such as the pendulum) contains an exploration problem that is not necessarily quickly solved by all algorithms. You can also try LunarLander-v2 (exercise 9) or HalfCheetah. Important is to see that the reward is optimized and the behavior is reasonable. This should allow you to debug your code.
Implement your algorithms of choice. I recommend to consider off-policy algorithms: Dueling Deep Q-learning (DDQN) [5], Deep deterministic policy gradient (DDPG) [4] or Twin Delayed DDPG (TD3) [1], Soft/Natural Actor Critic (SAC [2]). The versions from our exercises can be a good starting point, but need to be modified (as we provided a solution). Check out the *Rainbow* paper [3] for possible additions.
Make appropriate analysis and track your performance etc. Don't forget this procedure during the rest of the project. Remember that you want to create a report with plots giving detail about the training, comparisons etc.

1.2 Checkpoint 2: Hockey – learning to handle the puck

Start working on the Hockey game[2]. The repository provides the environment and a little notebook to see how the environment works. It is actually installable via pip, see the `README.md` file.
In order to learn how to play hockey there is a small training camp for the RL-agents to go through. These are:

TRAIN DEFENSE defending your goal against incoming shots (other agent is static)

NORMAL normal gameplay against another agent.

You can enable the game-modes with
`HockerEnv(mode=NORMAL|TRAIN_SHOOTING|TRAIN_DEFENSE)`.
Start with training the defense. Make sure you checkpoint your trained models.

1.3 Checkpoint 3: Play in normal mode against the basic opponent

The basic opponent has a weak and a strong mode. Train your agent against it in normal game mode. Think how to exploit the fact that you are using an off-policy algorithm.

1.4 Checkpoint 4: Self-play

Let your agents play against each other in normal game mode. Make appropriate analysis and track your performance etc. Experiment with different tournament modes.

1.5 Final

The tournament server, as described below, will be restarted on Wednesday 9th of August at 10am. From that time point on your client should be running and play against others. You can update the client during the tournament. The tournament will finish at 10am on Thursday the 10th.
For the final tournament, teams have to connect to the tournament server with a special client (v1.0)[3]. The host address of the server is
`al-hockey.is.tuebingen.mpg.de` and the port is `33000`. It also has an http interface.
To be compatible with the provided client, your agent needs to implement the abstract method `remote_act` of the `RemoteControllerInterface` provided in `client/remoteControllerInterface.py` that expects an observation as input and returns an action, both in form of a `numpy.ndarray`.

[2] `https://github.com/martius-lab/laser-hockey-env`
[3] `https://nextcloud.tuebingen.mpg.de/index.php/s/nGxF4zM8sDndJLL`, available soon.

See `remoteBasicOpponent.py` from the same directory for a reference implementation.

Teams can register at most 3 players, one for each algorithm/team member, on the server. Use the following name scheme to distinguish between the different algorithms: `teamname_name`, where name can be the name of the person who implemented the algorithm or the name of the algorithm (e.g. DDP, TD3). Later, you can connect multiple instances with the same name to the server which will play in parallel (see the example below). We suggest to check whether you can run the client on the university pc pool via a ssh session. In this way, your computer does not have to run constantly over one or two days.

The tournament server will constantly pair up teams which will play against each other by receiving observations from and sending actions to the server. For this to work, the client needs to run in the background permanently. We recommend to use a terminal multiplexer such as tmux[4] or screen[5] which allow you to keep a process running (by detaching the tmux/screen session) after closing a terminal/ssh session.

Already before the final tournament, you can play on our test server. Use port `33000` and the same host address as above.

The leaderboard can be accessed via `http://al-hockey.is.tuebingen.mpg.de/`.

To participate in the final tournament, your agent should primarily be trained by maximizing the reward provided by the environment (including the additional reward info and via reinforcement learning). You can use other techniques such as behavioral cloning as long as they do not contribute the most to your implementation efforts/training of the agent.
How to run multiple instances of a client in a tmux session via ssh:

- Connect via ssh from your local machine to the remote machine:
  ```
  local> ssh $remote_machine
  ```
- Change to the directory with the client:
  ```
  remote_machine> cd $dir_with_client
  ```
- Start a tmux session:
  ```
  remote_machine> tmux
  ```
- Start the first instance by running your script

[4] `https://github.com/tmux/tmux/wiki`
[5] `https://www.gnu.org/software/screen/`

- Press <ctrl>-b c (first control+b together and afterwards c) to open a new window. Start the next instance either with the same name (they will play in parallel) or with a different name
- If all your instances are up and running you can detach the tmux session with <ctrl>+b d and logout.
- If you login again later, you can re-attach your session with:

  ```
  remote_machine> tmux a
  ```

References

[1] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[2] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[3] Matteo Hessel, Joseph Modayil, H. V. Hasselt, T. Schaul, Georg Ostrovski, W. Dabney, Dan Horgan, B. Piot, Mohammad Gheshlaghi Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

[4] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[5] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.