Reinforcement Learning WS 2018/19
Instructor: Georg Martius <georg.martius@tuebingen.mpg.de>
TA: Sebastian Blaes <sblaes@tuebingen.mpg.de>
Final Project
Due-date: 08.02.2019
Intermediate steps will be discussed in the session on Jan. 11th onwards
February 8, 2019, Version 4.1

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. For teams of 3 there are special requirements detailed below. If you want to persue a different task, let us know and we decide whether this is okay. You will implement different reinforcement learning algorithms and evaluate their performance ona simple task first and then apply them to a simulated `laser-hockey` game. We are using Open AI gym, see `https://gym.openai.com/` with a custom environment. Instructions on installing gym are at `https://github.com/openai/gym`.The code for the laserhocket game for this project is at the git repository `https://github.com/martius-lab/laser-hockey-env`. We will need to change the code to bugs or change parameters, so please stay tuned. We will make releases,such that you know when things changed.
All teams will compete against each other in the game at the end. The tournament mode for different teams is not yet implemented, but we will provideit. There are intermediate checkpoints, see below, that we will discuss in theremaining recitation sessions.
For the final evaluation, you have to prepare:

(a) A report with:

   (a) an introduction, including a description of the game/problem **(min 1/2 - 1 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 **(min 1 page per algorithm/person)**
   (d) a final discussion, including a comparison of the different algorithms **(min 1 page)**

   For a single person, the report should not be longer than **6** pages.

   For a team of two people, the report should not be longer than **10** pages.

For a team of three people, the report should not be longer than **14** paes.

Each team member should have one algorithm implemented and his/her independent contribution should be clearly marked in the report.

(b) A presentation ∼8 min (for 2-person teams and 10min for 3-person teams)

(c) The source code

Everything needs to be finished/presented at the exam day: 08.02.2019
The submission deadline for the report and code is on **February, 9th, 2019 at 10pm** by email.
The tournament will happen before, exact details will follow.
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 the simple pendulum environment must be solved.

(e) 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.

1.1 Checkpoint 1: Get your algorithms up and running

Start with the Pendulum-v0 from the gym suite. Implement your algorithms of choice. I recommend to consider: Deep Q-learning (DQN) [3], Deep deterministic policy gradient (DDPG) [2], Soft/Natural Actor Critic (SAC [1]/NAC [4]), proximal policy gradient (PPO) [5], Policy Gradient by Parameter Exploration (PGPE) [6]. The version of DQN that you implemented for your last exercise would be a good starting point. Keep in mind that the full DQN also has a target Q network. Also, the Q-network outputs a value for each action at once. For algorithms using discrete actions, transform the continuous actions into a few discrete actions. The task is solved if you get on average a reward above -300. Make approriate analysis and track your performance etc. Dont 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: Laser-hokey learning to handle the puck

Start working on the laser-hockey game (`https://github.com/martius-lab/laser-hockey-env`). The repository provides the environment and a little notebook to see how the environment works.
In order to learn how to play laser-hokey there is a small training camp for the RL-agents to go through. These are:

TRAIN SHOOTING hitting a static ball into the goal (other agent is static)

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

NORMAL normal gameplay against another agent.

You can enable the game-modes with
`LaserHockerEnv(mode=NORMAL|TRAIN_SHOOTING|TRAIN_DEFENCE)`.

1.3 Checkpoint 3: 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.

1.4 Final

On **Thursday, 7th, 2019** will be the final tournament. Teams have to connect to the tournament server with a special client (v1.0)[1] on Thursday at the latest and on Wednesday at the earliest. The host address of the server is `al-hockey.is.tuebingen.mpg.de` and the port is `33001`. To be compatible with the provided client, your agent should at least implement a function `act` that expects an observation as input and returns an action, both in form of a `numpy.ndarray`.
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[2] or screen[3] which allow you to keep a process running (by detaching the tmux/screen session) after closing a terminal/ssh session.
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_algorithm`. 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.
Already before Wednesday/Thursday, you can play on our test server. Use port `33000` and the same host address as above if you want to play on the test server.
To participate in the final tournament, your agent should primarily be trained by maximizing the reward provided by the environment (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.

[1] `https://owncloud.tuebingen.mpg.de/index.php/s/WDnWNyzJcb5WdoC`
[2] `https://github.com/tmux/tmux/wiki`
[3] `https://www.gnu.org/software/screen/`

If you want to participate in the final tournament, please send a mail to us with your team name and team members until Wednesday. On Friday, the day of the presentations, we will analyse the tournament and announce the winner of the competition. **Also on this Friday, please bring your presentation with you on a usb stick and in form of a pdf file.**

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:

  ```
  remote_machine> python -m tournament_player --client-id teamname_algorithm1
  --host al-hockey.is.tuebingen.mpg.de --port 33001
  ```

- 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

  ```
  remote_machine> python -m tournament_player --client-id teamname_algorithm2
  --host al-hockey.is.tuebingen.mpg.de --port 33001
  ```

  to connect an instance with a different algorithm. Repeat.

- 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
  ```

Information regarding the university linux pools (ZDV):

- General information can be found here[4]
- The host address of the pool is `linux11.zdv-pool.uni-tuebingen.de` (until linux14)
- `screen` instead of `tmux` is already pre-installed. The handling is quite similar to tmux. Some of the bindings might be different.
- As far as we know, you should be able to detach from and re-attach to screen sessions.

If you encounter any problems, contact Senastian Blaes via `sblaes@tuebingen.mpg.de`.

[4] `https://uni-tuebingen.de/einrichtungen/zentrum-fuer-datenverarbeitung/dienstleistungen/clientdienste/pools/zdv-pools-linux/`

References

[1] 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, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[2] 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.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[4] J. Peters and S. Schaal. Natural Actor-Critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[6] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.