

# Reinforcement Learning

## Exercise 1

September 15, 2019

### 1 Introduction

The goal of this introductory exercise is to set up all the software and libraries that will be used throughout the rest of the course, as well as to get some intuition behind the basic reinforcement learning setup (states, actions, rewards, policy, etc.).

### 2 Setting things up

In this course, we will be using Python 3 along with some libraries: OpenAI Gym, NumPy and PyTorch. Let's start off by setting everything up.

OpenAI Gym is officially supported only on Linux and MacOS/OS X. If you don't have any of them on your machine, you can either set up a virtual machine (using VirtualBox or VMWare) or use one of the Aalto computers (for example in the Panikki room in CS building; they are also available for remote connections via SSH).

We received some reports of people successfully installing OpenAI Gym on Windows - for some of them it worked flawlessly, and some suffered from minor glitches (but were still able to complete the exercises). The general instructions for installing on Windows are also presented, but keep in mind that this procedure is experimental.

#### 2.1 Python 3

On virtually all Linux systems, Python 3 is either installed by default or can be fetched directly from the official repositories. You can check if Python 3 is installed (together with its exact version) by running `python3 --version`

If it's not installed, the exact commands needed to set it up depend on your Linux distro:

### 1. Debian, Ubuntu and their derivatives:

Open the terminal and run

```
sudo apt-get install python3 python3-pip
```

### 2. Arch (and derivatives)

Open the terminal and run

```
sudo pacman -S python3
```

On Windows, Python can be downloaded from <https://www.python.org/downloads/>

## 2.2 OpenAI Gym, NumPy, Matplotlib...

If you've installed pip, installing OpenAI gym can be installed with the following command:

```
sudo pip3 install gym numpy matplotlib seaborn pandas
```

If you don't have root access on your machine (for example on university computers) or if you're using a distro that insists on installing pip packages locally on your user account (such as Gentoo), you can run pip as follows:

```
pip3 install --user gym numpy matplotlib seaborn pandas
```

On Windows, the command should be

```
pip3 install gym numpy matplotlib seaborn pandas
```

For more details, see <https://gym.openai.com/docs/>, <https://www.scipy.org/scipylib/download.html> and <https://matplotlib.org/users/installing.html>.

## 2.3 PyTorch

Installing PyTorch on Linux should narrow down to running

```
sudo pip3 install torchvision or pip3 install --user torchvision.
```

On Windows, you can try the following:

```
pip3 install torch==1.2.0 torchvision==0.4.0 -f https://download.pytorch.org/whl/torch_stable.html
```

For installation instructions on other platforms (or with tools like anaconda), refer to <https://pytorch.org>

## 2.4 Working remotely

If you're experiencing some issues setting things up on your computer, you can use the computers in the CS and Maari buildings. To set things up, you should follow the standard Linux procedures (bare in mind that — as a student — you don't have sudo rights on those machines, so pip packages have to be installed with the `--user` switch).

You can later work remotely by using SSH to connect to these machines. First, you should open SSH connection to `kosh.aalto.fi` or `lyta.aalto.fi`, log in using your Aalto username and password, and — from Kosh/Lyta — connect to one of the classroom computers. The full list is available at <https://www.aalto.fi/en/services/linux-computer-names-in-it-classrooms>.

### 2.4.1 Connecting from Windows

To use SSH on Windows, the easiest way is to download and install PuTTY. If you want to observe the progress of your training, you will additionally have to install an X server for Windows (Xming is a good option) and enable X11 forwarding when connecting from PuTTY.

### 2.4.2 Connecting from Linux/Mac

On Linux and Mac, connecting basically narrows down to invoking `ssh username@kosh.aalto.fi` or `ssh username@lyta.aalto.fi`, entering your Aalto password, and from there connecting to any of the lab machines by running `ssh machine_name`.

If you wish to visually follow the training process, you should pass the `-X` flag to the `ssh` command. For example, if you're logging in to `kuukivi`, you would first run `ssh -X user@kosh.aalto.fi` followed by `ssh -X kuukivi`. Sometimes you may have to use `-Y` flag instead of `-X` (depends on your system configuration).

You can make the process easier by appropriately setting your SSH configuration in `/.ssh/config`:

```
Host kosh
    HostName kosh.aalto.fi
    User aaltousername
    ForwardX11 yes
    ForwardX11Trusted yes
```

Then, add similar definitions for the machines you want to use (example for `bit`):

```
Host bit
    HostName bit
    User aaltousername
    ProxyCommand ssh -q -A kosh nc -q0 %h %p
    ForwardX11 yes
    ForwardX11Trusted yes
```

After adding these setting, you can login to `bit` simply with `ssh bit`.

You can also copy your machine's RSA identification, such that you don't have to provide your password every time you log in with `ssh-copy-id kosh` (**don't do it on public computers**). For security reasons, this requires appropriate permissions on your SSH-related files and directories to work (e.g., your home directory and the `authorized_keys` file cannot be writable by other users).

On Linux, you may also have to enable indirect GLX rendering in your X server configuration; this can be done by adding the following to the `/etc/X11/xorg.conf` file on your local machine (create the file if it doesn't exist):

```
Section "ServerFlags"
    Option "AllowIndirectGLX" "on"
    Option "IndirectGLX" "on"
EndSection
```

You will have to restart your X server (or reboot) afterwards.

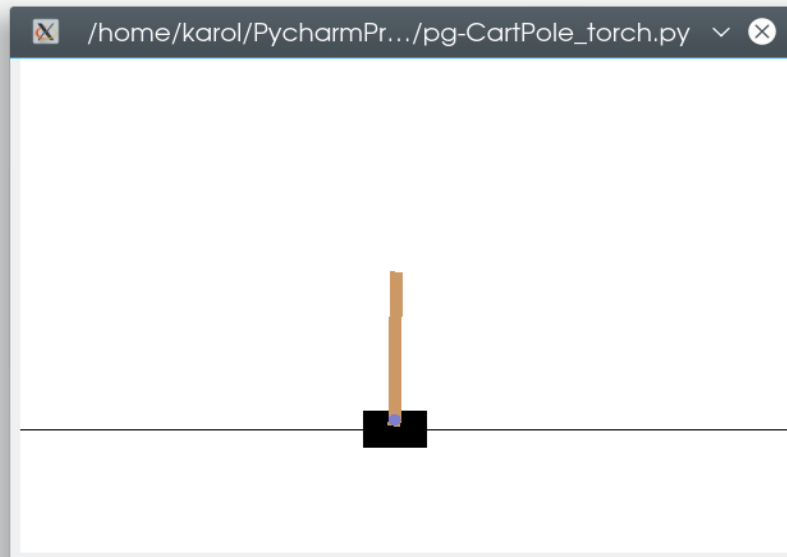


Figure 1: The Cartpole environment

Now, let's move on to the actual exercise.

### 3 States, observations and reward functions

In this exercise, you are given a Python script (`cartpole.py`) which instantiates a *Cartpole* environment and a RL agent that acts on it. The `agent.py` file contains the implementation of a simple reinforcement learning agent; for the sake of this exercise, you can assume it to be a black box (you don't need to understand how it works, although you are encouraged to study it in more detail). You don't have to edit that file to complete this exercise session - all changes should be done in `cartpole.py`.

#### 3.1 Cartpole

The *Cartpole* environment consists of a cart and a pole mounted on top of it, as shown in Figure 1. The cart can move either to the left or to the right. The goal is to balance the pole in a vertical position in order to prevent it from falling down. The cart should also stay within limited distance from the center (trying to move outside screen boundaries is considered a failure).

The observation is a four element vector

$$o = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix}, \quad (1)$$

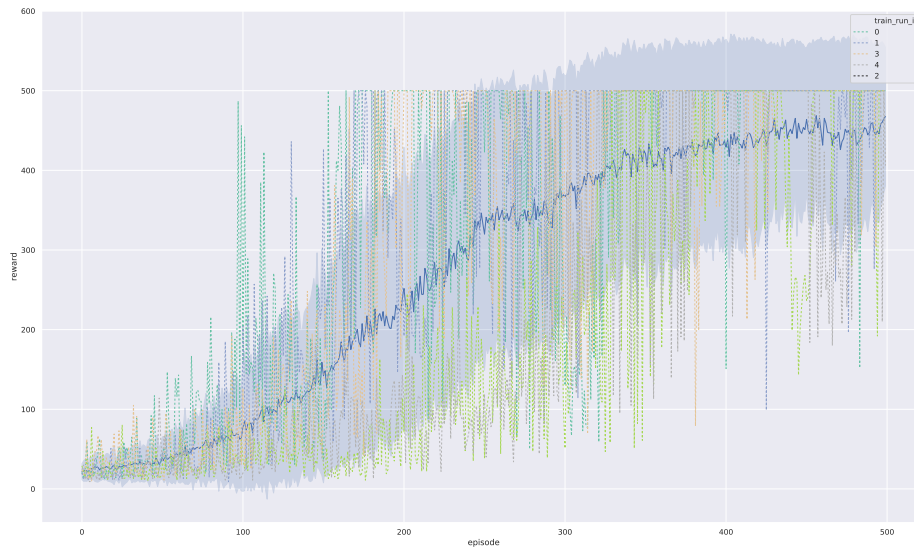


Figure 2: Variance in performance between multiple repetitions of the experiment.

where  $x$  is the position of the cart,  $\dot{x}$  is its velocity,  $\theta$  is the angle of the pole w.r.t. the vertical axis, and  $\dot{\theta}$  is the angular velocity of the pole.

In the standard formulation, a reward of 1 is given for every timestep the pole remains balanced and the task is considered to be solved if the pole is balanced for 200 timesteps.

### 3.2 Exercise

Run the `cartpole.py` script. See how it learns to balance the pole after training for some amount of episodes. You can use the `--render_training` argument to visualize the training (it will run slower when rendering). When the training is finished, the models are saved to `CartPole-v0_params.mdl` and can be tested by passing `--test path/to/model/file.mdl`.

Now, increase the maximum episode length to 500 (see line 130 in `cartpole.py`) and test the previously trained model.



**Q1:** Can the same model, trained with 200 timesteps, balance the pole for 500 timesteps? Why/why not?

#### 3.2.1 Repeatability

Figure 2 shows the mean and standard deviation throughout 100 independent training procedures. You can notice that there is a large variance between the runs of the script.

**Q2:** Why is this a case? What are the implications of this, when it comes to comparing reinforcement learning algorithms to each other?



You can generate a similar plot by running the `multiple_cartpoles.py` script. If the script is slow, try disabling PyTorch multithreading (by running the script with `OMP_NUM_THREADS=1`)


### 3.2.2 Reward functions

Write a new reward function and use it instead of the default reward returned by the environment (see the `new_reward` function for an example). Write different reward functions to do the following:

1. to give the agent extra incentive to balance the pole close to the center of the screen (close to  $x = 0$ ),
2. to incentivize the agent to balance the pole in an arbitrary point of the screen ( $x = x_0$ ), with  $x_0$  passed as a command line argument to the script,
3. make the agent try to keep moving the cart as fast as possible, while still balancing the pole.


Use the observation vector to get the quantities required to compute the new reward (such as the speed and position of the cart). If you feel like your model needs more time to train, you can leave it running for longer with the command line argument `--train_episodes number`, where `number` is the amount of episodes the model will be trained for (default is 500).

Once you've settled on a reward function that works fine, repeat the training at least 10 times using `multiple_cartpoles.py` (adjust the `--num_runs` and `--train_episodes` parameters). Observe the results.

**Q3:** How does changing the reward function  impact the time needed for training?

## 4 Submitting

The deadline to submit the solutions through MyCourses is on Sunday, 29. 09 at 23:55. Example solutions will be presented during exercise sessions the following week (30. 9 and 2. 10).

Your submission should contain **answers to the questions** asked in the text, **the code** with solutions used for the exercise and the **trained model files** for each reward function (remember to report what reward functions were used). 

If you need help or clarification solving the exercises, you are welcome to come to the exercise sessions in weeks 38/39.