Reinforcement Learning
Exercise 1

September 16, 2018

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.

2.1 Getting your Linux set 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). Some people reported getting OpenAI Gym to run on Windows as well; feel free to try it (and share your experiences if you do).

2.2 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`

2.3 OpenAI Gym

If you've installed pip, installing OpenAI gym can be installed with the following command:
`sudo pip3 install gym`

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`

For more details, see `https://gym.openai.com/docs/`.

2.4 NumPy

NumPy should be pulled in automatically when Gym was installed; if that's not the case, run the previously used pip command, replacing `gym` with `numpy`.

2.5 PyTorch

Installing PyTorch should narrow down to running
`sudo pip3 install torchvision` or `pip3 install --user torchvision`.

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

2.6 Cartpole

Run the `cartpole.py` script. See how it learns to balance the pole after training for some amount of episodes. Use the `--render_training` argument to visualize the training (it will run slower when rendering).

If you are having difficulties running the script, please make sure your Python environment is set up correctly.

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`.

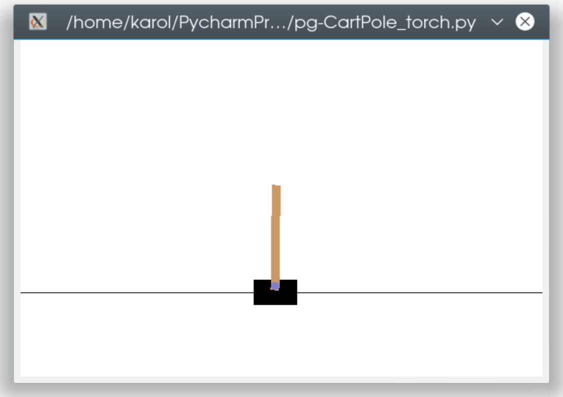

Figure 1: The Cartpole environment

3.1 Cartpole

The *Cartpole* environment consists of a cart and a pole mounted on top of it (fig. 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

$$o = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix} \tag{1}$$

,

where x is the position of the cart, $\dot{x}$ is its velocity, θ 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

First, increase the maximum episode length to 500 (see line 111 in `cartpole.py`). Can the agent still learn to balance the pole? Does it take longer to train?

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 while 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 1000).

Observe the results. 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, 30. 09 at 23:55. Example solutions will be presented during exercise sessions the following week (1. 10 and 3. 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.

If you need help or clarification solving the exercises, you are welcome to come to the exercise sessions in weeks 38/39 (registration for exercise groups is open in MyCourses).