Reinforcement Learning Spring 2023 Prof. Dr. Paul Swoboda

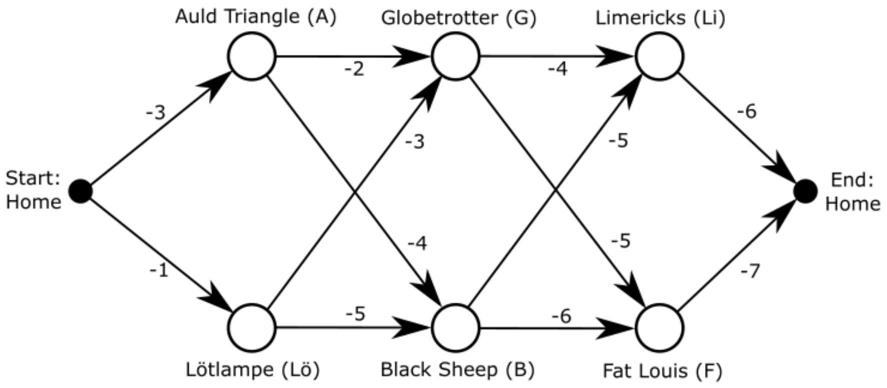
Exercise Sheet 1 - Due date 9.3.2023, 10am

General Instructions

- Send exercise solutions in one email to paul.swoboda@uni-mannheim.de with the Subject `[RL 2023] Exercise ${nr}, ${your names}`
- You can work on the exercise in groups of up to three people. Do not forget to write all the group participants in the email header.
- Whenever writing down is requested: Send your solutions in a single pdf. You can scan your handwritten writing.
- The practical solutions should be in the indicated files, as will be made clear in each exercise. I must be able to run it without modifications on my computer. You are always allowed to use numpy and matplotlib as well as the gym library (except where it defeats the purpose of the exercise) and later on pytorch. Apart from that you are only allowed to use the packages that are included in the beginning of the file with the solution skeleton. More libraries can be allowed upon request.
- Sometimes I will ask for plots of how the algorithms did (e.g. training curves). You can screenshot them and paste them into the solution pdf.

Question 1 – Policy Evaluation (4 points)

Let us say you drink three beer in three different pubs. There are six pubs available in town, you start at home and will end up at home. The problem is depicted in the following picture:



In our first example we follow the 50/50 policy. So after drinking in a pub - e.g. Auld Triangle, there is a 50 % probability to go "up" to the Globetrotter and 50 % probability to go "down" to the Black Sheep. Evaluate the state values using policy evaluation ($v_{\mathcal{X}} = \mathcal{R}_{\mathcal{X}} + \gamma \mathcal{P}_{xx'} v_{\mathcal{X}}$):

$$\begin{bmatrix} v_1^{50/50} \\ \cdot \\ \cdot \\ \cdot \\ v_n^{50/50} \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1^{50/50} \\ \cdot \\ \cdot \\ \cdot \\ \mathcal{R}_n^{50/50} \end{bmatrix} + \gamma \begin{bmatrix} p_{11}^{50/50} & \cdots & p_{1n}^{50/50} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ p_{n1}^{50/50} & \cdots & p_{nn}^{50/50} \end{bmatrix} \begin{bmatrix} v_1^{50/50} \\ \cdot \\ \cdot \\ \cdot \\ v_n^{50/50} \end{bmatrix}$$

The rewards are given as negative numbers next to the arrows and represent the distances between two bars as a penalty. In this exercise we will set $\gamma = 0.9$. In the shown problem we have $n = 8$ states (pubs, including start-home and end-home), ordered as given by the state space:

$$\mathcal{X} = \left\{ \begin{array}{c} \text{Start: Home} \\ \text{Auld Triangle} \\ \text{Lötlampe} \\ \text{Globetrotter} \\ \text{Black Sheep} \\ \text{Limericks} \\ \text{Fat Louis} \\ \text{End: Home} \end{array} \right\}$$

In file `exercise01-beer.py` write down the formula for calculating the state values.

Question 2 – Richardson Iteration (4 points)

Derive the value function of the 50/50 policy from Question 1 by implementing the Richardson iteration.

Write your solution to `exercise01-beer.py`.

Question 3 – Dynamic Programming (4 points)

We want to find the value function for the 50/50 policy and also find the optimal policy for visiting the pubs using dynamic programming.

Implement value iteration $v^*_{i+1}(x_k) = \max_a(r_{k+1} + v^*_i(s_{k+1}))$ and compute the state values of the optimal policy v^*.

Write your solution to `exercise01-beer.py`.

Question 4 – Multi-Armed Bandit (8 points)

A bandit is a tuple $\langle \mathcal{A}, \mathcal{R} \rangle$, where $\mathcal{A}$ is a finite set of actions (think machines you can use in a casino) and $\mathcal{R}$ is a random reward produced by using a certain machine. We can think of a bandit as an MDP without states. Crucially, we do not know the rewards and we must explore the bandit to get the to know which action to take for largest expected reward. Also the reward is stochastic: it can vary every time we take an action.

In this exercise we will look into a simple ten-armed bandit – you can take 10 different actions per step. Each action gives a random payout according to a Gaussian distribution with per-action dependent mean and variance.

Implement a good strategy (whichever you like) to get a high sum of rewards over 5000 steps. Do not give me an algorithm which learns the exact behaviour of the bandit used in the exercise file: I will test your algorithm on a new bandit and see how well it performs there. Also please do not consult RL resources on good algorithms for bandits – go along with your own ideas. You will get full points as soon as your algorithm is doing something somewhat reasonable. Write down your ideas and what worked and what not in a short report.

Attention: The best performing algorithm that is not a writedown from existing literature will get an honorary mention!

Hint: In RL people think about *exploration* (find out which actions are good) and *exploitation* (take good actions to obtain high rewards).

Write your solution to `exercise01-bandit.py`.

We will use the `gym_bandits` environment. To install it execute `git clone https:github.comJKCooper2gym-bandits.git`, then `cd gym-bandits` and finally `pip3 install .`