Reinforcement Learning
Exercise 3

October 15, 2018

1 Q-Learning

In this part of the exercise, you'll be applying grid-based Q-learning to the *Cartpole* and *LunarLander* environments.

1.1 Cartpole

Recall the Cartpole environment from exercise 1.

Task 1 Implement Q-learning for the Cartpole environment in the file `qlearning.py`. Compare using a constant value for ϵ to reducing the value of ϵ over time (use the *greedy in limit with infinite exploration* formula from the lecture).

Hint: The states in Cartpole are continuous, while grid-based methods can only be directly applied to discrete state spaces. You can overcome this issue by discretizing Cartpole states to a finite grid (example values can be found in `qlearning.py`).

Hint: Reasonable values for parameters are $\alpha = 0.1$ and $\gamma = 0.99$. For GLIE, aim at reaching ϵ somewhere in the 0.01-0.05 range by the end of training.

1.2 Lunar lander

The *Lunar lander* environment is shown in Figure 1. The goal is to make the lunar lander land on the ground between two flag poles. The agent receives a positive reward for moving towards the landing pad, landing, etc. A negative reward is given for firing the main engine (more fuel-efficient policies are better) and for crashing.

Four actions are available: firing the left/right/main engines, or doing nothing (free fall). The observation vector consists of 6 continuous and 2 discrete values:

$$o = \begin{pmatrix} x & y & \dot{x} & \dot{y} & \theta & \dot{\theta} & c_l & c_r \end{pmatrix}^T, \tag{1}$$

where x and y are the coordinates of the lander, $\dot{x}$ and $\dot{y}$ its velocities, θ represents the rotation angle and $\dot{\theta}$ the angular velocity of the lander. Two discrete values c_l and c_r indicate whether the lander's legs are in contact with the ground (0 or 1).

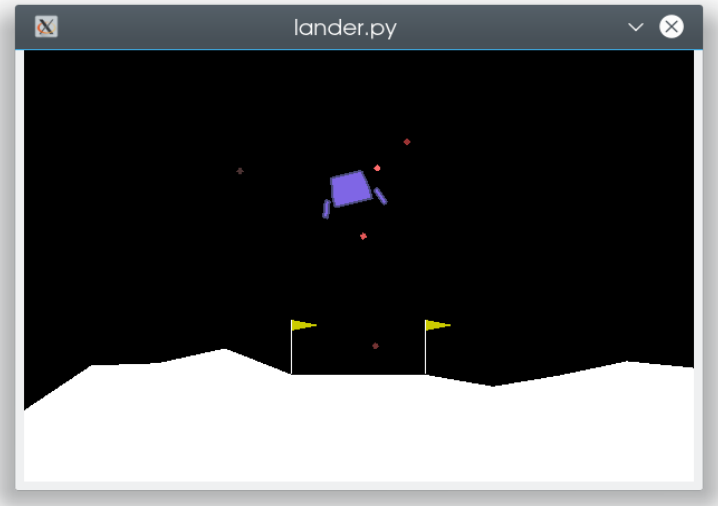

Figure 1: The Lunar lander environment.

Task 2 Modify your code and try to apply it in the Lunar lander environment. Run it for 20000 episodes (enough for the Carpole to learn) and see if the lander is able to learn any useful behaviour. Why/why not?

Hint: If you're getting errors about Box2D not being installed, you have to install the `box2d-py` package with `pip3`. You may also have to additionally install Swig from your distro repositories (`apt-get install`/`pacman -S`/`emerge`, etc.).

2 Function approximation

In many real world scenarios, the dimensionality of the state space may be too high to compute and store the Q-values for each possible state and action in a Q-table. Also - many states may be somewhat *similar to each other*, for example - in case of the lunar lander - crashing into the ground is always bad, no matter what the exact x, y coordinates or the lander angle are. The simple grid discretization used in previous exercise does not, in any way, share information between neighbouring states.

The state value and action value functions can be learned by using a function approximation method. There are many approaches to function approximation, perhaps the most famous one being neural networks. In this part of the exercise, you will use a simpler method called radial basis functions.

Task 3 Implement function approximation using radial basis functions for the LunarLander environment. Use the `rbf_approximation.py` file as a template. Is the lander able to reach its goal now?

Task 4 Use the computed approximation to plot a heatmap of $Q(s, a)$ for all possible lander positions (x and y) and all four actions (0-3). Assume other elements of the state vector to be 0 when computing the heatmaps. Analyze them in a few sentences.

Question 1 Change the number and parameters of the radial basis functions. What do you expect? What do you observe?

Question 2 Would it be possible to use linear approximator functions?

3 Submission

Your report should contain answers to questions posed in the text and an analysis of results. Furthermore enclose the parameters you used in your code to achieve the outcomes you describe in your answers. Include trained models (the files should be saved when training is finished) and the code used for the exercises. Deadline to submit your answers is 28.10, 23:55.

If you need help solving the tasks, you are welcome to visit the exercise sessions on Monday and Wednesday.

Good luck!