Robot Learning
Exercise 2a

November 16th, 2023
TAs: Gabriele Tiboni (*gabriele.tiboni@polito.it*), Andrea Protopapa (*andrea.protopapa@polito.it*), Elena Di Felice (*s303499@studenti.polito.it*)

1 Introduction

Throughout this exercise session, we will take a first look at a reinforcement learning environment and its components. In particular, we will deal with the Cart-Pole environment, a classic problem in reinforcement learning. Our primary goal is to implement and evaluate various control strategies, including a Linear Quadratic Regulator (LQR) baseline or random policies, to understand their impact on the system's performance and learn about the fundamental concepts of reinforcement learning, laying the groundwork for more advanced reinforcement learning techniques in subsequent exercises.

To proceed with the assignment, visit the dedicated starting code at the dedicated webpage and follow the steps listed below in this document. Note: if you haven't joined the course's github classroom yet, you will be asked to select your name when you visit it the first time.

As you follow the steps in this document, make sure you're able to answer **all** the *guiding questions* listed at the end of each step.
Your submission should consist of (1) a **PDF report** containing **answers to the tasks and questions** asked in these instructions and **reward plots**, (2) the **code with solutions** used for the exercise and (3) the **trained model files** for each reward function (remember to report what reward functions were used, see Section 4.3).

The **deadline to submit** the solutions through the Github Classroom will be defined in the next few days.

2 Cartpole Gym environment: states, observations

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 a limited distance from the center (trying to move outside screen boundaries is considered a failure).

The observation is a four element vector:

$$s = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix}$$

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.

Gym environment documentation

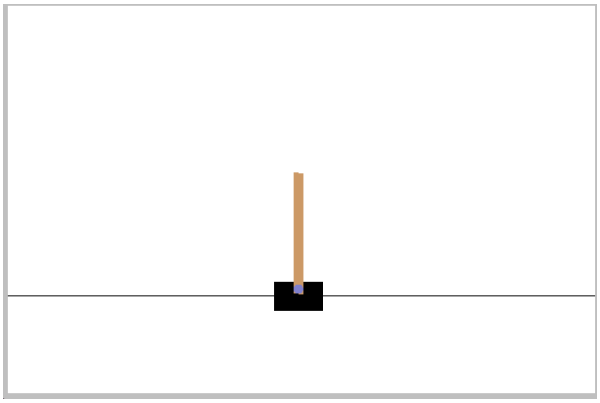

Figure 1: The Cartpole environment

3 LQR

For the first assignment, you are tasked with testing a baseline using a Linear Quadratic Regulator (LQR), resulting in an optimal control gain K. The baseline is already implemented in the provided Python script (`cartpole_lqr.py`). Since the system exhibits nonlinear dynamics, a linear approximation of the system is available through the `linearized_cartpole_system(...)` function. Given the linear state space representation, the controller K is optimized by minimizing the following cost function:

$$J = \int_0^\infty \left(\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u} \right) \, dt$$

where Q and R are weight matrices carefully chosen and designed. The Q matrix represents the gain of the cost function on the states of the system, while the R matrix represents the gain of the cost function on the input to the system. Therefore, Q and R define what we consider as optimal.

Task 1 Plot the states of the system (i.e., the elements of the observation vector) over the first 400 timesteps. Remember that the Cart-Pole task is considered solved by Gym when 199 time steps are reached without the pole falling over.

Question 1 Looking at the plot, after which timestep do all states converge to within 0 ± 0.05? Explain what it means for all states to converge to zero from a control standpoint.

Task 2 Vary the value of R within the range [0.01, 0.1, 10, 100] and plot the forces applied to each of the four controlled systems in a single graph over the first 400 timesteps.

> Note: for a better visual comparison, invert the single plot for each value of R if the first force applied on the cartpole is positive (e.g., if you are storing all the forces in an array called `forces`, this means `forces[0]>0`). In this way, all the plots would have the same trend.

Question 1 Analyzing the plot, how does the choice of R affect the force? Are they proportional or inversely proportional?

4 Reinforcement Learning

The provided Python script (`cartpole_rl.py`) 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_rl.py`.

4.1 Training and testing

Run the `cartpole_rl.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.ai` and can be tested by passing `--test <path>/<to>/<model>/<file.ai>`. You can use the `--render_test` argument here.

All the training algorithm is encapsulated within the `train(...)` function. Examine the training loop and answer the following questions:

Question 1.0.1 What are the conditions under which the episode ends? Do all episodes conclude at the same timestep?

Question 1.0.2 How is the reward internally computed?

> Note: You can refer to the environment's documentation for more details or inspect the code to determine when different episodes conclude. Rendering the environment may also aid in understanding episode termination and reward computation.

Task 1.1 Try to train the agent using a random policy. This entails not considering the predicted actions by the agent at each current timestep. Instead, sample a random action from within the action space, and perform this action on the environment.

Question 1.1 Evaluate the learned random policy. What is the performance in terms of the average test reward compared to training with the normal policy? Briefly analyze the underlying causes.

Task 1.2 Train a model with 200 timesteps per episode. Then test the model for 500 timesteps by changing line n.133 in `cartpole_rl.py`. You can test the model with:
`python cartpole_rl.py -t <MODELFILE> --render_test`

> Note: the *episode length* and the *number of episodes* given by the command line argument `train_episodes` are two different things. If you include plots in your submission, use the `train_episodes` as their x-axis.

Question 1.2 Can the same model, trained to balance for 200 timesteps, also balance the pole for 500 timesteps? Briefly justify your answer.

Task 1.3 Repeat the experiment a few times, each time training the model from scratch with 200 timesteps and testing it for 500 timesteps. Evaluate its performance by using the `--render_test` argument and remember the average test rewards.

Question 1.3 Are the behavior and performance of the trained model the same every time? Analyze the causes briefly.

4.2 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.

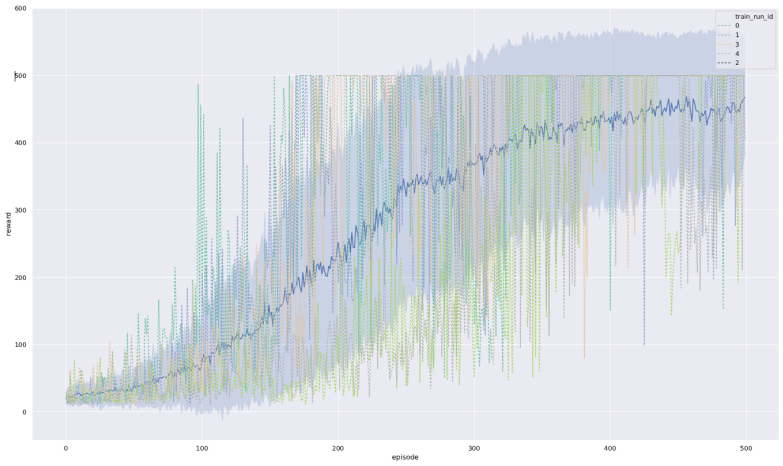

Figure 2: Variance in performance between multiple repetitions of the experiment.

Question 2: Why is this the case? What are the implications of this stochasticity, when it comes to comparing reinforcement learning algorithms to each other? Please explain.

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

4.3 Reward functions

Let us design a custom reward function and use it instead of the default reward returned by the environment (see the `new_reward(...)` function for an example).

Task 3.1 Write different reward functions to incentivise the agent to learn the following behaviors and include them in your report:

1. Balance the pole close to the center of the screen (close to $x = 0$),
2. 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. Keep the cart moving from the leftmost to rightmost side of the track as fast as possible, while still balancing the pole. The minimum speed of the agent should be high enough that the agent is

visibly moving from one side of the other (not just jittering in the center) and changes direction at least once and not just drifting in a random direction.

Train a model with each reward function and include them in your submission.

Hint: 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).

Task 3.2 Let us visualize the third reward function. Train the model at least 10 times using `multiple_cartpoles_rl.py` (adjust the `--num_runs` and `--train_episodes` parameters) and include the performance plots similar to Fig. 2.

Question 3.2 Observe the agent using `python cartpole_rl.py -t <MODELFILE> --render_test` and keep track of the observations. What do you observe? What is the highest velocity the agent reaches?

5 Extras

Question 1 Discuss briefly the advantages of using an LQR controller or an RL agent to solve a general task of controlling a robotic system. Analyze the differences and similarities between the two methods.