Report Laboratory 02 - Robot Learning

Federico Pulinas
s319881@studenti.polito.it

Abstract: The purpose of this laboratory is to provide a first look at the Cart-Pole environment while implementing diffent control strategies including Linear Quadratic Regulator (LQR), random policies and fundamental concepts of reinforcement learning.

Keywords: Cart-Pole, Reinforcement Learning, LQR

1 LQR - Linear Quadratic Regulator

In this part we are going to discuss the results obtained by testing a baseline using LQR, resulting in an optimal control gain K. The controller K is optimized by minimizing the following cost function:

$$J = \int_0^\infty (x^T Q x + u^T R u)\, dt$$

where Q and R are weight matrices with values:

$$Q = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}, R = 1$$

The matrix R represents the gain of the gost function on the input of the system, while Q represents the gain of the cost function on the states of the system.

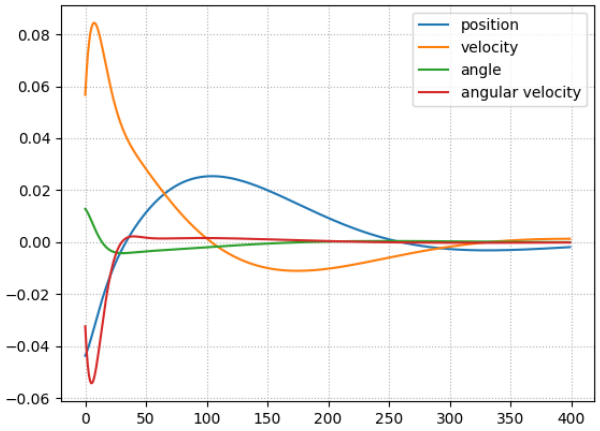


Figure 1: plot of the states of the system over the first 400 timesteps

The goal of this subsection is for the system to stabilize around the point of coordinates x = 0.0, meaning that the cart position should be in the neighborhood of that point while keeping the pole balanced. Studying the plot we can see that after only 50 timesteps the system reaches its goal with an accuracy of ± 0.05 and after 300 timesteps it converges to 0.

Considering Q fixed, an interesting experiment is to vary the value of R, compare the results and draw conclusions.

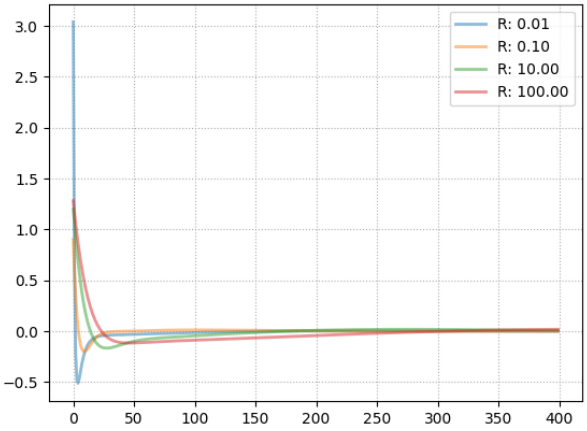


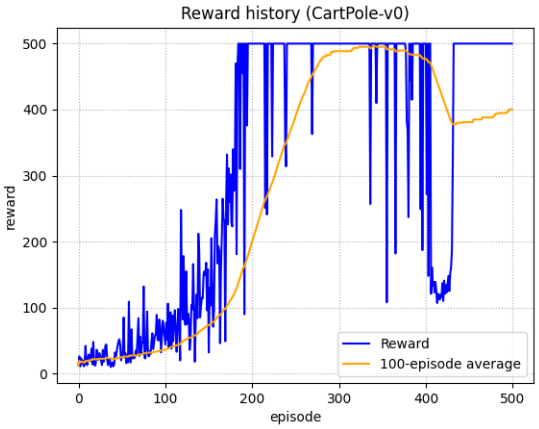
Figure 2: plot of the normalized force of the system referred to each variant of R

We now study the force applied to each of the four controlled system, noticing that all the plots have a common trend, with the only difference in the dimension of the spikes. Those spikes are referred to the force applied to correct the system in a specific moment, analyzing them it appears that the bigger the R the smaller the spike will result, thus having a smoother motion of the cart. There is no direct proportionality between the matrices Q and R, the key point is to choose them in order to have a trade-off between the state regulation and control effort.

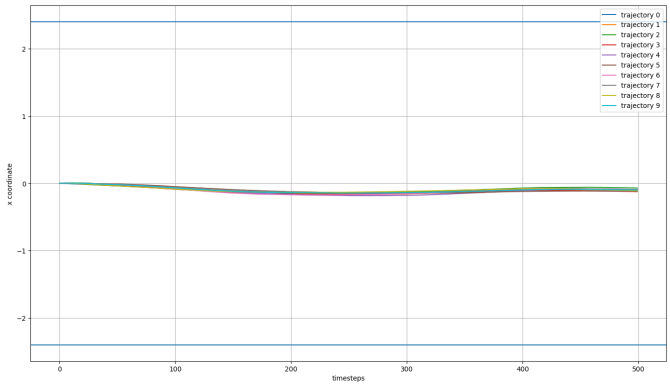
2 Reinforcement Learning

2.1 Training and Testing

Changing the approach on the system and passing to a Reinforcement Learning paradigm, we first experience a training on 500 timesteps with the default values, namely the desired point is positioned in x = 0.0 and the reward is computed adding +1 for every step taken, including the termination step.



(a) Trend of the total reward of each episodes



(b) Trajectory of the system averaged every 100 episodes

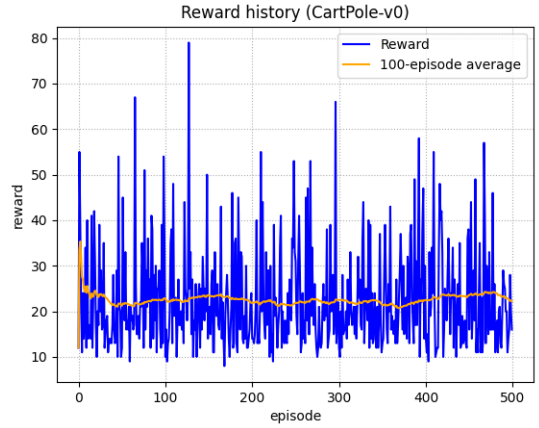
Figure 3

As for the episodes there are different cases for which it may finish:

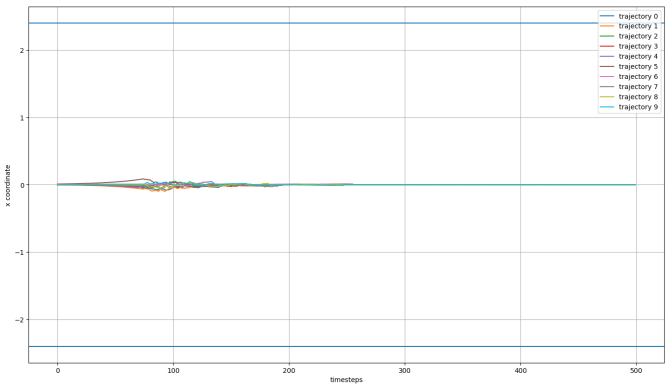
- It reaches the maximum timesteps available and it is terminated (best case);
- The cart x-position assumes a value outside the admitted range (-2.4, 2.4);
- The pole angle assumes a value outside the feasible range (± 12°)

By looking at the plots of the trajectories, created accordingly to the test function, we can see that the system has successfully learned its task.

A different scenario is produced when the system is trained using a random policy: at each timestep a random action is sampled from within the action space and performed on the environment.



(a) Trend of the total reward of each episodes

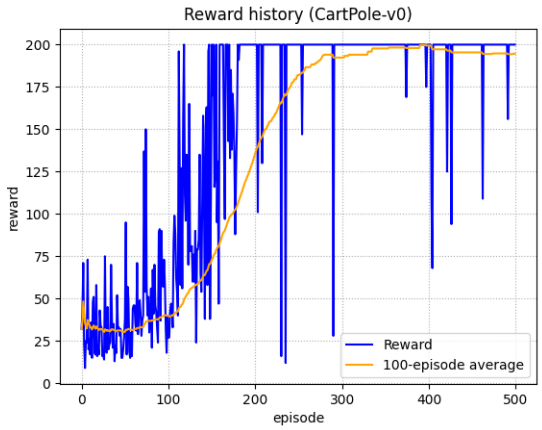


(b) Trajectory of the system averaged every 100 episodes

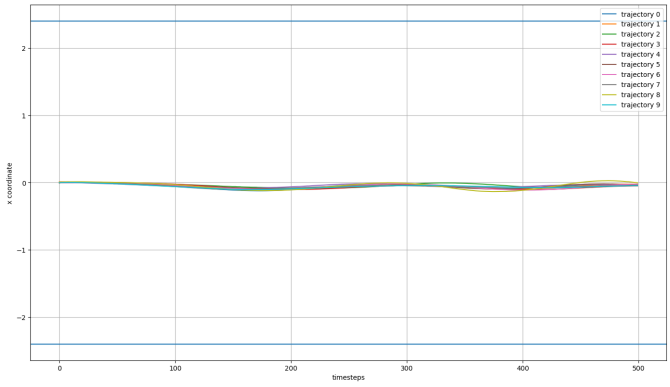
Figure 4

As expected the system does not learn because of how the policy is implemented, resulting in a noisy plot in the tested trajectories. Notice that the plot becomes completely flat after a certain point, meaning that the system didn't produce anymore output, thus it terminated before 500 timesteps.

Trying to play around with the parameters it is interesting to vary the timesteps per episode between the training and the testing. One possible case can be when the training phase has 200 timesteps available and the test in kept at 500, producing the following results.



(a) Trend of the total reward of each episodes



(b) Trajectory of the system averaged every 100 episodes

Figure 5

It's easy to notice that the performance of the test is comparable with the standard case, with some small differences in the accuracy of the movement and the coordination between the different set of episodes.[1]
By repeating the experiment (training + testing) a few time we can conclude that those settings produce feasible results with an average reward of 200 over 200 timesteps.

2.2 Repeatability

In this task it's required to run 100 independent training procedures. The output is a graph reporting the mean and standard deviation in performance between multiple repetitions.

[1] referred to the fact that each trajectory is obtained averaging over the trajectories produced by 100 episodes

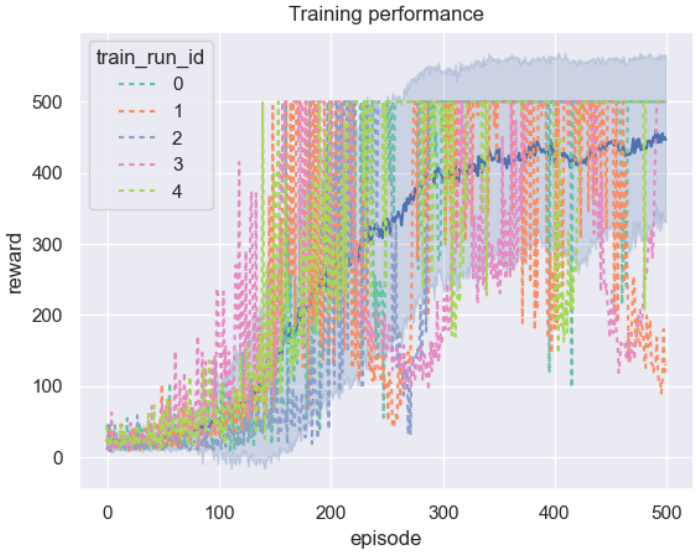


Figure 6

The high value of the variance may be given by the ”exploration-exploitation”, a trade-off between trying new actions and focusing on actions with known good outcomes, and the intrinsic randomness of the cartpole environment.

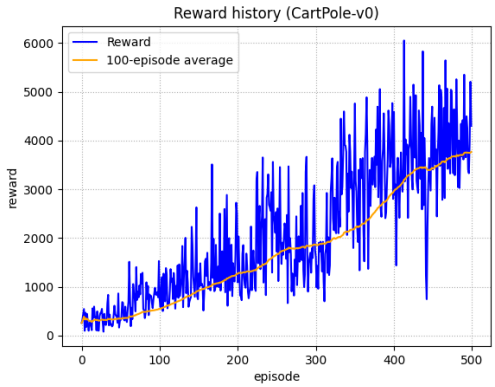
2.3 Study of Reward Functions

This part of the assignment required to write different reward functions to incentivise the agent to learn some specific behaviours.

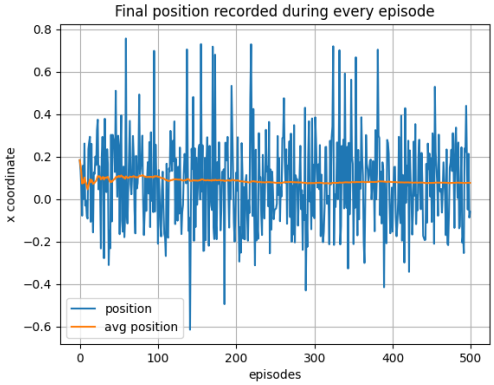
1. Balancing the pole close to an arbitrary point of the screen passed as a command line argument to the script. A particular case of this task is when the goal point is set at $x = 0$ For this scenario I have chosen to implement a linear reward function of the form

$$reward = -2x + 4.8 \tag{1}$$

where x is the euclidean distance between the position of the cart in that timestep and the goal point. The offset of 4.8 is given by the range of feasible distance that the cart can travel without the episode terminating. An addition $+20$ was added to the reward if it fell within a small range around the desidered location
This kind of function has its highest value when closer to 0 and lowest value (i.e. 0) at 2.4.

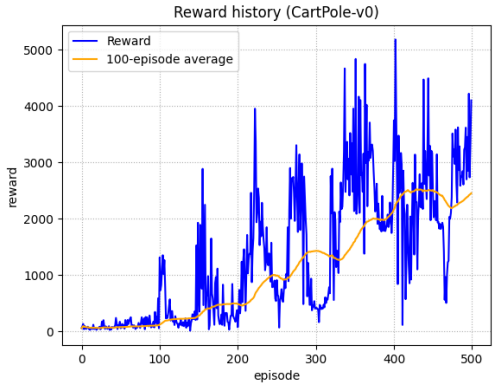


(a) Trend of the total reward of each episodes

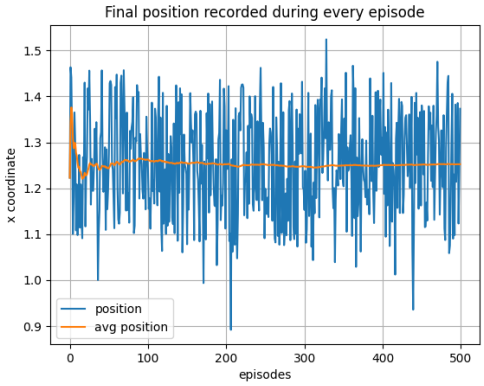


(b) Record of all the final position recorded of each episode

Figure 7: $x = 0$ pole centered close to the center of the screen

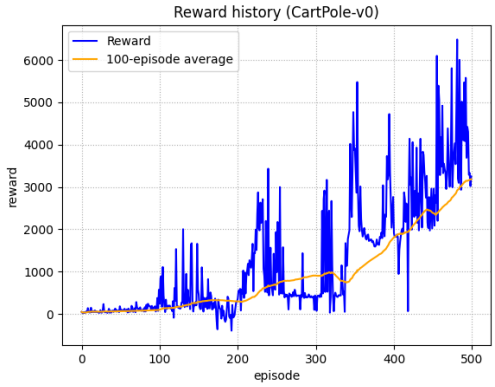


(a) Trend of the total reward of each episodes

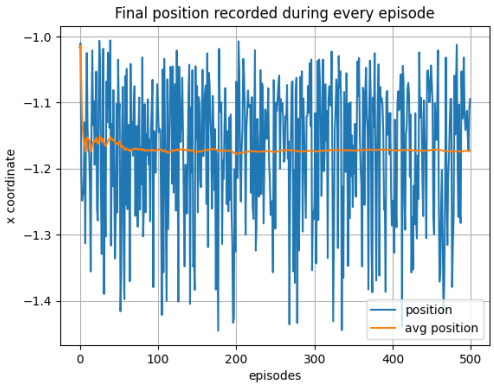


(b) Record of all the final position recorded of each episode

Figure 8: $x = 1.2$ pole center to the left side of the screen



(a) Trend of the total reward of each episodes



(b) Record of all the final position recorded of each episode

Figure 9: $x = -1.2$ pole centered to the right side of the screen

With this reward function I achieved an accuracy of ± 0.05 in the best case and ± 0.10 in the worst case.

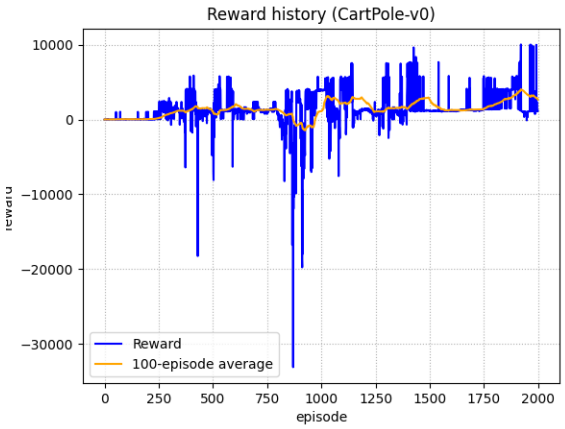
2. Keep the cart moving from the leftmost to rightmost side of the track as fast as possible, while still balancing the pole. In this case I chose to split the path from $-x_0$ to x_0, considering the points $\frac{-x_0}{2}$, $\frac{x_0}{2}$ and 0 as checkpoints.
   At the beginning the function check in which direction the system started moving and sets the desired checkpoint, in order to not restrict the system to prefer one specific direction to start with. After the first step two scenarios may occur:

   - the system changes the direction of the path, instead of going from $\pm\frac{x_0}{2}$ to $\pm x_0$ it goes to the opposite point. The algorithm allows only one change of this type per episode. Note that this situation may happen only at the beginning of the motion.
   - the system continues on the current path.

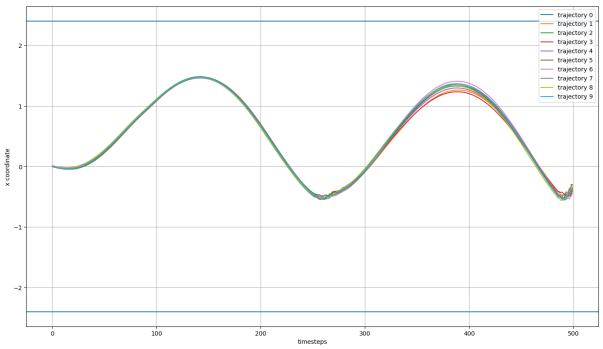
   The reward function is linear wrt to the euclidean distance between the current position of the cart and the current checkpoint, with a variable slope depending of the checkpoint:

   - if it is a middle point the slope is such that the maximum distance allowed before the function touches the 0 is around 1.5
   - if the checkpoint if moving from the center to one of the two sides the slope is lower allowing wider movements.
   - if it is in one of the extremes the slop is set high enough for the system to be directed towards the new checkpoint and learn the boundaries of the environment

   In general for each checkpoint it is provided an additional bonus for the reward and, even if in a small portion, the velocity of the system is considered after an appropriate manipulation.



(a) Trend of the total reward of each episodes



(b) Record of all the final position recorded of each episode

Figure 10

After repeating the experiments many times I came to the conclusion that the method adopted was not the optimal one and it required quite some luck to achieve an appropriate behaviour of the system. As for the plots it successfully moved around the screen without actually managing to touch the borders.

2.3.1 Study of Repeatability on the third reward function

In this case we test the repeatability property of the system implementing the third reward function, i.e. the system moves back and forth on the screen.
We train 30 independent system using the script `multiple_cartpoles.py`, considering 1000 episodes for each system.

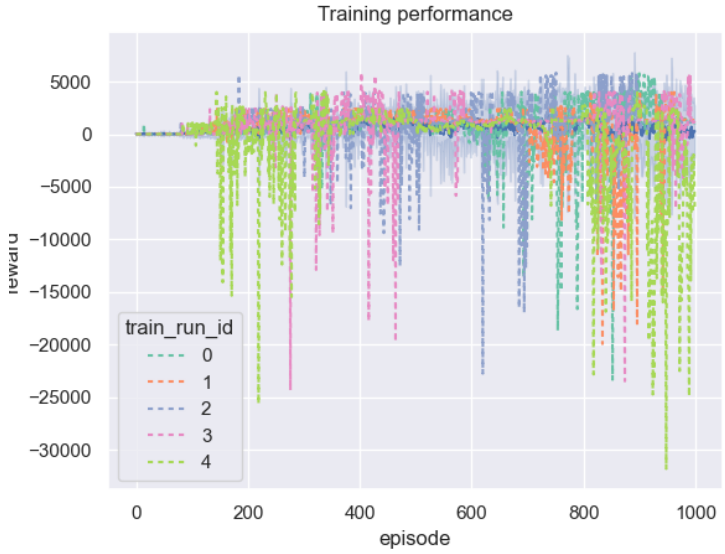


Figure 11: Multiple_cartpoles training on 30 systems of 1000 episodes each

For notation we call ”multiple_cartpoles” the trained model using `multiple_cartpoles.py` and ”cartpole” the trained model using `cartpole.py`, both cases set to 1000 episodes.
By studying the rendering of the test produces by the multiple_cartpoles I noticed that the performances are lower compared to the cartpole. This is given by the fact that many of the episodes produced a movement for which the cart was always close to 0 and few times moving around the screen. Meanwhile the cartpole rendering produces a larger variety of movements and never stopped in 0. As for the velocity, the highest values reached from both the cases is around 1. In general both the trained models have a randomic component, give by the reward function that requires more specific command to understand the assignment, thus producing mostly unsatisfying behaviours.