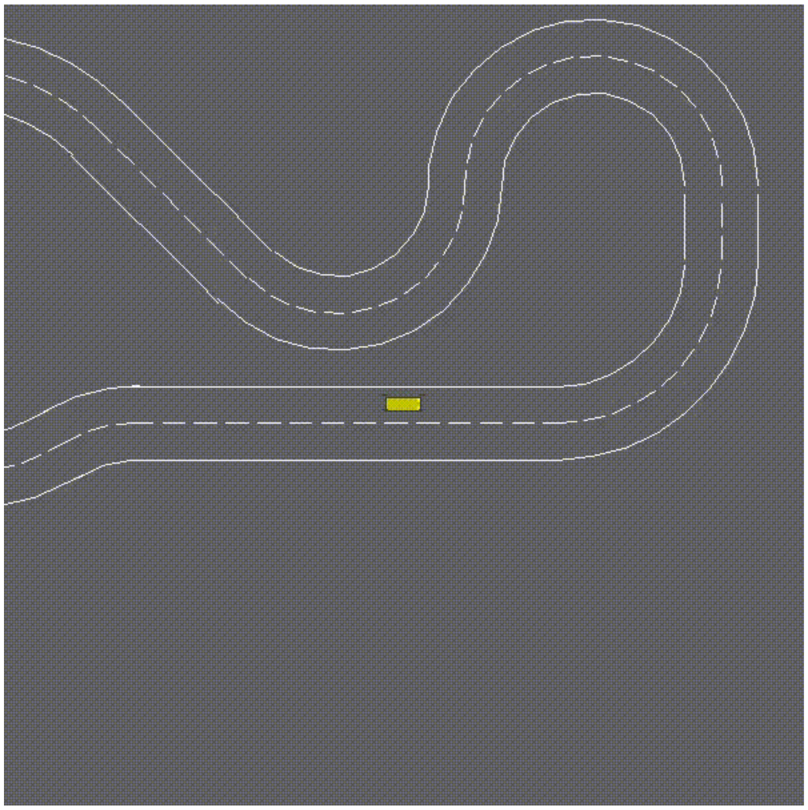
CS 747: Programming Assignment 3

Optimal Driving Control

Total marks: 12

(Prepared by Sandarbh Yadav)



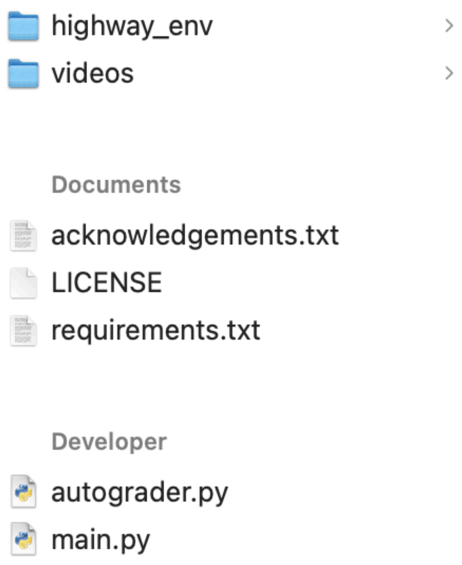
In this assignment, you will be designing an agent to drive a car across various tracks at high speeds. The car gets local information about the track from the environment and has two controls: accelerator and steering. Your task is to come up with a policy that maps information from the environment to the car controls. The maximum duration of each episode is fixed; the episode can also terminate when the car goes off road. The objective of your agent is to cover as much distance as possible within each episode. Your policy should generalise to different tracks, based on which you will be evaluated. You are given a simulator (thanks to The Farama Foundation) that simulates different driving environments for you.

Installation

You should use Python 3.12.0 for this assignment. Your code will be tested using a python virtual environment containing the libraries listed in `requirements.txt` of the compressed directory linked below. You are not allowed to use libraries other than the ones listed in `requirements.txt`. Your submission will receive zero marks if it requires installation of additional libraries. For more details, see the section below on "Getting started".

Code Structure

This compressed directory consists of the entire code required for this assignment. The structure within the code directory is shown below:


highway_env
videos

Documents
acknowledgements.txt
LICENSE
requirements.txt

Developer
autograder.py
main.py

You only have to modify `main.py` file for this assignment. Grading of your agent is done by `autograder.py` file. The environment used for this assignment is contained in `racetrack_env.py` present in `envs` directory of `highway_env` directory. The `videos` directory stores recorded videos of your agent evaluations. The only config variable you are allowed to modify through the `main.py` file is `config["track"]` which represents the index of the six tracks (0 to 5). Your submission will receive zero marks if you change any other config variables through `main.py`. You are allowed to construct and add your own tracks in `racetrack_env.py` if you think learning on your tracks will lead to a better agent. However, you will be evaluated only on the tracks we have provided (visible) and which we have kept for evaluation (unseen).

Getting Started

To get started with the code, follow these steps.

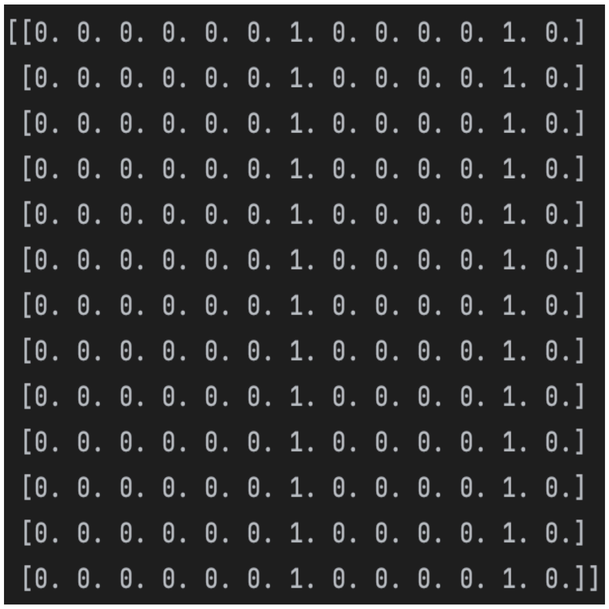
1. Download the compressed directory linked above, and decompress it. It will yield `CS747PA3` directory.
2. Enter `CS747PA3` directory and you will see a structure as in the image above.
3. Create a new python virtual environment and install the libraries listed in `requirements.txt`.
4. Run the following command: `python main.py --render`

A default agent is provided in the `policy()` function of `main.py` file whose acceleration and steering are always 0. The agent drives straight at a constant speed of 5 units. A pop-up window will display how this default agent performs after running the above command. Additionally, you can access the recorded videos in the `videos` directory.

Environment Details

The simulator provides you one of the chosen six tracks (indexed from 0 to 5) for each episode. You can select a particular track by changing `env.unwrapped.config["track"]` to the index of your desired track. The desired index of the track should be configured before calling `env.reset()`. Each track has two lanes and the car will be initialised on one of them. The car is allowed to switch lanes and is initialised with a speed of 5 units. The maximum achievable speed is 20 units. The distance covered by the car is measured in counter-clockwise direction along the road. The environment returns the following information.

- `obs` describes the local front view of the car and contains a 13*13 binary matrix with ones representing centres of the lanes. This binary matrix is aligned with the axes of the car, which is placed at the middle of the last row of the binary matrix i.e. at index [12, 6] of the binary matrix. You can access the binary matrix by `obs[0]`. A sample binary matrix is shown below. The car is located at index [12, 6]. Here, it is initialised on the left lane and there is another lane to the right of it. Note that the ones represent centres of the lanes, not road boundaries.



```
[[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]]
```

- `trunc` is a boolean variable indicating whether the maximum duration (set as 350 time steps) of the current episode has been reached.

- `term` is another boolean variable indicating whether the current episode has terminated. An episode is terminated when the car goes off road.

- `info` is a dictionary containing information about the current speed of the car (`info["speed"]`), total distance covered by the car (`info["distance_covered"]`) and whether the car is on road (`info["on_road"]`).

Note that unlike typical gymnasium environments, our environment does not provide you a reward signal at each time step. You are given the creative freedom to design your own reward signal based on the information provided to you by the environment.

Control Details

The agent acts in the environment through two controls: [*acceleration*, *steering*]. The first control represents the acceleration of the car while the second control represents the steering angle of the car. Both these controls take real values in the range [-1, 1]. Internally, the environment maps the two controls as follows.

- *acceleration* is mapped from [-1, 1] to [-5 units/s^2, 5 units/s^2]
- *steering* is mapped from [-1, 1] to [-π/4, π/4]

Creating your Agent

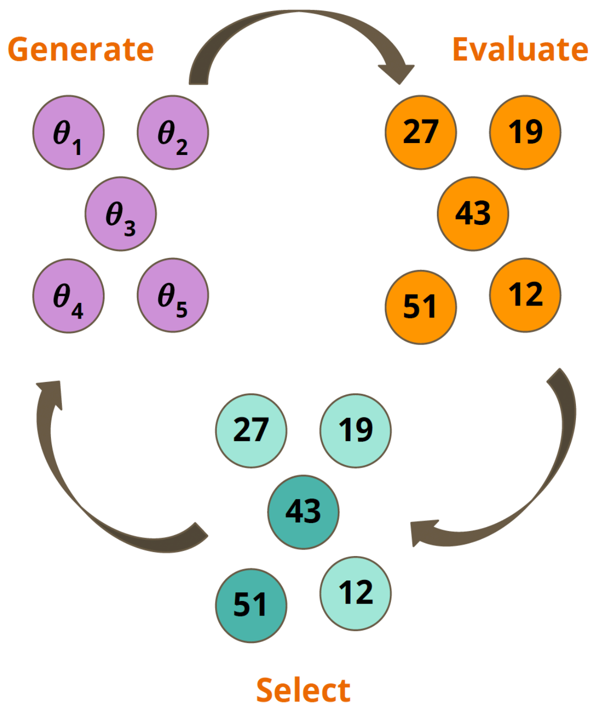
You have to code the policy of your agent in the `policy()` function of `main.py` file. The `policy()` function contains a default policy with controls [0, 0]. You have to replace the default policy by your policy. The `policy()` function has access to information provided by the environment (state, info) and returns a list with two elements corresponding to the two controls of your car: [*acceleration*, *steering*]. Your policy might contain some parameters which have to be optimised.

You are urged to first understand all the relevant aspects of the environment and the code given above, in conjunction with making small changes to `policy()` function to test out any ideas. Thereafter, spend some time coding up your own agent policy. It is recommended that you craft good features based on the information provided by the environment and use them to formulate your policy. We expect that in the span of two weeks, the most successful agents will be based on a combination of hand-coding and parameter-learning.

In the course, we cover various methods for reinforcement learning; you have a free hand to choose any of these and try to apply them to learn better policies. However, to make your job easier, we provide support for a particular method for doing **policy search** (covered in the Week 11 lectures, and further discussed in the review for that week). Please refer to the section below on CMA-ES for details on optimising policy parameters.

CMA-ES

Covariance matrix adaptation evolution strategy (CMA-ES) is a derivative-free black-box optimisation technique inspired by biological evolution. Suppose θ represents your parameter vector. There could be multiple instances of θ based on the values of the parameters. Each such instance is mapped to a fitness value *f*(θ) using a fitness function *f*. The fitness function *f* represents some notion of the goodness of θ. CMA-ES initialises a population of `pop_size` candidate parameter vectors and evolves them across `num_gen` generations to find an optimising θ. In each generation, CMA-ES evaluates the population of candidate θ using the fitness function *f*. Then, based on the fitness values CMA-ES greedily selects few top candidate θ and uses them to generate the population of next generation. The guiding principle behind CMA-ES is "survival of the fittest". Across generations, parameter vectors with high fitness survive and those with low fitness get eliminated. An illustration is shown below. You can read more about CMA-ES here.



We have provided the skeleton code of CMA-ES in `main.py`. If you decide to use it, your main task is to come up with suitable parameters to optimise (up to a few tens or even hundreds is usually okay), and importantly, an appropriate fitness function. It is recommended that you formulate the fitness function keeping generalisation in mind. Note that the `cma` library is built for minimising by default. In order to maximise your fitness function, put a minus sign in front of your fitness values. Additionally, you will have to specify `num_gen`, `pop_size`, and `num_policy_params` for using CMA-ES.

Running the Code

The commands used for running the code are different for students who use CMA-ES and those who don't.

Students who rely only on hand-coded strategies, without using CMA-ES to learn policy parameters, should use the following commands:

`python main.py --numTracks 3 --seed 619` (to evaluate your policy on the first 3 tracks using seed 619; numTracks should take integer values from 1 to 6)

`python main.py --numTracks 5` (to evaluate your policy on the first 5 tracks using default seed 2025)

`python main.py --seed 333` (to evaluate your policy on the 6 public tracks using seed 333) `python main.py` (to evaluate your policy on the 6 public tracks using default seed 2025)

You can use `--render` alongside any of the previous 4 commands to render your policy evaluations and record videos. Note that each command executed with `--render` will replace your older videos.

`python autograder.py` (to grade your policy on the 6 public tracks; autograder evaluates each track on 2 different seeds)

Students who use CMA-ES to learn policy parameters, should use the following commands:

`python main.py --train` (to train the policy parameters using CMA-ES; the learned parameters of your policy get stored in `cmaes_params.json` file)

`python main.py --eval --numTracks 3 --seed 619` (to evaluate your policy on the first 3 tracks using seed 619; numTracks should take integer values from 1 to 6)

`python main.py --eval --numTracks 5` (to evaluate your policy on the first 5 tracks using default seed 2025)

`python main.py --eval --seed 333` (to evaluate your policy on the 6 public tracks using seed 333)

`python main.py --eval` (to evaluate your policy on the 6 public tracks using default seed 2025)

You can use `--render` alongside any of the previous 4 commands to render your policy evaluations and record videos. Note that each command executed with `--render` will replace your older videos. The `--eval` argument ensures that your policy parameters are read from the `cmaes_params.json` file.

`python autograder.py --cmaes` (to grade your policy on the 6 public tracks; autograder evaluates each track on 2 different seeds)

The `--cmaes` argument ensures that your policy parameters are read from the `cmaes_params.json` file while running autograder.

Evaluation

4 marks are reserved for the public tracks. 4 more marks are reserved for the hidden tracks, which will be similar to the public ones. The final 4 marks are reserved for your report. 4 + 4 + 4 = 12.

There are 6 public tracks and an undisclosed number of private tracks. Both the public and private tracks will be evaluated on two different seeds. We will evaluate your agent against the performance of two (unseen) baseline agents. For each track and seed combination, you will receive half mark for clearing the first baseline and another half for clearing the second baseline. You can see the performance details of the two baselines in `autograder.py`

Important: It is your responsibility to ensure that the entire autograder procedure (6 public tracks, 2 seeds each) does not take more than 6 minutes. This is a strict requirement and submissions which fail to do so will be penalised accordingly. Also note that we won't be installing additional libraries for evaluating your submission. If you have used CMA-ES, we won't be training your parameterised policies. It is your responsibility to ensure that the learned parameters of your policy are stored in `cmaes_params.json` file. We have already provided the code for storing your policy parameters.

Report: Unlike the previous assignments, you have been given an open field to design and optimise your solution in this assignment. Your report needs to communicate how you navigated your way to a solution. Your report should elucidate the ingredients of your solution in detail. You should describe the features used, how you mapped your features to the two controls, any intermediate experiments that may have guided your decisions, and so on. If you have optimised your policy parameters using CMA-ES, you should specify `num_gen`, `pop_size` & `num_policy_params` and describe the fitness function, learning curve, etc. If your report is not sufficiently clear and informative, you will stand to lose marks.

Unlike the previous assignments, you have been given a free hand to come up with your agent. Hence, we would like to see a clear presentation of your approach. The TAs and instructor may look at your source code and notes to corroborate the results obtained by your program, and may also call you to a face-to-face session to explain your code.

Submission

You have to submit one tar.gz file with the name (roll_number).tar.gz. Upon extracting, it must produce a folder with your roll number as its name. It must contain a `report.pdf` - the report as explained above, and one code file: `main.py`. If you have used CMA-ES to optimise your policy parameters, you must provide the `cmaes_params.json` file containing parameters of your policy. You must also include a `references.txt` file if you have referred to any resources while working on this assignment (see the section on Academic Honesty on the course web page).

Deadline and Rules

Your submission is due by 11.59 p.m., Sunday, April 13, 2025. Finish working on your submission well in advance, keeping enough time to generate your data, compile your report, and upload to Moodle.

Your submission will not be evaluated (and will receive zero marks) if it is not uploaded to Moodle by the deadline. Do not send your code to the instructor or TAs through any other channel. Requests to evaluate late submissions will not be entertained.

Your submission will receive a score of zero if your code does not execute on the specified Python and libraries version. To make sure you have uploaded the right version, download it and check after submitting (but before the deadline, so you can handle any contingencies before the deadline lapses).

You are expected to comply with the rules laid out in the "Academic Honesty" section on the course web page, failing which you are liable to be reported for academic malpractice.

Have fun!!