Performance Evaluation project: Optimizing cars' trajectory with AI

Ottavy Macéo, Longatte Mathieu, Mocq Louison

Abstract

The goal of this project is divided into five parts:

- Creating a racing car environment to simulate a simple 2D racing car model.
- Implementing Deep Q-Learning and Genetic Algorithms to optimize the behavior of a car on tracks, enabling it to follow the best possible trajectories.
- Evaluating the performance of Deep Q-Learning and Genetic Algorithms and comparing their results.
- Assessing the performance of Deep Q-Learning with respect to different hyperparameters.
- As a bonus: evaluating the performance of the best car behavior achieved by both algorithms.

Introduction

motivation

Modeling

0.1 Racing environment

0.1.1 Tracks

A track is originally a .png file wich look like the left image of figure 0.1.1. Then, the image is converted to a matrix T such that $T[0][0]$ is the bottom left corner. After that, we crop the image, compute the starting point and the lines of track (that will be explained in the reward part) to have a final result which look the right image of figure 0.1.1. The white case represent the road, the green point represent the starting point.

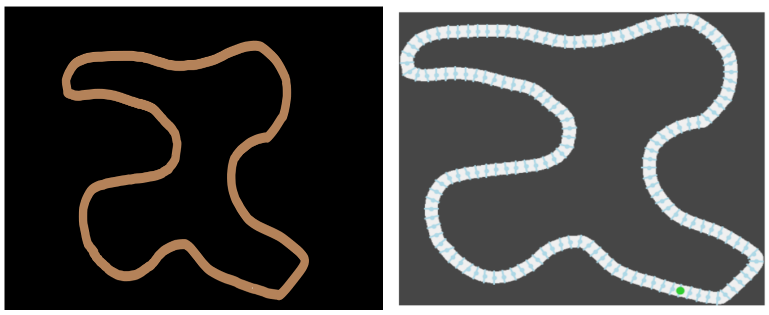


Figure 1: .png and computed track

0.1.2 Cars' physics

The Car physic is really simple. It is a 2D cartoon-like physics that act as follow:
The car has two main informations: its speed $\in$ [0, MaxSpeed] and its rotation $\in [0, 360]$. The physics

acts as follow: at each times step the car move to next coordinates on the direction of the car's rotation and of distance equal to the car's speed.
If the coordinates of the car is (x, y), its speed is s and its rotation is α, then, after a time step, the coordinate of the car will be:

$$(s.cos(\frac{\pi}{180}\alpha) + x,\ s.sin(\frac{\pi}{180}\alpha) + y)$$

Moreover, at each time step, the car can make some actions:

- It can accelerate, this will increase the car's speed by a constant.
- It can brake, this will decrease the car's speed by reduce the car speed by a constant. The car cannot have a negative speed.
- It can turn, i.e. add a constant $\in [\![-K, K]\!]$ to its rotation. K is a constant that is the maximum angle the car can turn per each time step.

The behaviour of the car will need to interact with the track therefore we need to decide what is the state of a car, i.e. how the car see the environment. We could give to our algorithms the track matrices and the informations of the car but this will leed to to many parameters because a track can have size 900×600. Therefore we will need to train on all possible state wich will be at least $2^{900\times600}$.
Therefore, we decided to give a more realistic state wich represent how a car racer see. Then the state of a car is a array T of size 8.

- T_0 is the current speed of the car
- $\forall i \in \{1, ..., 7\}$, T_i is the distance of the car to the next wall in the direction $\alpha + A_{i-1}$ where α is the current rotation of the car and $A = [60, 40, 20, 0, -20, -40, -60]$

Then, the representation looks like figure 0.1.2.

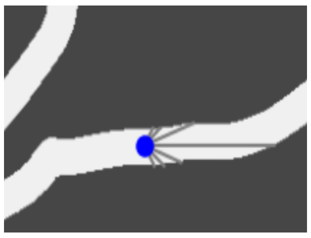


Figure 2: Car state

0.1.3 Technical aspects of the environment

To manipulate our environment, we use the python packages `gymnasium` which provide code convention for those type or environment, i.e. environment where at each time step, you have one action to do. The environment has to have some essential function: `reset()` that reset the environment to be able to do an other simulation, `render()` that render the current state of our environment and the most important one is `step()` that do one step of time, i.e. given an action, the `step()` function figure out is the car has crashed or not, move the car to its next position and return the new state of the car, a reward and if the car has crashed.
Our environment has a variable named `time` which give us the opportunities to discreet more or less the time.

0.1.4 Rewards

For those type of problem where the AI model has to compute a behavior, the AI model produces something which look like a function f that take a car state and return an action. We need to specifies to our AI model when it produce a good action and a bad action, for instance, if a car crash, we need to punish the AI model.
We do that thanks to a function reward implemented in the function `step()` of our environment.

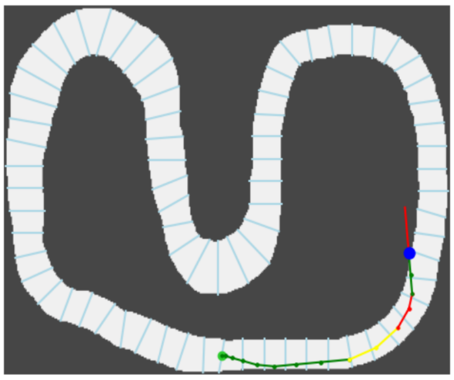
The reward is an integer, the bigger it is the best the action was. The function might be the most important one of all the project because it is thanks to it that our AI model will perform well or not. We try lot of reward function and we finish by using the following one. To punished the car when it do something bad we do:

- If the car crashes, we stop the simulation and return a reward of -500
- If the car is not moving, i.e. has a speed of 0, the reward is -10

For the positive reward, we have automatically computed some track line (represented in right image of figure 0.1.1). If the car crosses next line, it has a reward of ($+10\times$ the number of lines it has cross in the good order with this action). If the car cross a line in the wrong order, it means that it has gone backward, therefore, we punished the car with a reward of -200 and we stop the computation. On top of that, at each time step, we add to the current reward the speed of the car divided by a constant to encourage the car to go fast.

0.1.5 Conclusion

After explaining all of this, here is an example in figure 0.1.5. We have plot the trajectory of the car, the green color is when the car has accelerate, red color when it brake and yellow otherwise.



List of rounded reward: `[10, 1, 2, 12, 13, 13, 13, 14, 24, 14, 13, 12, 13, -497]`

Figure 3: Car state

The total reward of a car behaviour is the sum of all reward of a simulation with a car behaviour. For example, the reward of the car behaviour or the figure 0.1.5 is -343.

0.2 Deep Q-learning

Deep Q-Learning is a reinforcement learning algorithm that combines Q-Learning with Deep Learning to solve complex decision-making problems. It allows an agent to learn how to act optimally in environments with large state spaces by approximating a function, known as the *Q-function*, which evaluates the quality of an action taken in a given state.

0.2.1 Q-function

The Q-function, $Q(s, a)$, represents the expected cumulative reward an agent will receive after taking action a in state s, and then following the optimal policy. The cumulative reward is computed as:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a'),$$

Where:

- r is the immediate reward received after taking action a in state s.
- s' is the next state reached.

- a' is the next action.
- $\gamma \in [0, 1]$ is the discount factor, which balances immediate and future rewards.

0.2.2 Key Techniques

- **Replay Buffer**: A memory that stores past experiences (s, a, r, s'). Randomly sampling experiences from the buffer during training reduces correlations between consecutive samples, improving learning stability.
- **Exploration-Exploitation Balance**: The agent uses an ϵ-greedy policy to choose actions, where it explores randomly with probability ϵ and exploits the best-known action otherwise.

0.2.3 High-Level Workflow

1. Observe the current state s.
2. Choose an action a using an ϵ-greedy policy.
3. Execute the action, observe the reward r and next state s'.
4. Store the experience (s, a, r, s') in the replay buffer.
5. Sample a mini-batch of experiences from the buffer to train the Q-network.

0.3 Genetic algorithms

0.3.1 What are genetic algorithms?

Genetic algorithms (GA) are probabilistic algorithms based on natural selection. Therefore, GA takes some populations which are sets of solutions (here a solution is a car's behaviour), select the best solutions thanks to the reward function. Then, it changes the population by adding new random solutions, adding some mutations which are some small variations of a behaviour, adding some cross-over which are the equivalent of natural reproduction. We can either repeat this process a fixed number of generations or for a fixed amount of time.

0.3.2 Markov Chain modelisation

We will now introduce a Markov chain modelisation to genetic algorithm. We define a Markov chain $(Y_n)_{n\in\mathbb{N}}$ as following:

- A state of $(Y_n)_{n\in\mathbb{N}}$ is a population.
- Let y_0 be a special state such that if $Y_n = y_0$ then it means that the population of state Y_n contain an optimal solution.

Now, the sequence of population of genetic algorithm can be describe with this Markov chains. Y_n represent the population at generation n. Notice that the state y_0 is an absorbing state. In fact if $Y_n = y_0$ then it mean that the population P_n contain an optimal solution. Since we always keep the best solution of the previous population, it means that $\forall n' > n$, we have that $P_{n'}$ contain an optimal solution. Therefore, $\forall n' > n$, $Y_{n'} = y_0$. Moreover, y_0 is the only absorbing state of $(Y_n)_{n\in\mathbb{N}}$.

If we suppose that our mutation and cross-over are made such that a solution x can reach y by a series of a finite number of those operations. Then, all solutions x can reach an optimal value. Then every state y_n can reach state y_0. The set of all possible state is finite. Then $\mathbb{P}(Y_n = y_0) \underset{n\to+\infty}{\to} 1$
Then $\mathbb{P}($ The population P_n contain an optimal solution$) \underset{n\to+\infty}{\to} 1$
Thus, the genetic algorithm converge toward a global optimal solution. However, we do not know how many time it will take in average.

0.3.3 NEAT

Basic genetic algorithms are not efficient enough to compute an optimize behavior. Therefore, we will use the famous python packages called `NEAT`. It is an optimized generalized genetic algorithms with represent solution as dynamic neural network. By dynamic we mean that the algorithm can add or delete some of the nodes of the neural network. The principle of GA stay the same but we have a lot more hyper parameters.

Simulation

Once the models are selected (in this case, Deep Q-Learning and Genetic Algorithms), the next step is to compare them using various metrics. For this purpose, we choose the following metric: the average reward after training.

We measure this value across different training durations and varying numbers of tracks used for training the models. To ensure robustness and evaluate potential overfitting, we test the models on tracks that were not included in the training set.
This approach is applied in the context of an AI project where we train AI agents to complete a racing circuit. The goal is for the cars to complete laps as quickly as possible, and the average reward reflects their performance under these conditions.

0.4 GA VS Q

metrics: GA VS Q: training time : 10, 40, 60, number of tracks: 10, 40, 0.8 times number of tracks

0.5 Q hyper parameter

0.6 Best car

Experimental

Result of simulation, interpretation

0.7 GA VS Q

0.8 Q hyper parameter

0.9 Best car

Conclusion