Reinforcement Learning for Highway Driving

Marcelo García Escalante
mrgaresc@stanford.edu

Saba Bokredenghel
sababo@stanford.edu

May 18, 2023

1 Introduction

Autonomous driving is a rapidly growing field of research. The goal of this project is to develop a Reinforcement Learning agent that can drive a car on a highway. The agent will be trained in a simulated environment and will be evaluated on its ability to drive safely (without collisions) and efficiently (at the highest speed allowed). The environment consists of a highway with multiple lanes, where the agent will be evaluated on its ability to drive safely and efficiently.

The environment that we will use is built on top of OpenAI Gym.[1] The environment is called "highway-v0"[2] We will use this environment to train and evaluate the agent using Reinforcement Learning algorithms we have learned in class such as *Value Iteration* or *Q-learning*, compare their outcomes and discuss the results.

2 Literature Review

Our goal is to follow the implementation of Value Iteration and Deep Q-learning in[2] to solve the RL task stated in section 1. We might consider also other variants of Deep Q-Learning, like Double DQN, which is also stated in[2].

Other interesting works are included in [3] and [4], where DQN techniques become also the choice of approaches to these problems. In these research articles, the question of safe decision making is discussed. There are two main difficulties: Collisions should never happen and the algorithms need to take into account new observable states (e.g. unpredictable behavior of other agents such as cars) in the environment.

In [3] a Deep Reinforcement Learning agent is designed by training an ego vehicle, which learns a driving policy by interaction with diverse simulated traffic. The work in this article is based mainly on a modified version of the double Q-network (DDQN) algorithm.

Moreover, there were three ingredients, which were essential for the stability and efficiency of the agent: *Short-horizon safety check*, *two buffers* (a simpler version of prioritized experience replay), and a *safety controller*. The short-horizon safety check is necessary because Q-learning relies on an exploration policy (e. g. ϵ-greedy). This would lead to scenarios like collisions and thus to simulation resets. By implementing a short-horizon safety check that evaluates chosen actions and provides an alternative safe action, this can be avoided. The safety controller was important during the inference phase and played a key role in finding a good policy.

We will also implement a Deep Reinforcement Learning Algorithm, namely DQN, but without taking into account so much in detail the implementation of safe decision making. Moreover, instead of using two buffers, we will focus on prioritized replay.

3 Dataset

Since we are using a simulated environment, we will not need to use any external dataset. The environment will provide us with the necessary data to train and evaluate the agent. The environment provides us with a continuous state space and a discrete action space.

The shape of the input is as follows: `-inf, inf, (5, 5), float32`

- **presence**: 1.0 if a vehicle is present, 0.0 otherwise.
- **x**: World offset of ego vehicle or offset to ego vehicle on the x axis.
- **y**: World offset of ego vehicle or offset to ego vehicle on the y axis.
- **vx**: Velocity on the x axis of vehicle.
- **vy**: Velocity on the y axis of vehicle.

Note: the coordinates are relative to the ego-vehicle, except for the ego-vehicle which stays absolute. The world frame is at the top left-corner of the highway. The ego-vehicle is always centered on the lane, and its coordinates are relative to the world coordinate system, as shown below:

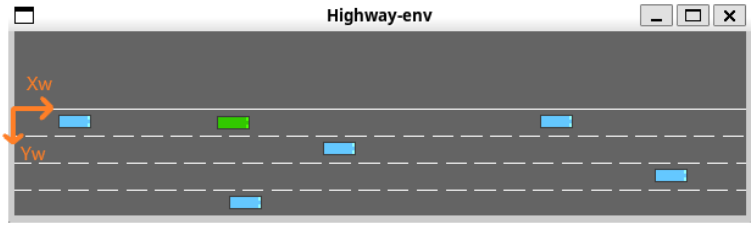

Figure 1: Environment with a world coordinate system at the top left corner of the highway.
ego-vehicle is represented by the green car and the other vehicles are represented by the blue cars.

Note that the coordinate system is fixed relative to the image, so it always remains at the top left corner of the highway. The ego-vehicle has always a x-coordinate of 1.0 and a y-coordinate from 0.0 to 0.12 depending on the lane it is in. Thus, each lane is 0.3 units wide.

4 Baseline

For the Baseline we used a simple agent that randomly chooses an action from the action space at each step with a uniform distribution. The action space is as follows:

Action index	0	1	2	3	4
Action name	lane change left	idle	lane change right	accelerate	decelerate

To evaluate the agent we used the following metrics that are best described in the Evaluation Metric section 6:

- **Collision rate**: 97% of the episodes ended in a collision.
- **Mean cumulative rewards**: Between 0.6 and 0.8 depending on the episode.

The results can better be visualized in the Results & Analysis section 7, where we also discuss the results.

5 Main approach

We will follow [**?**] and [**?**]. In the lecture, we already introduced RL algorithms, but let's introduce some notation before diving deeper. A Markov Decision process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S} \in \mathbb{R}^n$ is the state space, $\mathcal{A}$ the action space, $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ the state transformation function and $\mathcal{R}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the reward function. A policy is defined by $\pi\colon \mathcal{S} \to \mathcal{A}$, where $a_t = \pi(s_t)$ denotes the action at timestep t. Every action a_t leads to a state transition from s_t to s_{t+1} with a reward r_{t+1}. The goal is to find an optimal policy, such that the total accumulated reward is maximized:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$

where $0 < \gamma \leq 1$. Notice that we are working with a continuous state space and a discrete action space, given by

$$\mathcal{A} = \{0 : \text{lane change left}, 1 : \text{idle}, 2 : \text{lane change right}, 3 : \text{accelerate}, 4 : \text{decelerate}\}$$

Our first algorithm will involve Value Iteration, which is a model-based algorithm, which finds the optimal policy. Value Iteration leads to computing the state-action value, and acts greedily with respect to it. Since Value Iteration is only applicable to finite discrete MDPs, the environment is approximated by a finite-mdp environment, which builds on **env.to_finite_mdp()**. We will try to learn a deterministic policy. For that

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Figure 2: Value Iteration with the optimal policy π_* (see Sutton & Barto book).

we will assume that the transition model is built up in a simplistic manner, in the sense that each vehicle is driving at constant speed without changing the lanes. Our reset or start state will be defined as follows:

$$\text{transition} = \text{None}, \text{Rewards } R_t = 0, \text{max\_steps} = \#epochs$$

After that we will introduce a model-free approach, namely Deep Q-Network (or DQN). This algorithm is based on Q-Learning with function approximation, where the state-action value function Q is represented by a neural network. Since the approximation of the state-action value function leads to instability, experience

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 3: DQN (see Mnih et al.).

replay comes into play. Another observation is that DQN uses a second neural network, namely the target network. This is to ensure to obtain an unbiased estimator of the mean-squared Bellman error used in training the Q-network.

6 Evaluation Metric

For the evaluation of the agent we are using the following metrics:

- **Collision rate**: The percentage of episodes that ended in a collision. The goal is to have a collision rate of 0% because we can't trade-off safety for efficiency. So any collision is considered a failure.
- **Mean cumulative rewards**: The mean of the cumulative rewards of all the episodes. That is we collect the rewards for each step in the episode and then we take the mean of all the rewards for that specific episode. The goal is to maximize the mean cumulative rewards.

  The **reward function** depends on 4 factors that are described below:

- **Collision reward**: -1 if there is a collision, 0 otherwise. The episode ends when there is a collision.
- **High speed reward**: 0.4 if the speed is between 20 and 30 m/s, 0 otherwise.
- **Close to right lane reward**: 0.1 if the ego-vehicle is on the right lane, 0 otherwise. We prefer the ego-vehicle to be on the right lane because it is the safest lane and the road infrastructure is designed to be driven on the right lane such as signs, lane markings, and exit ramps. (assuming that the ego-vehicle is in a country where people drive on the right side of the road)

- **Standard deviation of the cumulative rewards**: The standard deviation of the cumulative rewards for all the episodes. The goal is to minimize the standard deviation of the cumulative rewards so that we can have a consistent agent.

7 Results & Analysis

The results we have so far are for the baseline agent. Based on the metrics we have defined in the Evaluation Metric section 6, we get the following results presented in figure 4 after running the agent for 100 episodes:

Figure 4: Results for the baseline agent.

As we can see from the results, the agent is not able to drive safely and efficiently. The collision rate is 97% and the mean cumulative rewards are between 0.6 and 0.8 depending on the episode. This means that the agent is colliding with other vehicles most of the time and it is not able to drive at the highest speed allowed, is not able to stay on the road and is not able to stay close to the right lane. Besides that, the standard deviations of the cumulative rewards are high relative to the means, which means that the agent is not consistent.

Based on these results, we got a better understanding of the environment and the agent as well as an intuition of the challenges that our agents might face. It is important to design our agent robust enough to drive safely and efficiently in the environment so that we can achieve a collision rate of 0% and a high mean cumulative rewards.

8 Future Work

In the next steps we will implement Value Iteration and DQN. All our approaches will be evaluated based on our defined evaluation metrics in section 6.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[2] Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

[3] Subramanya Nageshrao, Eric Tseng, and Dimitar Filev. Autonomous highway driving using deep reinforcement learning. *arXiv preprint arXiv:1904.00035*, 2019.

[4] Arash Mohammadhasani, Hamed Mehrivash, Alan Lynch, and Zhan Shu. Reinforcement learning based safe decision making for highway autonomous driving. *arXiv preprint arXiv:2105.06517*, 2021.

A Appendix: Raw state space samples

Sample 1

Vehicle	presence	x	y	vx	vy
ego-vehicle	1.0	1.0	0.1200	1.0	0.0
vehicle 1	1.0	0.1900	0.0700	-0.3700	-0.1800
vehicle 2	1.0	0.4300	0.1100	-0.2000	0.1400
vehicle 3	1.0	0.6400	0.0000	-0.2000	0.0000
vehicle 4	1.0	0.8800	0.0800	-0.1200	0.0000

Sample 2

Table 1: Sample 2

Vehicle	presence	x	y	vx	vy
ego-vehicle	1.0	1.0	0.0800	1.0	0.0
vehicle 1	1.0	0.2100	-0.0400	-0.1800	0.0000
vehicle 2	1.0	0.2400	-0.0800	-0.4800	0.0000
vehicle 3	1.0	0.5700	-0.0800	-0.3400	0.0000
vehicle 4	1.0	0.9500	-0.0800	-0.1600	0.0000

Sample 3

Vehicle	presence	x	y	vx	vy
ego-vehicle	1.0	1.0	0.0500	0.6200	-0.1400
vehicle 1	1.0	0.0100	-0.0300	-0.0900	0.1400
vehicle 2	1.0	-0.2400	-0.0500	0.2200	0.1400
vehicle 3	1.0	0.2500	-0.0500	0.3500	0.1400
vehicle 4	1.0	0.8000	0.0300	0.3300	0.1400

Sample 4

Vehicle	presence	x	y	vx	vy
ego-vehicle	1.0	1.0	0.0	1.0	0.0
vehicle 1	1.0	0.0600	0.0800	-0.4200	0.0000
vehicle 2	1.0	0.2500	0.0000	-0.4600	0.0000
vehicle 3	1.0	0.4800	0.0400	-0.5000	0.0000
vehicle 4	1.0	0.7700	0.0800	-0.4300	0.0000

Sample 5

Vehicle	presence	x	y	vx	vy
ego-vehicle	1.0	1.0	0.0800	1.0	0.0
vehicle 1	1.0	0.1600	-0.0400	0.0700	0.0000
vehicle 2	1.0	0.2200	-0.0800	-0.2000	0.0000
vehicle 3	1.0	0.5700	-0.0800	-0.0600	0.0000
vehicle 4	1.0	0.9300	-0.0100	-0.0500	-0.3300

Sample 6

Vehicle	presence	x	y	vx	vy
ego-vehicle	1.0	1.0	0.0800	1.0	0.0
vehicle 1	1.0	-0.1600	-0.0800	-0.0300	0.0000
vehicle 2	1.0	0.2600	0.0000	-0.0200	0.0000
vehicle 3	1.0	-0.3300	-0.0400	-0.0600	-0.0100
vehicle 4	1.0	0.6800	-0.0800	0.0200	0.0000