Autonomous Driving Systems in CARLA

Mohammadhossein Karimi Rozveh
McGill University
Montrèal, Quèbec
mohammad.karimirozveh@mail.mcgill.ca

Mohammad Arabzadeh
McGill University
Montrèal, Quèbec
mohammad.arabzadeh@mail.mcgill.ca

Abstract

This study investigates the efficacy of Deep Q-Networks (DQN) for autonomous driving within the CARLA simulation environment. A DQN agent was developed to autonomously navigate a vehicle along a set route, aiming to avoid collisions and remain on the roadway. The environment was simplified by excluding pedestrians and other vehicles to focus exclusively on the driving task. Various experiments were conducted to refine the action space and reward function. The findings suggest that DQN may not be ideal for this application, indicating a potential need to explore alternative approaches like deep deterministic policy gradient (DDPG) for further performance assessment.

Roles of the team members:
M. Arabzadeh: Developing the Carla simulation environment
M. Karimi Rozveh: Developing the DQN agent

1 Introduction

The rapid development of artificial intelligence and advancements in machine learning algorithms have initiated significant progress in task automation. Autonomous driving, in particular, presents one of the most difficult challenges in this field due to the complexity and unpredictability of driving conditions and the diversity of possible scenarios.

This project aims to implement a deep reinforcement learning model in the CARLA simulator to manage a vehicle within a virtual setting, building on the findings of a previous study [1]. The main objective is to navigate the vehicle along a specified path to a destination while avoiding collisions and staying on the road. This experiment specifically excludes pedestrians and other vehicles to focus solely on the driving task.

This project introduces a custom-developed deep Q-network (DQN) framework, incorporating a replay buffer and threaded training capabilities. It offers a choice between multilayer perceptron (MLP) and convolutional neural network (CNN) architectures[1]. Additionally, a specialized simulation environment was created in CARLA, featuring the ability to define routes, design reward functions, and set other specifications tailored to the project's needs.

2 Background

2.1 CARLA Simulator

CARLA is an open-source graphical simulator that has been leveraged as a cost-effective tool to facilitate studies on autonomous urban driving [2]. This framework also offers a Python API that allows researchers to link the graphical interface to a controllable back-end, offering various options such as deploying machine learning models to control the agent within the simulation environment.

[1]Although in this project, we only explored the MLP structure.

2.2 DQN Agents

Deep Q-Networks revolutionized reinforcement learning by integrating deep neural networks into the Q-learning framework, aiming to approximate the optimal action-value function $Q^*(s, a)$. In DQN, a deep neural network takes the state s as input and outputs Q-values for all possible actions a, enabling the agent to select actions based on maximizing Q-values. The training process involves iteratively updating the Q-network's weights to minimize the discrepancy between predicted and target Q-values, computed using a target network and the Bellman equation. Q-learning rules guide this process, where the agent learns by updating Q-values based on rewards obtained from its actions and the maximum Q-value of the next state, promoting the selection of actions that lead to higher cumulative rewards over time. In this project, our objective is to train a DQN model to determine optimal actions from a predefined set of actions, as detailed in the previous section.

3 Methodology

3.1 Problem Formulation

To be able to apply any RL algorithm in an intended environment, we define a Markov decision process consisting of a state space (S), an action space (A), a state transition probability (P), reward function (R).

State space Defined as $[wp_0, wp_1, \cdots, wp_5, v_t, d_t, \phi_t]$, where wp_0 is the closest road waypoint[2] to the car, and wp_1 to wp_5 are five next waypoints on the road. v_t is the vehicle velocity, d_t is the distance of the vehicle to the closest path waypoint, and ϕ_t is the angle between the car front and the road direction in time-step t.

Action space The discrete action space consists of three main controls: acceleration, steering to the right, steering to the left, and braking. Depending on the intensity of each action, they can be subdivided into more specific actions; for instance, full steering vs steering only halfway. Different configurations of these actions have been experimented in this project.

State transition probability The probability that action a in state s_t at time t will lead to state s_{t+1} at time $t + 1$: $P_a = P_r(s_{t+1}|s_t, a_t)$. These probabilities define how the vehicle's state changes in response to specific control commands under varying states. By accurately modeling these transitions, the DQN can better predict and optimize the sequence of actions that lead to successful navigation and task completion, adhering to the core principles of the Markov Decision Process.

Reward function The reward function consists of three components; the car's velocity v_t, the angle between its direction and the road direction ϕ_t, and its distance to the closest path waypoint. The reward at timestep t is defined as:

$$R_t = \begin{cases} +100, & \text{if reached the destination,} \\ -200, & \text{if collision or line crossing detected,} \\ |v_t|(|\cos\phi_t| - |\sin\phi_t| - d_t), & \text{otherwise.} \end{cases} \tag{1}$$

Figure 3 shows how the reward changes when d_t and ϕ_t change.

3.2 DQN Model Architecture

The architecture of the DQN model utilized for learning and estimating the Q-values in this simulation is depicted in Figure 4. The model takes into account a set of environmental features, including the predefined waypoints crucial for directing the vehicle along the designated path, as well as a set of car-specific features such as the car's velocity denoted by V_t, the distance d_t from the closest waypoint, and Φ_t, representing the yaw angle relative to the road direction.

The proposed Model consists of a fully connected network with two hidden layers, with the first and second layer having 100 and 200 neurons respectively with ReLU activation and the output layer with $N_{actions}$ neurons, where $N_{actions}$ represents the number of defined actions.

[2]In CARLA, the waypoints are the nodes defining the roadways. Each waypoint contains location and direction information. In this project, we use the location coordinates of the waypoints.

4 Experimental Results

Three experiments were conducted with different parameters. The specifications and hyper-parameters of each experiment are listed below. In all experiments, the Adam optimizer function was used. The episode finishes when the car collides with the environment, goes off the main road, or reaches the destination.

Experiment 1 **Reward:** large reward values may cause large gradients according to [3]. To avoid this, the reward function in Equation 1 was divided by 100.

Actions: [accelerate, steer left, steer right, brake]

Other hyper-parameters: 500 episodes. A replay buffer with minimum and maximum sizes of 1,000 and 5,000 was used. The model was trained 10 times per second. Discount factor of 0.99. Learning rate of 0.001. Batch size of 32. Epsilon decays from 1.0 to 0.1 over 5,000 steps.

Experiment 2 **Reward:** Used $R'_t = R_t/200 - 1$ as the reward function where R_t is the reward in Equation 1. In this case, the reward values are normalized to be between -2 to 0.

Actions: [accelerate, steer left while moving forward, steer left while slowly moving forward, steer right while slowly moving forward, steer right while moving forward]. Note that brake was removed in this experiment to force the agent to move.

Other hyper-parameters: 1000 episodes. A replay buffer with minimum and maximum sizes of 10,000 and 100,000 was used. The model was trained 20 times per second. Discount factor of 0.9975. Learning rate of 0.0001. Batch size of 32. Epsilon decays from 1.0 to 0.01 over 200,000 steps.

Experiment 3 **Reward:** The division factor in experiment 2 was used. The third part of Equation 1 was modified to $|v_t|(|\cos\phi_t|^2 - 2|\sin\phi_t| - d_t)$ to achieve a smother curve with higher penalty for deviation. Moreover, a new component of $0.2t/t_{max}$ was subtracted from the reward, where t is the elapsed time in the episode (in seconds), and t_max is the maximum duration of an episode. This means that as the episode becomes longer, the agent receives more penalty.

Actions: [accelerate, steer left while moving forward, steer left while slowly moving forward, steer right while slowly moving forward, steer right while moving forward, brake]

Other hyper-parameters: 1500 episodes. A replay buffer with minimum and maximum sizes of 10,000 and 100,000 was used. The model was trained 20 times per second. Discount factor of 0.9975. Learning rate of 0.0001. Batch size of 128. Epsilon decays from 1.0 to 0.05 over 200,000 steps.

Figure 5 shows that the agent did not perform well in these experiments. It is seen that the reward doesn't improve over time. Moreover, the car never reaches the goal point in any of the experiments. The loss figures show an improvement in the first training epochs ($\sim$first 10,000 epochs), but then it degrades afterward. More hyper-parameters need to be tested in order to reach better results. We may need to have a smaller learning rate, a higher number of episodes, with more number of exploration steps. DQN models perform well, but they usually need a long training process.

5 Conclusion and Future Work

As seen in Figure 5, the purposed DQN model wasn't able to reach an adequate performance in our experiments. It is essential to explore other methods such as DDPG to solve this problem. The action space in our environment is continuous, while DQN works only with a discrete space which is why we discretized our actions. On the other hand, DDPG is known for its compatibility with continuous action spaces; therefore it might perform better than our DQN model.

In this project, we used the next waypoint coordinates in front of the car as states. However, in reality, we don't have access to those points. In that case, we need to rely on the camera sensors to train the agent. Therefore, we need to use structures such as convolutional neural networks to train the model. This requires a higher processing power and longer training periods (a couple of days) to achieve good results.

In this project, we assumed that there are no pedestrians or other vehicles in the environment. An interesting problem would be to have such objects in the environment to have more realistic scenarios.

References

[1] Óscar Pérez-Gil et al. “Deep reinforcement learning based control for Autonomous Vehicles in CARLA”. In: *Multimedia Tools and Applications* 81.3 (2022), pp. 3553–3576.

[2] Alexey Dosovitskiy et al. “CARLA: An open urban driving simulator”. In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.

[3] Hado van Hasselt et al. “Learning values across many orders of magnitude”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS’16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 4294–4302. ISBN: 9781510838819.

I Appendix: Figures

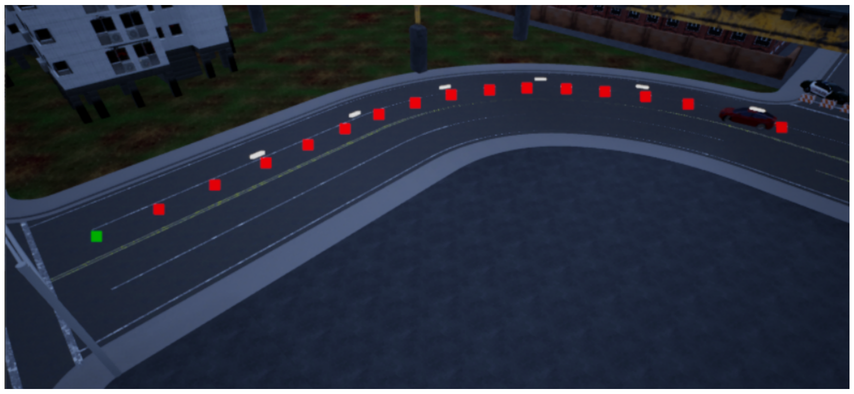


Figure 1: The designated route for the car to follow. The path waypoints are shown with red squares and the destination is shown with a green square.

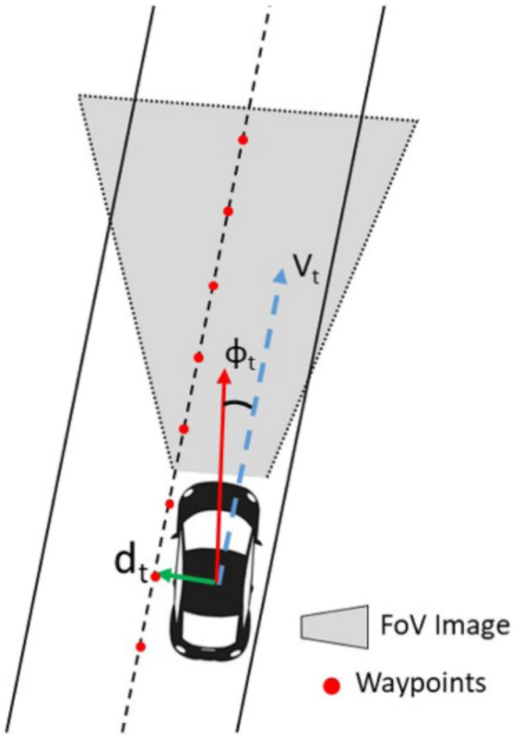


Figure 2: State space definition. d_t denotes the distance of the car from the closest waypoint of the path. ϕ_t denotes the angle between the vehicle's direction and the road direction. Finally, v_t denotes the velocity of the vehicle.

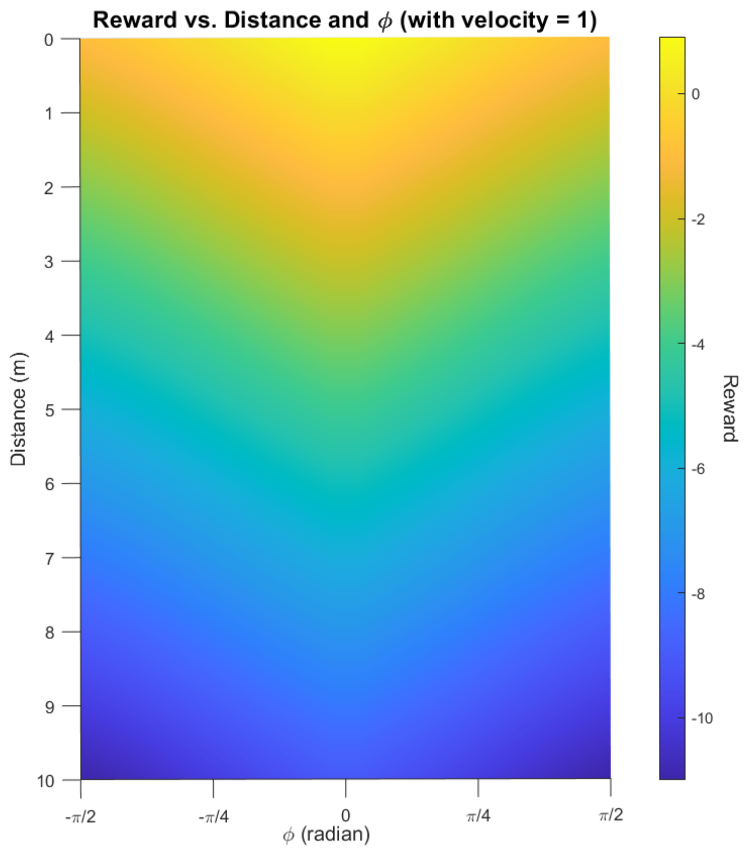


Figure 3: Reward value vs the values of d_t and ϕ_t when $v_t = 1$

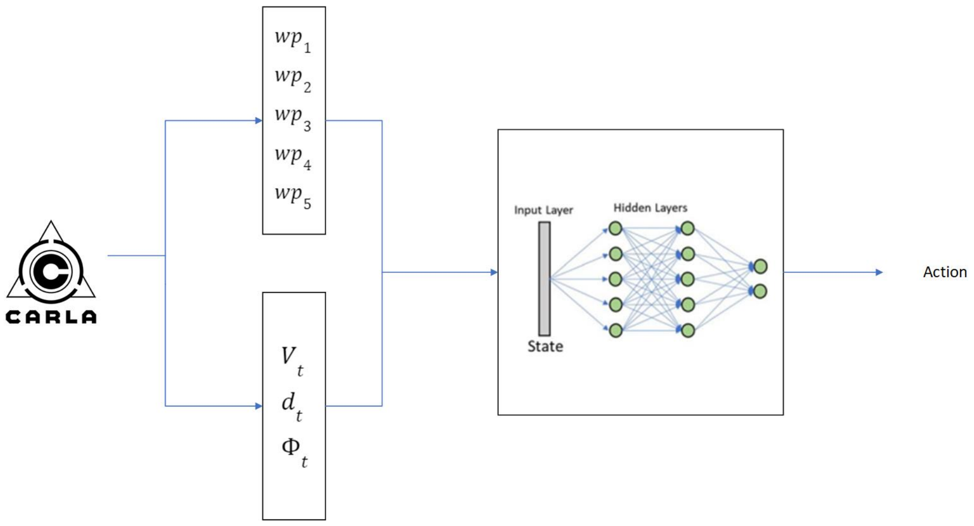
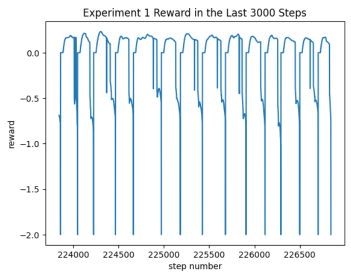
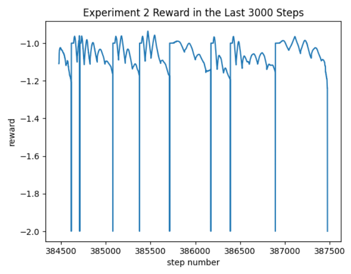


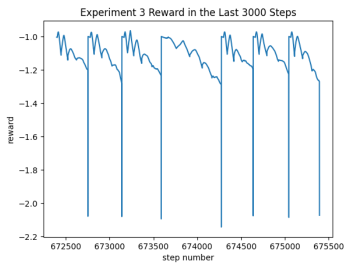
Figure 4: The structure of the DQN.



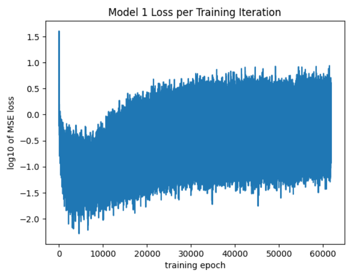
(a)



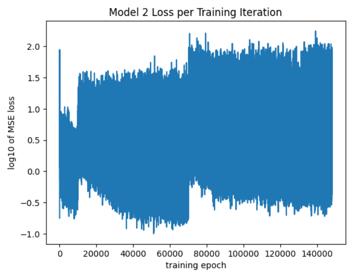
(b)



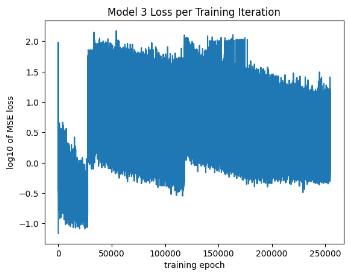
(c)



(d)



(e)



(f)

Figure 5: Experiment results. The top figures show the final 3000 step rewards and the bottom figures show the log10 values of the MSE loss of the model throughout the training.