ENPM690 Project: Decision-Making for Autonomous Vehicle Overtaking using Dueling Double DQN

Aditya Varadaraj UID: 117054859 Saurabh Palande UID: 118133959

***Abstract*—Vehicle overtaking is a critical task in traffic scenarios that requires efficient decision-making and maneuvering skills. In this project, we propose a reinforcement learning approach utilizing the Dueling Double Deep Q-Network (Dueling Double DQN) algorithm to train an autonomous vehicle agent for safe and effective overtaking maneuvers in traffic scenarios. Through interactions with a specialized simulation environment, the agent will gain the essential abilities to make informed decisions regarding acceleration, deceleration, lane changes, and overtaking slower vehicles. This training will encompass adhering to traffic regulations and prioritizing passenger safety. The Dueling Double DQN architecture enhances the agent's learning and estimation capabilities by separately handling state value and action advantages, resulting in improved accuracy and stability of the value function approximation.**

I. Introduction

Overtaking slower vehicles on highways is a critical and demanding aspect of driving that necessitates precise decision-making and maneuvering skills. Developing autonomous vehicles capable of executing overtaking maneuvers effectively presents challenges, requiring dynamic assessment of traffic conditions, prediction of other vehicles' behavior, and appropriate action execution.

Reinforcement learning (RL) has emerged as a promising method for training autonomous agents to make intelligent decisions in complex and dynamic environments. The Dueling Double Deep Q-Network (Dueling Double DQN) is a successful RL algorithm that combines the benefits of Dueling DQN and Double DQN. The Dueling DQN architecture enables the agent to estimate state value and action advantages separately. This decoupling enhances the accuracy of the value function representation and improves learning efficiency. The Double DQN algorithm addresses overestimation bias by using a separate network during the learning process.

In this project, our goal is to leverage the Dueling Double DQN algorithm to train an autonomous vehicle agent capable of safely and efficiently performing overtaking maneuvers on highways. We will extensively train and evaluate the performance of the Dueling Double DQN-based agent in various overtaking scenarios. This evaluation will include measuring the success rate of completing overtaking maneuvers, adaptability to changing traffic conditions, and impact on traffic flow.

This project aims to advance our understanding of RL-based approaches for vehicle overtaking, contributing to the development of safer and more efficient autonomous driving systems. The insights gained from this project can inform the design and implementation of intelligent agents capable of performing complex driving tasks, improving road safety and efficiency in the future.

II. Methodology

As per [4], Q-function is defined as the expected total reward given the current state(s), current action(a) and policy to be followed (π). So,

$$Q_\pi \;=\; E[r_1 + \gamma r_2 + ... | s_0 = s, a_0 = a, \pi] \tag{1}$$

The Q-value is determined by weighing the impact of immediate and future rewards using a discount factor (γ). To find the best policy, we maximize the Q-value over all possible actions in the action set. In the most basic version of Q-Learning, we update the Q function based on something called the Temporal Difference (TD) error which is the error between estimated value obtained by maximizing Q function for next state over all actions and adding it to the current/immediate reward and the true value according to the previous Q function. Most real-life scenarios are too large to learn all action values and all states separately. Thus, we parametrize $Q(s, a)$ as $Q(s, a; \theta_t)$.

A. DQN

In DQN (Deep Q-Learning Network) [2], we use a neural network to learn these parameters used to determine the Q-value for each action in the action space given the current state. DQN is characterized by 2 main features:

- **Experience Replay:** DQN uses 2 networks (main network (θ) and target network(θ^-)) having same architecture. The main network gets trained regularly while the target network gets updated by the weights from main network every τ epochs.
- **Use of Neural Network:** As discussed, 2 neural networks are used (main and target). The **target network** is used for both **selection** of action that maximizes the Q value and the **evaluation** of corresponding Q value in the update step.

The parameters and Q-value are updated as follows using Gradient Descent:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha(Y_t^{DQN} - Q(s_t, a_t; \theta_t))\nabla_{\theta_t} Q(s_t, a_t, \theta_t) \\ , where,\ Y_t^{DQN} &= r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_t^-) \\ \theta_t^- &= \theta_t \ every\ \tau\ episodes \end{aligned} \tag{2}$$

B. Double DQN

As discussed above, in DQN, the max operator uses the same target network to both select action for optimum Q-value and evaluate the Q-value corr. to that action. Note that Q value itself is an estimate. By taking the max of estimates, we are essentially obtaining an estimate of the actual maximum Q-value. This is called **overestimation bias**. We can reduce this bias by decoupling the selection and evaluation using Double DQN.

In Double DQN [2] [1], the main network is used purely for selecting the optimal action and the target network is used purely for evaluation of corresponding Q value. Double DQN can learn faster and reach better performance than DQN in some environments. Since the same 2 networks are being used, there is no significant increase in computational complexity.

The update rule remains same as DQN except for the target estimate Y_t defined as follows:

$$Y_t^{DoubleDQN} = r_{t+1} + \gamma Q(s_{t+1}, arg\max_a Q(s_{t+1}, a; \theta_t); \theta_t^-) \tag{3}$$

C. Dueling Double DQN (D3QN)

As per [3], Dueling DQN just changes the networks' architecture. The output of the final dense layer is decoupled into 2 streams: Value and Advantage (Fig. 1). State value stream estimates the value (V) of being in a particular state. Advantage stream estimates the advantages (A) of each possible action in that state defined originally as $A = Q - V$. Then it combines these values to compute the Q value. But just saying $Q = V + A$ would make the network unidentifiable, i.e., given Q, we cannot recover V and A uniquely. This problem is solved by subtracting the advantage by its mean. On the one hand this loses the original semantics of V and A because they are now off-target by a constant, but on the other hand it increases the stability of the optimization.

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) - \frac{1}{|A|}\sum_{a'} A(s, a'; \theta, \alpha) \tag{4}$$

Using this network architecture for the main and target networks used in Double DQN discussed above, we get Dueling Double DQN (D3QN). The Epsilon-Greedy Algo is used for action selection to maintain balance between Exploration-Exploitation.

$$\begin{aligned} &\epsilon = \epsilon - \delta\epsilon \\ &x = random.uniform(1) \\ &a = \begin{cases} action\_space.sample() & if\ x < \epsilon \\ arg\max_{a'} Q(s, a'; \theta) & otherwise \end{cases} \end{aligned} \tag{5}$$

The D3QN algorithm is as in Algorithm 1.

D. Network Architecture

It consists of passing the 5×5 input through Dense layer of output size 32 followed by flattening the input and then passing it through 3 Dense Layers of output size 64, 64 and 128 respectively. Then it is divided into 2 streams. The output of this 4^{th} layer is passed through a Dense layer of output size of 64 and then passed through an output layer of size 1 to get V. The output of 4^{th} layer is passed through a Dense layer of output size of 64 and then passed through an output layer of size $num\_actions = 5$ to get A.

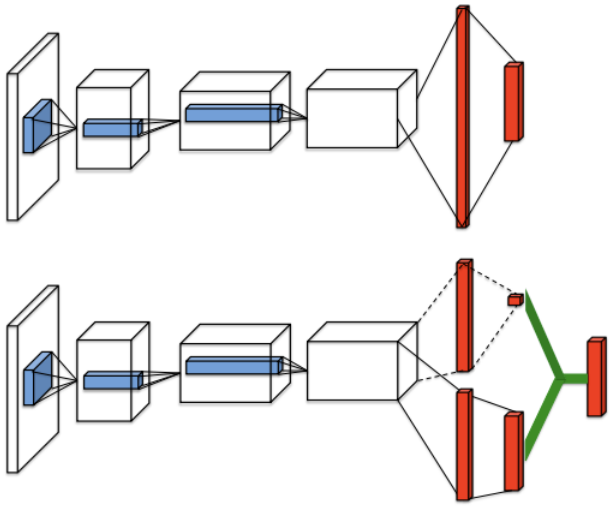

Fig. 1. Dueling Network Architecture [3]

Optimizer used was Adam and activations were ReLU. Loss used was MSE loss.

E. Simulation Environment And Setup

The model was trained on Google Colab. The following steps were followed for the Simulation and Training environment setup:

1) Choose a Simulation Framework:
   The first step involved selecting a suitable simulation framework that supports highway environments, such as SUMO, CARLA, or Highway-env. For this project, the OpenAI highway env environment was chosen. Following the framework selection, the subsequent step was to install the chosen framework and its dependencies by following the provided instructions to visualize the environment and its output in Google Colab using a virtual display.
2) Define and configure the simulation parameters:
   Once the simulation environment was selected, the subsequent task involved defining the specific characteristics of the highway environment. This included determining the number of lanes, setting the desired traffic density, specifying the speed of vehicles, and assigning appropriate rewards based on the defined problem statement. By configuring these parameters, the highway environment was tailored to reflect the intended scenario accurately.
3) Setup the reward scheme:
   An effective reward scheme is crucial for the proper functioning of any reinforcement learning algorithm. In this project, a carefully designed reward scheme was

Algorithm 1 Dueling Double DQN

Initialize Main network $Q(s, a; \theta)$ and Target Network $Q(s, a; \theta^-)$, replay buffer $\mathcal{D}$ and ϵ

for each episode **do**

for each environment step **do**

Observe State s_t and Select action a_t based on epsilon-greedy method.

Execute the action a_t and observe next State s_{t+1} and reward r_t

Store $(s_t, a_t, r_t, s_{t+1}, done)$in Replay Buffer $\mathcal{D}$

if $|\mathcal{D}| \geq batch\_size$ **then**

sample $e_t \leftarrow (s_t, a_t, r_t, s_{t+1}, done) \in \mathcal{D}$

Update $Y_t^{DoubleDQN}$ using (3)

Use Gradient Descent on MSE Loss to update θ, i.e., train the main network

end if

end for

if $episode\%10 = 0$ **then**

Update Target Network Weights ($\theta_t^- = \theta_t$)

end if

$\epsilon \leftarrow \epsilon - \delta\epsilon$ if $episode < 250$

Save Losses, Rewards and Model

end for

employed to guide the learning process. The chosen reward scheme aimed to provide meaningful feedback to the agent, encouraging favorable behaviors and discouraging unfavorable actions. It involved assigning positive rewards for desired outcomes and negative rewards were assigned for collisions. By implementing this reward scheme, the agent was able to learn and optimize its behavior, ultimately facilitating safe and efficient overtaking maneuvers within the project's defined parameters.

collision reward: -1 The reward received when colliding with a vehicle.

right lane reward": 0 The reward received when driving on the right-most lanes, linearly mapped to zero for other lanes.

high speed reward: 0.4, The reward received when driving at full speed, linearly mapped to zero for lower speeds.

lane change reward: 1, The reward received at each lane change action.

reward speed range: [25, 35]

4) Run the Simulation and Train the D3QN network: After completing the necessary setup, the subsequent step involved initiating the simulation and training loop to execute the desired number of iterations. Throughout this process, the TensorFlow model was saved at each iteration to capture the progress of the agent's learning. Additionally, the loss function and episodic rewards were monitored and recorded to evaluate the performance of the agent in overtaking maneuvers and other pertinent metrics. This data served as valuable insights to assess the agent's progress, refine the training process, and make informed decisions regarding the optimization of overtaking capabilities in the simulation environment.

III. Results

The hyperparameters used were:

- No. of epsiodes: It trained for 410 episodes due to GPU limitations on Google Colab.
- $\epsilon = 1$ initially and decays with $\delta\epsilon = 0.0038$ for first 250 episodes then remains fixed.
- Discount Factor, $\gamma = 0.99$
- Target Network weights are updated every $\tau = 10$ epsiodes.

The following results were obtained after training the model for around 35000 iterations and 410 episodes: Videos capturing the simulation output for episode 20, 200, and 400 were recorded to provide visual representations of the agent's performance. These videos serve as valuable resources for observing the agent's behavior and understanding its progress throughout the training process. Here are the links to access the recorded videos:

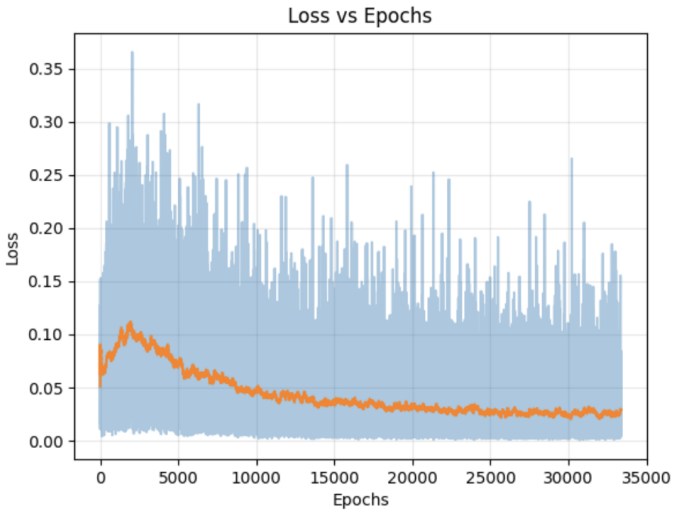

Fig. 2. Loss Vs Epochs

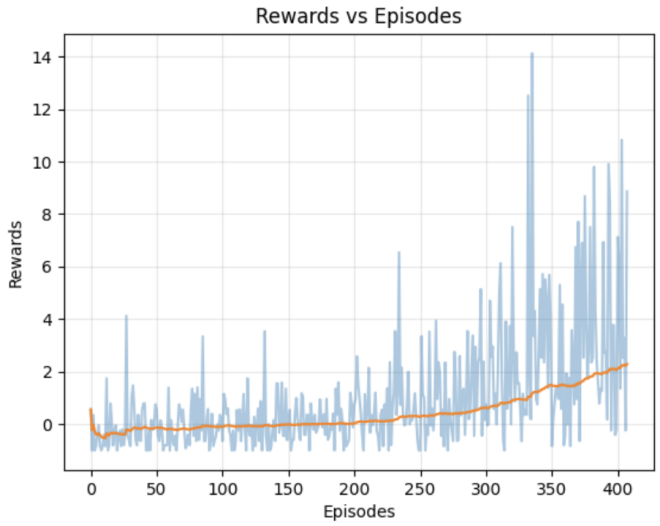

Fig. 3. Rewards vs Episodes

1) Episode 20 : Video
2) Episode 200 : Video
3) Episode 400 : Video

Our Code: GitHub Link

IV. Discussion

By analyzing the Loss (Fig. 2) and Rewards (Fig. 3)graphs, it is evident that the loss consistently decreases over time, while the reward consistently increases. This implies that the model is able to learn with each episode.

By reviewing these videos, one can gain a better understanding of the agent's overtaking maneuvers and assess its ability to navigate the highway environment effectively. This implies that the vehicle's ability to maneuver overtaking scenarios gets better with number of episodes.

V. Conclusion

1) Thus, to summarize, we can conclude that Dueling Double DQN performs well for the autonomous vehicle overtaking task with proper selection of environment variables and hyperparameters.
2) The model's learning capacity might improve with a higher number of iterations. However, due to our training limitations on Google Colab, we reached the maximum number of iterations possible within the available GPU resources.

VI. Acknowledgements

1) **Saurabh:**
Implemented various helper functions for setting up the environment, managing the replay buffer, training the deep learning model, and simulating the environment. These functions were designed to streamline the workflow and ensure smooth integration of the different components of the project. Drafted the Abstract, Introduction, Methodology(Simulation and setup), Results, Discussion, and Conclusion sections of the report
2) **Aditya:**
Developed the network architecture for Dueling Double DQN, ensuring its compatibility with the project requirements and also to create a robust and effective model.Conducted extensive experiments to optimize the hyperparameters of the Dueling Double DQN algorithm. Drafted the Methodology (DQN, Double DQN, Dueling Double DQN, Network Architecture), and References sections of the project report.

References

[1] S. Mo, X. Pei and Z. Chen, "Decision-Making for Oncoming Traffic Overtaking Scenario using Double DQN," 2019 3rd Conference on Vehicle Control and Intelligence (CVCI), Hefei, China, 2019, pp. 1-4, doi: 10.1109/CVCI47823.2019.8951626 .
[2] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Thirtieth AAAI conference on artificial intelligence, 2016
[3] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." arXiv preprint arXiv:1511.06581 (2015)
[4] Steve Brunton's videos on Reinforcement Learning