POLITECNICO DI TORINO

Master of science program
in ICT For Smart Societies

Master Thesis

A deep reinforcement learning approach to cooperative cruise control for connected cars

Supervisor
prof. Carla Fabiana Chiasserini
prof. Nicola Amati

Candidate
Keyu Wang

ACADEMIC YEAR 2021-2022

If you renew yourself for one day, you can renew yourself daily, and continue to do so.
[Confucius, *Liji*]

Summary

In this thesis, we use a deep reinforcement learning algorithm called Deep Deterministic Policy Gradient (DDPG) to achieve vehicle longitudinal control in the CoMoVe framework. And we properly design the reward function, considering some key performance indicators like safety, comfort and stability to guide the model's learning direction. In the meanwhile, we apply multiple methods to reduce overfitting: fine-tuning hyperparameters, diversifying training scenarios and regularization, which improve the performance and robustness of DDPG. In addition, we also test its improved version called Twin Delayed DDPG (TD3) algorithm for comparison as well as providing reference for future research.

Key words: *reinforcement learning, Deep Deterministic Policy Gradient, CoMoVe, longitudinal control, reward function, overfitting, Twin Delayed DDPG*

Acknowledgements

First of all, my biggest thanks to my supervisor, prof. Carla Fabiana Chiasserini. She gives me tremendous help throughout the period of writing this thesis and provides invaluable guidance and insights at every stage. She always replies me promptly when I have doubts. I will always be grateful for her help and trust for providing this opportunity for me to participate in this thesis project.

I'd also like to thank prof. Nicola Amati from the Center for Automotive Research and Sustainable Mobility for his supervision of my thesis project.

A special thanks to Dinesh Cyril Selvaraj, PH.D. student from Telecommunication Networks Group, whose advice and support are crucial to my thesis.

I'm grateful to all my classmates and friends from ICT for smart Societies, we have learned from each other and made progress together during this period, which is a unforgettable experience in my master's program.

Last, I would like to thank my parents for their continued encouragement and support, which gives me the power I needed to complete my master's program and also become a priceless asset for my future development.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the development of the world economy, cars have become an important method of transportation for people mobility. According to the forecast of the International Energy Agency, in the global the number of passenger car will double to 2 billion vehicles by 2050[1]. As a result, serious road safety problems arise. Worldwide, road traffic accidents have caused nearly 1.3 million preventable deaths and an estimated 50 million injuries each year over the past 20 years and the World Health Organization predicts a further 13 million deaths and 500 million injuries on the roads in the next decade[2]. To solve these problems, various technologies are being developed. One of advanced applications is called intelligent transportation systems(ITS) aiming at enabling convenient transportation management to maintain a safe, efficient and sustainable transportation network, reduce traffic congestion and improve driver experience.

In order to better evaluate the performance of ITS, the related research group developed the CoMoVe framework[3] to solve this problem. The CoMoVe framework includes four main modules: vehicle mobility, vehicle dynamics, vehicle communication and vehicle control module. The vehicle control module applies the model-based method, so the design of the control logic needs to use the prior knowledge and internal principles of the environment, which makes the control algorithm complex, difficult to fine-tune, and cannot meet the needs of customization. However, in recent years, due to the rapid development of deep learning and its strong learning ability and good adaptability, it is widely used in computer vision, risk control, natural language processing and many other fields. So, there is an increasing amount of researches on how to apply deep learning techniques on ITS[4]. In this thesis, we use the deep reinforcement learning(DRL) algorithm to realize the vehicle control module of the CoMoVe framework, which takes into account passenger comfort and vehicle stability while assuring driving safety.

The remaining chapters are organized as follows: Chapter 2 mainly introduces the background knowledge related to CoMoVe framework and reinforcement learning,

as well as the details of DRL algorithm frameworks and performance improving methods; Chapter 3 mainly introduces the key steps in the process of integrating deep reinforcement learning algorithms into the CoMoVe framework; Chapter 4 mainly introduces the simulation scenarios, preliminary and optimized performance evaluation and analysis of deep reinforcement learning; Chapter 5 contains the thesis summary of and the future outlook.

Chapter 2

Background

2.1 Connected cars

The Internet of Vehicles can be seen as an extension of the Internet of Things in the automotive industry. The connected cars in the network can use communication technology to exchange information with other entities[5].
In the driving scenario, connected cars collect information including vehicle driving status, road conditions from chassis, powertrain, surrounding cars and infrastructures through electronic control units (ECUs)[6]. These collected information will be further processed and shared to facilitate the related services such as on-board diagnostics, collision avoidance, platooning and so on, providing customers with safe, comfortable and intelligent driving experience.

2.1.1 Network architecture

The network architecture of connected vehicle is usually built as a layered structure, where each layer plays a different role. Currently, there is no specific architecture because of the existing different comprehension of connected cars[7]. However, from a logical point of view, it can be divided into the following layers:

1) **Perception layer:** The perception layer is the basis of recognition services, which collects information of vehicle status and road environment mainly through multivariate information collection via radar, vehicle camera and GPS[7]. The information will then be transmitted to the actor (driver or computer). Actor will make appropriate actions based on feedback so as to realize assisted or automatic driving.

2) **Network layer:** The main function of this layer is to realize network transmission, data sharing and processing, and to provide low-latency, high-speed and wide-range network connections for connected vehicles[8].

3) **Application layer:** The application layer can provide a variety of services[8], such as autonomous driving, in-vehicle entertainment, remote monitoring and emergency rescue. With the development of AI technology, some resource-intensive applications can also be realized to meet the growing market demand.

2.1.2 Types of connectivity

In real-world scenarios, connected cars will communicate with various entities in the environment including pedestrians, traffic lights, other vehicles, and cloud databases. To sum up, there are five types of connectivity[9]:

1) **Vehicle-to-infrastructure(V2I):** V2I technology uses infrastructure as a medium, which collects information about safety status, environment and road congestion information from vehicles, and forward these obtained data to others with better decision-making suggestions[10].

2) **Vehicle-to-vehicle(V2V):** V2V technology refers to the communication method used when exchanging information between cars. The information exchanged mainly includes vehicle speed, distance and geographic location, etc., which are crucial for avoiding collisions and improving traffic efficiency in congested areas[11].

3) **Vehicle-to-cloud(V2C):**V2C technology refers to the exchange of information between the car and cloud database. Due to the wide application of cloud technology, automobile-related data can be access by other industries (such as smart grids, gas stations, and high-precision maps), realizing mutual promotion and common development of the entire automotive ecosystem[12].

4) **Vehicle to Pedestrian(V2P):** V2P technology mainly refers to the communication between cars and pedestrian electronic devices, which enables cars to sense the surrounding vital signs in time and make reasonable actions, reducing the risk of accidents and greatly improving road safety[13].

5) **Vehicle-to-everything(V2X):** V2X technology generally refers to the communication between cars and all entities that can be interacted with in the Internet of Vehicles, which includes other communication types such as V2V and V2I[14].

2.1.3 Key communication technology

Currently, dedicated short-range communications (DSRC) and cellular-based vehicle to everything communication technology(C-V2X) constitute the main standards for vehicular communications[14].

1) **DSRC:** DSRC is a vehicular communication technology based on the IEEE 802.11p standard led by the United States, which works in the 5.9GHz frequency band with bandwidth in range of 75Mhz. DSRC can support both vehicle-to-vehicle (V2V) and vehicle-to infrastructure (V2I) type of communications.
   DSRC device can be divided into two categories: on board unit(OBU) located inside the vehicles and road site unit(RSU) acts as fixed access point mounted along the roadside. These two type of units can transmit information to each other and OBU can also transmit the obtained information to the cloud platform. In order to ensure sufficient response time after receiving message, DSRC is required to have round trip latency below 50ms in safety application[14].

2) **C-V2X:** C-V2X mainly refers to LTE-V2X and 5G-V2X technologies with two communication modes: direct vehicle to vehicle communication and cellular network-based communication. Cellular communications rely on existing LTE and 5G cellular networks to enable information transmission with high bandwidth and long distances. On the contrary, direct communication allows device to device data transmission without infrastructure.
   C-V2X technology is easier to be recognized by the market than DSRC in the following aspects: First, cellular networks support communication over longer distances, which allows sufficient time for connection establishment between vehicles and base station. Second, the capacity of the cellular network is large, which supports the access requirements of high bandwidth and big data. As a result, the network switching frequency and control overhead are reduced while ensuring communication quality. In addition, C-V2X deployment is cost-effective, requiring no new infrastructure to be built, just upgrading existing equipment.[15].

2.2 Platooning scenario

Recent technologies in artificial intelligence and vehicle-to-vehicle (V2V) communication allow cooperative driving between vehicles. Vehicle platooning as a type of collaborative driving technology has received considerable attention over the past few decades due to the considerable benefits once implemented. Vehicle platooning systems are also known as Adaptive Cruise Control (ACC)[16] or Cooperative

Adaptive Cruise Control (CACC) systems[17], with the difference that CACC enables vehicular communication.
Platooning is defined as a scenario in which vehicles travel in line and maintain a safe distance with each other. Vehicles in platoon have the ability to communicate based on specialized in-vehicle electronics and relative communication standards. This allows a vehicle to share its own status (speed, acceleration, position, etc.) with other members, and as a result platooning improves the road capacity, traffic safety and travel efficiency[18].
Since the first platooning model was proposed in the 1950s[19], various models have been developed. Among them, the most basic one is linear feedback control model, which is designed to maintain a constant spacing[20] or time headway[21]. Recent studies have applied optimal control methods[22][23]whose objective functions can incorporate multiple targets such as comfort , efficiency and key safety constraints(e.g. collision avoidance).
Before commercialization, platooning technology must be thoroughly tested and validated to prove its safety. But since vehicle platoon is an interdisciplinary project, which involves control theory, communication engineering, software programming and so on. Its field experiments are often expensive and the simulation scale is far from reality. Although a number of researches have been performed in field test, most of these studies are conducted in constrained environments[24][25]. An alternative approach is to develop a virtual framework that can simulate the real driving environment, for which there are related researches and available simulation frameworks[26].

2.3 Vehicle dynamics

The subject of vehicle dynamics is very broad, but can be loosely defined as the analysis of the motion of vehicles with the aim of describing various behaviors and understanding their causes[27], which can be divided into three main parts: longitudinal dynamics, lateral dynamics and vertical dynamics[28]. Since this thesis only deals with platooning scenario, we mainly focus on the vehicle longitudinal dynamics, while also specifying the key indicators related to vehicle comfort and stability used in thesis.

1) **Longitudinal dynamics**
   In the longitudinal vehicle model, the traction force is provided by the powertrain, which mainly includes the engine, suspension, transmission system and other components. At the same time, it is necessary to provide a basis for describing the dynamic performance of vehicle by analyzing the forces balance including gravity, resistance and driving forces. Figure2.1[29] shows the external longitudinal forces acting on the vehicle when road slope angle is θ.

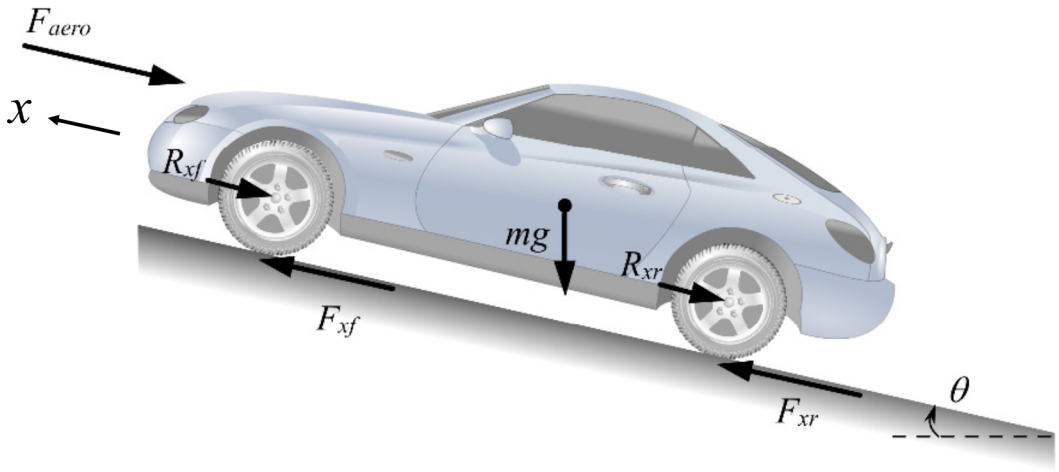

Figure 2.1. Longitudinal dynamics model

The force balance equation of the vehicle is expressed as follows:

$$m\ddot{x} = F_f + F_r - F_a - R_f - R_r - mg\sin(\theta) \tag{2.1}$$

where

- m is the mass of the vehicle
- $\ddot{x}$ is the vehicle acceleration
- F_f is the driving force of the front tires
- F_r is the driving force of the rear tires
- F_a is the aerodynamic resistance
- R_f is the rolling resistance of the front tires
- R_r is the rolling resistance of the rear tires
- g is the gravitational acceleration
- θ is the slope angle

2) **Comfort**
We use jerk to indicate driving comfort, which is defined as the derivative of acceleration and is considered to have a large impact on passenger comfort[30].

$$jerk = \dddot{x} \tag{2.2}$$

3) **Stability**
In this thesis, we choose slip ratio S_l to indicate the vehicle stability, which is defined as[31]:

$$S_l = \left(\frac{\omega r_e - v}{v}\right) \tag{2.3}$$

where

- ω is the wheel angular velocity
- r_e is the effective free-rolling radius of tires, which can be calculated from the revolutions per kilometer
- v is the vehicle velocity

As shown in Figure 2.2[32], the variation in slip ratio affects both lateral force coefficient μ_y and longitudinal force coefficient μ_x of the tire.

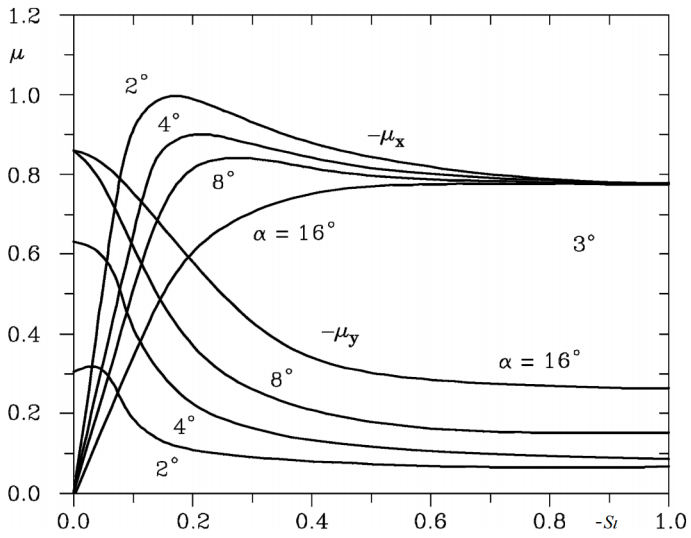

Figure 2.2. lateral and longitudinal force coefficients as function of slip ratio

For example, when side slip angle $\alpha = 2°$[32], as slip ratio increases the lateral force coefficient μ_y tends to zero, which means that tire can not provide enough lateral force to maintain direction. Therefore, slip ratio S_l is usually need to be controlled between 0.1 and 0.3 to obtain high vehicle stability.

2.4 Reinforcement learning

Reinforcement learning(RL) is an area of machine learning, which is listed as the three basic machine learning paradigms along with supervised learning and unsupervised learning, as shown in Figure 2.3[33].

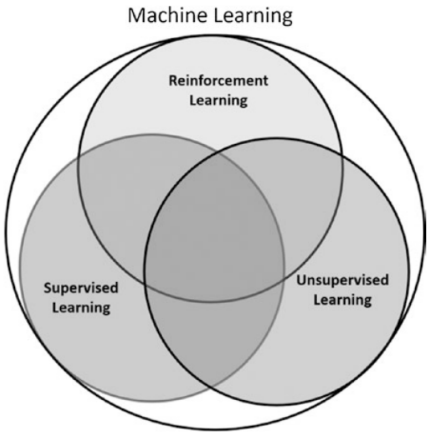

Figure 2.3. Branches of machine learning

Different from supervised and unsupervised learning, the the objective of RL algorithms is to figure out how the agent (the entity that executes the RL algorithm) take actions to maximize the cumulative reward in the environment. For example, when playing chess, the algorithm need to decide how to move pieces according to the current layout in order to win the whole game. Similar to supervised learning, reinforcement learning also has a training process. It needs to continuously taking actions and observe the corresponding effects, and finally a well-trained model is obtained through the accumulated experience. However, the difference is that each action in reinforcement learning does not have a labeled value as supervision. The environment only gives a reward to current action, which is always lagged i.e. the consequence of the current action cannot be fully reflected only after a period of time.

2.4.1 Learning process

Formally, RL can be described as a Markov decision process (MDP)[34]. The characteristic of Markov process is that the next state of the system is only related to the current state and is independent of earlier moments.
The state transition of RL process can be represent as a tuple:

$$(s_t, a_t, r_t, s_{t+1}) \tag{2.4}$$

where

- s_t is the state at time step t
- a_t is the action at time step t
- r_t is the environment reward at time step t
- s_{t+1} is the state at time step t

Its state transition probability is:

$$P(s_t, s_{t+1}) = P(s = s_{t+1} \mid s = s_t, a = a_t) \tag{2.5}$$

RL learning process is shown in the Figure 2.4. At current time step t, based on observed state s_t, the agent takes action a_t to interact with the environment. Then at next time step $t + 1$, the environment transitions to state s_{t+1} and feeds back reward r_t to the agent. This forms a closed loop, in which the agent continuously optimizes its own policy through reinforcement learning algorithm to maximize cumulative reward.

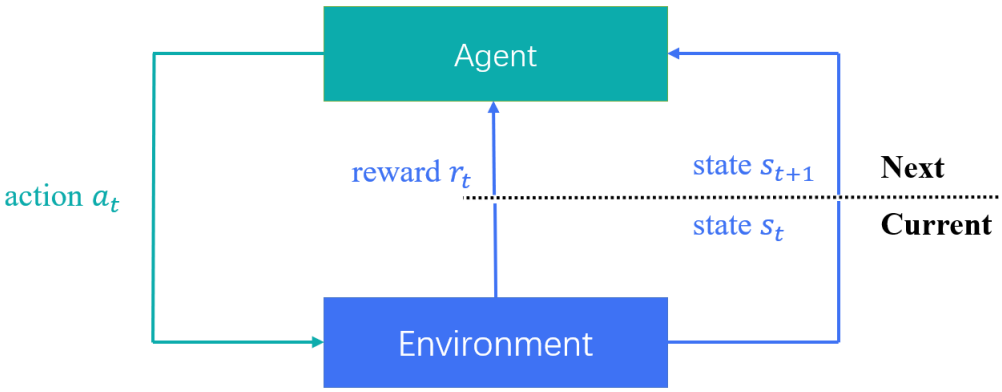

Figure 2.4. Reinforcement learning process

2.4.2 Key components

A reinforcement learning system can be divided into four key components: agent, environment, reward, and value function.

1) **Agent**
An agent is the entity that performs actions such as a car in autonomous driving scenario. It will act based on the policy function, which is generally classified as stochastic policy and deterministic policy.

- **Stochastic policy function** π defined by parameter θ can be expressed as:
$$a_t \sim \pi\left(\cdot \mid s_t, \theta\right) \tag{2.6}$$
at time step t, it takes state s_t as input and output the probability distribution of actions, which is used for sampling to obtain value of the action a_t.
- **Deterministic policy function** μ defined by parameter θ can be expressed as:
$$a_t = \mu\left(s_t \mid \theta\right) \tag{2.7}$$
at time step t, it takes state s_t as input and output the deterministic value of action a_t.

2) **Environment**
An environment is the entity that the agent can interacts with, which is capable of state generation and transition. Besides, an environment should have a set of rules to score the agent action, such as the Atari game.

3) **Reward**
Reward is a value returned by the environment after the agent performs an action. We can customize the calculation rules of rewards function to guide the agent learning direction in the desired way. In addition, the quality of reward function design normally affects learning performance.

4) **Value function**
Value function is the expectation of the cumulative reward in the future, which is used to assess the quality of the action. There are two types of value functions: state and action value functions.

- **Cumulative reward** can be expressed as:
$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2.8}$$
with discount rate $\gamma \in (0.1]$.
- **State value function** can be expressed as:
$$V(s_t) = \mathbb{E}\left\{R_t \mid s_t, \pi\right\} \tag{2.9}$$

which means the expected return of cumulative reward in state s_t by following policy π.

- **Action value function** can be expressed as:

$$Q(s_t, a_t) = \mathbb{E}\left\{R_t \mid s_t, a_t, \pi\right\} \tag{2.10}$$

which means the expected return of cumulative reward for selecting action a_t in state s_t by following policy π.

2.4.3 Classification

- **Model-based and model-free method**
In **model-based** methods, the agent will directly utilize the properties of the environment including state transition function and reward function to solve the problem. The main advantage is that the agent can predict future states and rewards through the environment model, which leads to better trajectory plan. Typical model-based algorithms are MBMF algorithm[35], AlphaGo algorithm[36] and etc. The disadvantage of model-based methods is that assumptions in which the model is constructed are usually too strong. Dynamic models in practical problems can be so complex that they cannot even be represented explicitly, making models often unavailable. On the other hand, in real-world applications, the constructed models are often inaccurate, which introduces estimated errors in training process. Policy optimization based on the model with error often causes failure in real environment.
Model-free methods do not attempt to figure out the internal principles of the environment, but directly explore the optimal policy. Taking Q-learning algorithm[37] as example, the agent directly estimates the action value $Q(s, a)$ of the state-action pair, then selects the action corresponding to the maximum Q value and uses the feedback reward to update the Q value function. As the Q value converges, the policy gradually converges to reach optimal. The feature of this type of method is that the agent does not focus on the model, but directly searches for the policy that can maximize the value, which is relatively easier to realize and train respect to model-based methods. However, the disadvantages of model-free methods are also obvious. Due to the high randomness of action selection in the early exploration stage, if this method is directly deployed in a real world to learn, the agent may cause serious damage to the environment as well as security issues.

- **Value-based and policy-based method**
Value-based methods usually optimize the action value function $Q(s_t, a_t)$. The explored optimal value function can be expressed as $Q^* = \max_\pi Q\left(s_t, a_t\right)$. Value based methods can usually learn from samples for more efficiently and

their estimation error of the value equation is smaller. In the process of iterative update, it is not easy to fall into the local optimum and results in the failure of convergence. However, since value-based methods can usually only deal with discrete action scenarios, they cannot be directly applied to the simulation environment in this thesis.
Policy-based methods iteratively update the policy by estimating the policy gradient $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a)]$[38] to maximize the cumulative reward. Compared with value-based methods, policy-based methods converge faster and are suitable for continuous action.
Besides value-based and policy-based methods, **Actor-Critic** methods, the combination of above two are usually used more widely. The Actor-Critic methods benefit from the advantages of both methods, using the value-based method(Critic) to improve the efficiency of experience sampling and using the policy-based method(Actor) to deal with continuous or high-dimensional spaces.

- **On-policy and off-policy method**
  On-policy methods require that the behavior policy must be aligned with the target policy. However, **off-policy** method don't have this constraint. An off-policy agent can use the experience collected from others to improve its own policy. The behavior policy refers to the strategy that controls the interaction between the agent and the environment, whose role is to collect experience by observing environment states, actions, and rewards. The target policy refers to the training purpose i.e. the agent needs to select actions based on this policy. The most obvious property of off-policy methods is that they can use the behavior policy to collect experience and save the tuple (s_t, a_t, r_t, s_{t+1}) into a buffer, the agent then can replay these experience to update the target policy.

2.5 Deep reinforcement learning

Deep reinforcement learning(DRL) refers to reinforcement learning algorithms where neural networks are used to estimate value and policy functions.
In recent years, there has been an explosion of deep learning algorithms based on neural networks. With the development of big data technology, improved hardware performance, and the emergence of new algorithms, it is now possible to train models using raw inputs such as images, text, etc., in a manner similar to humans. In the wave of artificial intelligence, a revolution is taking place that uses neural networks to replace specific models to extract features. The DQN network proposed by DeepMind[37] can be seen as a good example of integrating deep learning into reinforcement learning, where the key is to estimate the action

value function with a neural network. This algorithm is a model-free method that does not require prior knowledge about the environment, but instead learns from collected raw environmental data. With deep reinforcement learning, robots can become more intelligent without the need for humans to impart specialized domain knowledge. The developments taking place in this field are exciting.[33].

2.5.1 Deep learning basics

1) **Neural network**
The basic unit of neural network is called perceptron, shown in the Figure 2.5.

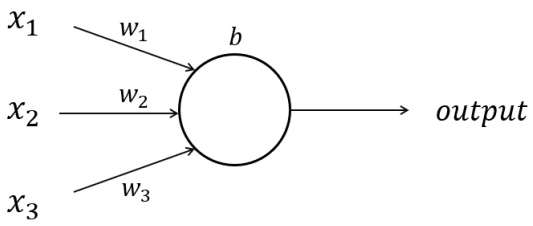

Figure 2.5. Perceptron unit

It take multiple inputs to generate output, which can be expressed as:

$$\begin{aligned} \text{output} &= \sigma(\text{ input }) \\ &= \sigma\left(\sum_i w_i x_i + b\right) \end{aligned} \tag{2.11}$$

where

- $\sigma()$ refers to activation function
- x_i refers to input i
- w_i refers to weight of x_i
- b refers to bias

By connecting perceptrons to each other we can get a neural network, show in Figure2.6. The input layer in the network is in the first column, the output layer is the last column, and the columns of perceptrons between them cannot be accessed from the outside of neural network, so it is called the hidden layer.

Those parameters related to the neural network settings are called hyperparameters, including width that refers to how many neurons per layer and the

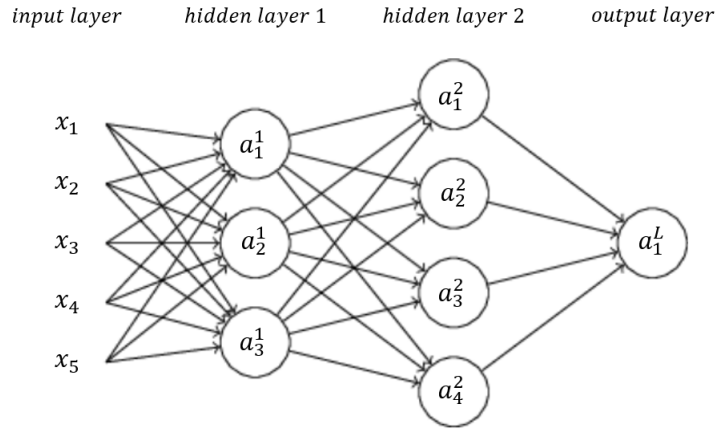

Figure 2.6. Neural network sketch

depth that refers to how many layers there are in network. Besides, there are other hyperparameters related to the training configuration, such as batch size, learning rate, L2 penalty and etc. These parameters have a large impact on model's training performance and usually require fine-tuning to achieve desirable results.

2) **Back-propagation method**
The difference between the output of a well-trained neural network and the true value should be as small as possible. This difference is called the loss, which can be used to indicate the accuracy of the neural network's predictions. There are several types of loss functions, taking the mean square error(MSE) loss function as an example, the expression is:

$$\begin{aligned} C &= \frac{1}{N}\sum_n \left\|a^L - y\right\|^2 \\ &= \frac{1}{N}\sum_n \sum_j (a_j^L - y_j)^2 \end{aligned} \tag{2.12}$$

where

- N is the total number of inputs
- $a^L = a^L(\mathbf{x}^{(n)})$ is the output vector corresponding to the input $\mathbf{x}^{(n)}$, $a^L = [a_1^L, ...a_j^L, ...]^T$
- $y = y(\mathbf{x}^{(n)})$ is the true value vector corresponding to the input $\mathbf{x}^{(n)}$, $y = [y_1, ...y_j, ...]^T$

- $\mathbf{x}^{(n)}$ is the n_{th} input vector, $\mathbf{x}^{(n)} = [x_1^{(n)}, ...x_i^{(n)}, ...]^T$
- a_j^L is the output of the j_{th} perceptron in output layer L
- y_j is the true value corresponding to the j_{th} perceptron

After we obtain the loss C, we can update the network parameters θ (mainly including the weights and bias) based on the back-propagation method and the optimization algorithm, and finally minimize the loss after iterations. The purpose of the **back-propagation** method is to calculate the gradient of the loss w.r.t the network parameters, which can be expressed as:

$$g = \nabla_\theta C \tag{2.13}$$

First, let's formally define the parameters in the neural network, as shown in Figure 2.7[39] where

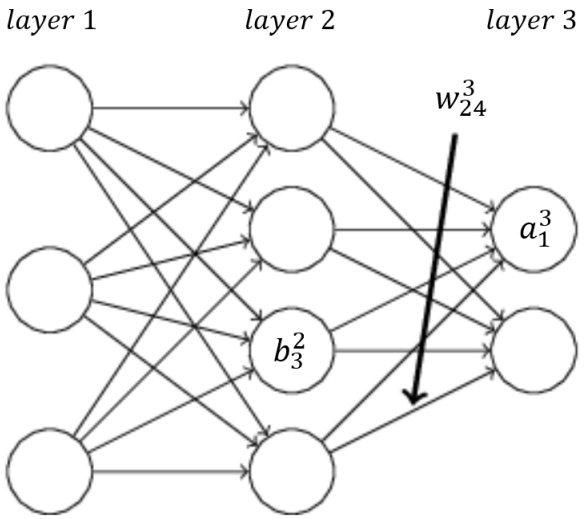

Figure 2.7. Neural network

- w_{jk}^l is the weight of connection from the k_{th} perceptron in layer $l-1$ to the j_{th} perceptron in layer l
- b_j^l is the bias of the j_{th} perceptron in layer l
- z_j^l is the input of the j_{th} perceptron in layer l, $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$
- a_j^l is the output of the j_{th} perceptron in layer l, $a_j^l = \sigma(z_j^l)$

The whole process of back-propagation method can be divided into four steps:

- **BP1** Calculate the gradient of the loss C w.r.t the input z_j^L of the output layer:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right) \tag{2.14}$$

  The equation can be rewritten in a matrix form:

$$\delta^L = \frac{\partial C}{\partial a^L} \bigodot \frac{\partial a^L}{\partial z^L} = \nabla_a C \odot \sigma'\left(z^L\right) \tag{2.15}$$

  where $\odot$ refers to element-wise product.

- **BP2** Back propagate and calculate the gradient of the loss C w.r.t the input z_j^l of each layer:

$$\begin{aligned}
\delta_j^l = \frac{\partial C}{\partial z_j^l} &= \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} \\
&= \sum_k \delta_k^{l+1}\frac{\partial\left(\sum_x w_{kx}^{l+1}a_x^l + b_k^{l+1}\right)}{\partial a_j^l}\sigma'\left(z_j^l\right) \\
&= \sum_k \delta_k^{l+1}\frac{\partial\left(w_{kj}^{l+1}a_j^l\right)}{\partial a_j^l}\sigma'\left(z_j^l\right) \\
&= \sum_k \delta_k^{l+1} w_{kj}^{l+1}\sigma'\left(z_j^l\right)
\end{aligned} \tag{2.16}$$

  The equation can be rewritten in a matrix form:

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right) \tag{2.17}$$

- **BP3** Calculate the gradient of the loss C w.r.t the weight w_{jk}^l:

$$\begin{aligned}
\frac{\partial C}{\partial w_{jk}^l} &= \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} \\
&= \delta_j^l \frac{\partial\left(\sum_x w_{jx}^l a_x^{l-1} + b_j^l\right)}{\partial w_{jk}^l} \\
&= \delta_j^l \frac{\partial\left(w_{jk}^l a_k^{l-1}\right)}{\partial w_{jk}^l} \\
&= \delta_j^l a_k^{l-1}
\end{aligned} \tag{2.18}$$

- **BP4** Calculate the gradient of the loss C w.r.t the bias b_j^l:

$$\begin{aligned}\frac{\partial C}{\partial b_j^l} &= \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} \\ &= \delta_j^l \frac{\partial \left(\sum_x w_{jx}^l a_x^{l-1} + b_j^l\right)}{\partial b_j^l} \\ &= \delta_j^l \frac{\partial b_j^l}{\partial b_j^l} \\ &= \delta_j^l\end{aligned} \tag{2.19}$$

3) **Optimization algorithm**
After gradient $g = \nabla_\theta C$ is obtained, we can choose one type of optimization algorithm to update network parameters θ. For example, the gradient descent method, which is expressed as:

$$\theta_t = \theta_{t-1} - \eta g_t \tag{2.20}$$

where

- t refers to current time step
- η refers to learning rate

However, there are some problems with the gradient descent method. For example, the learning rate is difficult to determine. Optimal may never be reached with high learning rate and the convergence will become slowly with low learning rate. To solve this problem, an adaptive learning rate algorithm can be used to speed up the convergence. The principle of these algorithms is that when a small gradient is obtained, the algorithm will increase the learning rate; on the contrary, if the gradient is too large, the algorithm will decrease the learning rate automatically.
Adam[40] is the most common adaptive learning rate algorithm and its key step can be expressed as:

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t \tag{2.21}$$

where

- $\hat{m}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t}$
- $\hat{v}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t}$
- m_t is the first moment calculated by moving average method: $m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t$

- v_t is the second moment calculated by moving average method: $v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2$
- β_1, β_2, α are predefined parameters
- β_1^t refers to β_1 to the power of t
- β_2^t refers to β_2 to the power of t
- ε is set to 10^{-8}

4) **Regularization**

A machine learning model optimized to reduce the loss on train set can not guarantee that it will also perform well on test set. An over-optimized model will have a small loss in train set but a large loss in test set, this phenomenon is known as overfitting. We need to use regularization methods to make a model perform well in both train set and test set. Common regularization methods in deep learning include dropout, weight decay and so on.

The basic idea of **dropout** is to randomly delete a certain proportion of perceptrons in each iteration and only train the network composed of the remaining perceptrons, which reduces the correlation between perceptrons and parameter update no longer depends on the common effect of nodes. This is equivalent to training multiple neural networks in parallel and averaging their outputs, so that the overfitting of different networks cancels each other out, reducing the overall overfitting phenomenon and increasing the model robustness, which similar to bagging methods in ensemble learning.

Weight decay method is another regularization method used to reduce overfitting. It adds an extra term to the loss function, so that the network weights w will have smaller absolute values after optimization. In a network with small weights, when the input is slightly perturbed, the output will not change drastically, which objectively filters out noise and reduces overfitting. The L2 regularization is the type of weight decay method usually used to reduce the overfitting phenomenon, which adds one term $\frac{\lambda}{2N}\|w\|^2$ to construct a new loss function:

$$C_{\text{new}} = C + \frac{\lambda}{2N}\|w\|^2 \tag{2.22}$$

where

- λ refers to the regularization parameter
- N is the inputs size

The gradient of the new loss C_{new} w.r.t the weight:

$$\begin{aligned} \frac{\partial C_{new}}{\partial w_{jk}^l} &= \frac{\partial C}{\partial w_{jk}^l} + \frac{\lambda}{2N}\frac{\partial \|w\|^2}{\partial w_{jk}^l} \\ &= \delta_j^l a_k^{l-1} + \frac{\lambda}{N} w_{jk}^l \end{aligned} \tag{2.23}$$

The new weight iteration equation is:

$$\begin{aligned} w_{jk}^{l} &= w_{jk}^{l} - \eta\left(\delta_{j}^{l} a_{k}^{l-1} + \frac{\lambda}{N} w_{jk}^{l}\right) \\ &= \left(1 - \frac{\eta\lambda}{N}\right) w_{jk}^{l} - \eta\delta_{j}^{l} a_{k}^{l-1} \end{aligned} \tag{2.24}$$

where

- $\left(1 - \frac{\eta\lambda}{N}\right)$ refers to weight decay with absolute value smaller than 1

The equation2.24 shows that weight will be re-scaled by weight decay in each iteration making its absolute value close to 0.

2.5.2 DRL algorithms

The deep reinforcement learning algorithms we use in this thesis are Deep Deterministic Policy Gradient (DDPG)[41] and Twin Delayed DDPG (TD3)[42] algorithms which are both deterministic policy gradient, model-free, actor-critic and off-policy based algorithms for continuous action.

1) **Deep Deterministic Policy Gradient**(DDPG)
The learning framework of DDPG algorithm is show in Figure 2.8

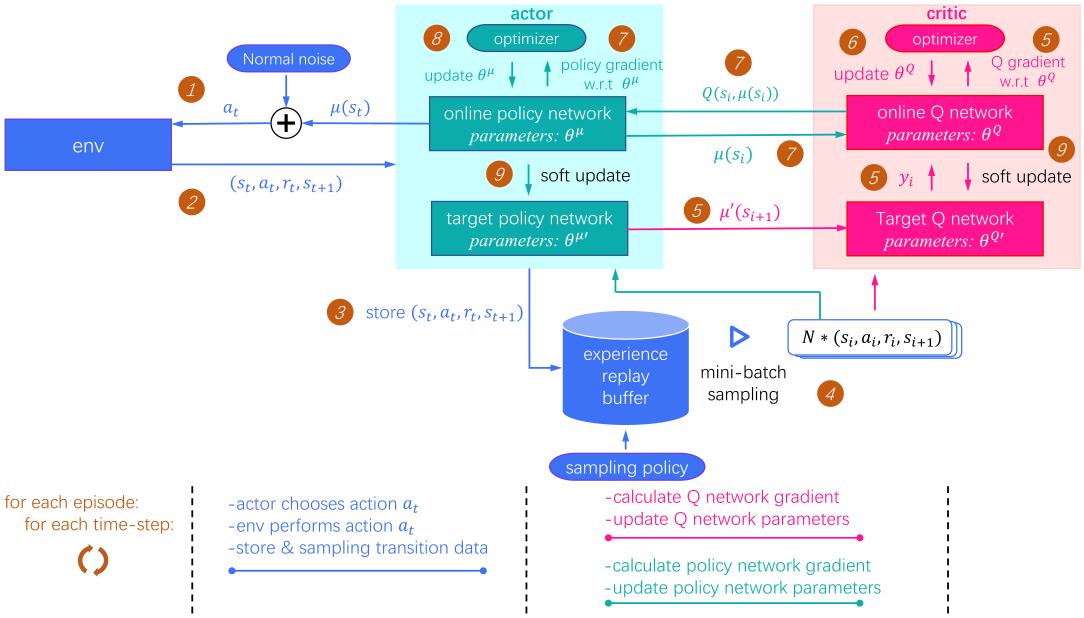

Figure 2.8. Deep Deterministic Policy Gradient framework

Description of the DDPG algorithm:

- Initialize the actor & critic online neural network parameters: θ^{μ}, θ^{Q};
- Copy parameters to corresponding target networks: $\theta^{\mu'} = \theta^{\mu}$, $\theta^{Q\prime} = \theta^{Q}$;
- Initialize the experience replay buffer R;
- **for** each episode:
  - **for** each step t:
    (The following steps correspond to the numbers in learning framework Figure 2.8)
    1. The actor chooses an action a_t with noise $\mathcal{N}_t$:

$$a_t = \mu\left(s_t \mid \theta^{\mu}\right) + \mathcal{N}_t \tag{2.25}$$

    2. The environment executes action a_t, then observes reward r_t and generates new state s_{t+1};
    3. Store transition (s_t, a_t, r_t, s_{t+1}) in buffer R as training dataset;
    4. Randomly sample a batch of N transitions (s_i, a_i, r_i, s_{i+1}) from buffer R;
    5. Calculate the gradient of the online Q network:
       We use mean squared error (MSE) as Q network loss function L:

$$L = \frac{1}{N}\sum_{i}\left(y_i - Q\left(s_i, a_i \mid \theta^{Q}\right)\right)^2 \tag{2.26}$$

       set

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right) \tag{2.27}$$

       Then base on back-propagation method, we can calculate the gradient of L w.r.t θ^{Q}:

$$\nabla_{\theta^{Q}} L \tag{2.28}$$

    6. Use Adam optimizer to update θ^{Q};
    7. Calculate the gradient of the online policy network:
       We use sampled mean value to estimate the policy network loss function J:

$$J \approx -\frac{1}{N}\sum_{i} Q\left(s_i, \mu\left(s_i \mid \theta^{\mu}\right) \mid \theta^{Q}\right) \tag{2.29}$$

       Then base on back-propagation method, we can calculate the gradient of J w.r.t θ^{μ}:

$$\nabla_{\theta^{\mu}} J \tag{2.30}$$

    8. Use Adam optimizer to update θ^{μ};

9. Soft update parameters of target networks:

$$\begin{cases} \theta^{\mu'} = \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \\ \theta^{Q'} = \tau\theta^{Q} + (1-\tau)\theta^{Q'} \end{cases} \tag{2.31}$$

- **end for**

- **end for**

2) **Twin Delayed DDPG**(TD3)
The learning framework of TD3 algorithm is show in Figure 2.9

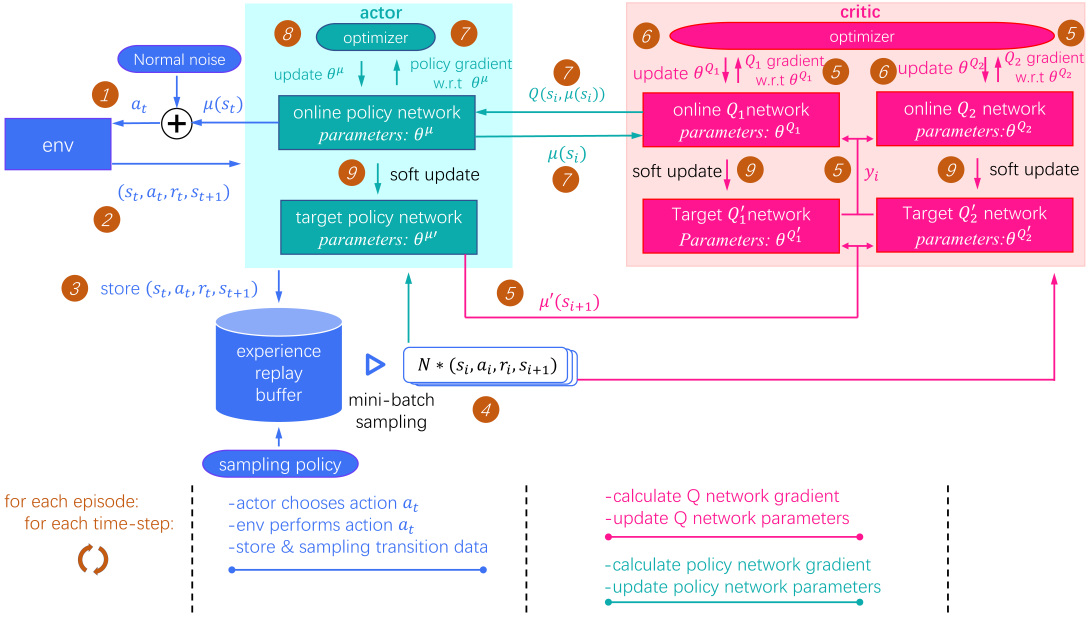

Figure 2.9. Twin Delayed DDPG framework

Description of the TD3 algorithm:

- Initialize actor online neural network parameters θ^{μ} and critic online neural network parameters θ^{Q_1}, θ^{Q_2};
- Copy parameters to corresponding target networks: $\theta^{\mu'} = \theta^{\mu}$, $\theta^{Q_1'} = \theta^{Q_1}$, $\theta^{Q_2'} = \theta^{Q_2}$;
- Initialize the experience replay buffer R;
- **for** each episode:
  - **for** each step t:
    (The following steps correspond to the numbers in learning framework Figure 2.9)

1. The actor chooses an action a_t with noise $\mathcal{N}_t$:

$$a_t = \mu\left(s_t \mid \theta^{\mu}\right) + \mathcal{N}_t \tag{2.32}$$

2. The environment executes action a_t, then observes reward r_t and generates new state s_{t+1};
3. Store transition (s_t, a_t, r_t, s_{t+1}) in buffer R as training dataset;
4. Randomly sample a batch of N transitions (s_i, a_i, r_i, s_{i+1}) from buffer R;
5. Calculate the gradient of the online Q_1, Q_2 network:
   We use mean squared error (MSE) as Q_1, Q_2 network loss function:

$$L_1 = \frac{1}{N}\sum_i \left(y_i - Q_1\left(s_i, a_i \mid \theta^{Q_1}\right)\right)^2 \tag{2.33}$$

$$L_2 = \frac{1}{N}\sum_i \left(y_i - Q_1\left(s_i, a_i \mid \theta^{Q_2}\right)\right)^2 \tag{2.34}$$

   set

$$y_i = r_i + \gamma \min\left\{Q_1'\left(s_{i+1}, \tilde{a}_i \mid \theta^{Q_1'}\right), Q_2'\left(s_{i+1}, \tilde{a}_i \mid \theta^{Q_2'}\right)\right\} \tag{2.35}$$

$$\tilde{a}_i = \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) + \mathrm{clip}(\epsilon, -c, c), \quad \epsilon \sim \mathcal{N}(0, \sigma) \tag{2.36}$$

   Then base on back-propagation method, we can calculate the gradient of L_1 and L_2 w.r.t θ^{Q_1} and θ^{Q_2}:

$$\nabla_{\theta^{Q_1}} L_1, \nabla_{\theta^{Q_2}} L_2 \tag{2.37}$$

6. Use Adam optimizer to update θ^{Q_1}, θ^{Q_2};

- **if** step t mod delay-frequency $= 0$, **then**

7. Calculate the gradient of the online policy network:
   We use sampled mean value to estimate the policy network loss function J:

$$J \approx -\frac{1}{N}\sum_i Q_1\left(s_i, \mu\left(s_i \mid \theta^{\mu}\right) \mid \theta^{Q_1}\right) \tag{2.38}$$

   Then base on back-propagation method, we can calculate the gradient of J w.r.t θ^{μ}:

$$\nabla_{\theta^{\mu}} J \tag{2.39}$$

8. Use Adam optimizer to update θ^{μ};
9. Soft update parameters of target networks:

$$\begin{cases} \theta^{\mu\prime} = \tau\theta^{\mu} + (1-\tau)\theta^{\mu\prime} \\ \theta^{Q_1'} = \tau\theta^{Q_1} + (1-\tau)\theta^{Q_1'} \\ \theta^{Q_2'} = \tau\theta^{Q_2} + (1-\tau)\theta^{Q_2'} \end{cases} \tag{2.40}$$

- **end if**
- **end for**
- **end for**

2.5.3 Prioritized replay

The feature of off-policy algorithms is that they can learn from experience buffer. The process of sampling from the buffer is called experience replay. Prioritized experience replay (PER)[43] is a method that works better than random experience replay for its faster convergence and better test performance. The main idea of PER is to sample according to experience importance, data with higher importance are easier to be replayed. PER uses temporal difference (TD) error to measure the importance of each sample. In this thesis, we use PER to improve DDPG algorithm.

Description of the DDPG with PER algorithm:

- Initialize the actor & critic online neural network parameters: θ^{μ}, θ^{Q};
- Copy parameters to corresponding target networks: $\theta^{\mu'} = \theta^{\mu}$, $\theta^{Q'} = \theta^{Q}$;
- Initialize the experience replay buffer R;
- Set priority $D_1 = 1$, exponents α, β;
- **for** each episode:
  - **for** each step t:
    (The following steps correspond to the numbers in learning framework Figure 2.8)
    1) The actor chooses an action a_t with noise $\mathcal{N}_t$:

$$a_t = \mu\left(s_t \mid \theta^{\mu}\right) + \mathcal{N}_t \tag{2.41}$$

    2) The environment executes action a_t, then observes reward r_t and generates new state s_{t+1};
    3) Store transition (s_t, a_t, r_t, s_{t+1}) in buffer R as training dataset and set priority $D_t = \max\left\{D_j\right\}, j \in [1,2 \cdots t-1]$;
    4) Sample a minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from buffer R with **PER** algorithm:
       - **for** $i = 1$ **to** N:
         Sample transition (s_i, a_i, r_i, s_{i+1}) with probability $P(i) = \frac{D_i^{\alpha}}{\sum_j D_j^{\alpha}}$;
         Compute corresponding importance-sampling weight $W_i = \frac{1}{V^{\beta} P(i)^{\beta}}$;

where V is the number of samples in buffer R;
Compute TD-error

$$\delta_i = y_i - Q\left(s_i, a_i \mid \theta^Q\right) \tag{2.42}$$

set

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right) \tag{2.43}$$

Update the priority of transition i $D_i = |\delta_i|$;

- **end for**

Normalize importance-sampling weight $W_i = W_i / \max\{W_j\}, j \in [1,2 \cdots N]$;

5) Calculate the gradient of the online Q network:
We use mean squared error (MSE) as Q network loss function L:

$$L = \frac{1}{N} \sum_i W_i \delta_i^2 \tag{2.44}$$

Then base on back-propagation method, we can calculate the gradient of L w.r.t θ^Q:

$$\nabla_{\theta^Q} L \tag{2.45}$$

6) Use Adam optimizer to update θ^Q;
7) Calculate the gradient of the online policy network:
We use sampled mean value to estimate the policy network loss function J:

$$J \approx -\frac{1}{N} \sum_i Q\left(s_i, \mu\left(s_i \mid \theta^\mu\right) \mid \theta^Q\right) \tag{2.46}$$

Then base on back-propagation method, we can calculate the gradient of J w.r.t θ^μ:

$$\nabla_{\theta^\mu} J \tag{2.47}$$

8) Use Adam optimizer to update θ^μ;
9) Soft update parameters of target networks:

$$\begin{cases} \theta^{\mu\prime} = \tau\theta^\mu + (1-\tau)\theta^{\mu\prime} \\ \theta^{Q\prime} = \tau\theta^Q + (1-\tau)\theta^{Q\prime} \end{cases} \tag{2.48}$$

- **end for**

- **end for**

Chapter 3

Methodology

3.1 State & action definition

1) **State**
The observation state s_t is an one-dimensional array and its elements are listed in Table 3.1.

Elements	**Description**
ego vehicle velocity v_{ego}	
relative distance ΔD	$\Delta D = Pos_{lead} - Pos_{ego}$
relative velocity Δv	$\Delta v = v_{lead} - v_{ego}$
longitudinal slip ratio of front left wheel $(S_l)_{FL}$	
longitudinal slip ratio of front right wheel $(S_l)_{FR}$	$S_l = \frac{\Omega R_e}{V} - 1$
longitudinal slip ratio of rear left wheel $(S_l)_{RL}$	
longitudinal slip ratio of rear right wheel $(S_l)_{RR}$	
friction coefficient of front left wheel μ_{FL}	
friction coefficient of front right wheel μ_{FR}	
friction coefficient of rear left wheel μ_{RL}	
friction coefficient of rear right wheel μ_{RR}	

Table 3.1. Observation state s_t

2) **Action**
The action a_t is a value of acceleration or deceleration.

3.2 Reward functions design

The reward consists of four parts, and $(\omega_1, \omega_2, \omega_3, \omega_4)$ are their relative weights:

$$\begin{aligned} r_{tot} =& \omega_1 r_1\left(\Delta D, v_{lead}\right) + \omega_2 r_2(\Delta v) + \\ & \omega_3 r_3(Jerk) + \omega_4 r_4\left(\left(S_l\right)_{FL}, \left(S_l\right)_{FR}, \left(S_l\right)_{RL}, \left(S_l\right)_{RR}\right) \end{aligned} \tag{3.1}$$

1) Distance reward

$$r_1\left(\Delta D, v_{lead}\right) = \begin{cases} -1 & \Delta D < 0.6 v_{lead} \\ \frac{1.6}{(0.9 * v_{lead})^2} * \left(\Delta D - 0.6 * v_{lead}\right)^2 - 1 & 0.6 v_{lead} \leq \Delta D < 1.5 v_{lead} \\ \frac{-0.4}{(1.5 * v_{lead})^2} * \left(\Delta D - 3 * v_{lead}\right)^2 + 1 & 1.5 v_{lead} \leq \Delta D < 4.5 v_{lead} \\ \frac{1.6}{(0.9 * v_{lead})^2} * \left(\Delta D - 5.4 * v_{\text{lead }}\right)^2 - 1 & 4.5 v_{lead} \leq \Delta D < 5.4 v_{lead} \\ -1 & \Delta D \geq 5.4 v_{lead} \end{cases} \tag{3.2}$$

2) Speed reward

$$r_2(\Delta v) = 2e^{-|\Delta v|} - 1 \tag{3.3}$$

3) Comfort reward

$$r_3(Jerk) = 1 - 2 * \text{ Sfuzzyfunction } (Jerk, a, b), \tag{3.4}$$

where

- $Jerk = \frac{\Delta acceleration}{\Delta t}$
- $a = 0.6$
- $b = 2$
- Sfuzzyfunction is S-like shape function begins to climb from 0 when $Jerk = a$, and levels off at 1 when $Jerk = b$.

4) Stability reward

$$r_4\left(\left(S_l\right)_i\right) = \min\left\{2.0098 * \tanh\left(-3 * \left(S_l\right)_i\right) + 1\right\} \tag{3.5}$$

with $i \in \{FL, FR, RL, RR\}$

These four reward functions are shown in Figure 3.1.
The weights of reward components $(\omega_1, \omega_2, \omega_3, \omega_4)$ can be customized according to human demand to change the priority performance that the agent wants to maintain when controlling the vehicle. Currently, weights are set to be $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{6}\right)$. So that, the RL agent can be trained to prioritize the maintenance of safe speed and distance to prevent collision respect to the comfort and stability.

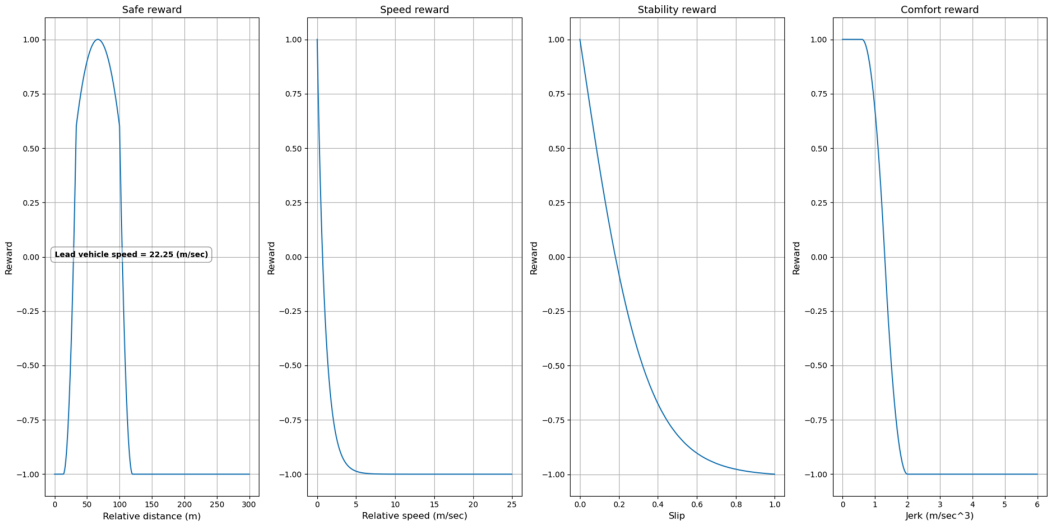

Figure 3.1. Reward components

3.3 CoMoVe framework

The CoMoVe framework shown in Figure 3.2[3] follows the micro-service design pattern and divides the entire system into four modules: vehicle mobility, vehicle dynamics, vehicle communication and vehicle control module.

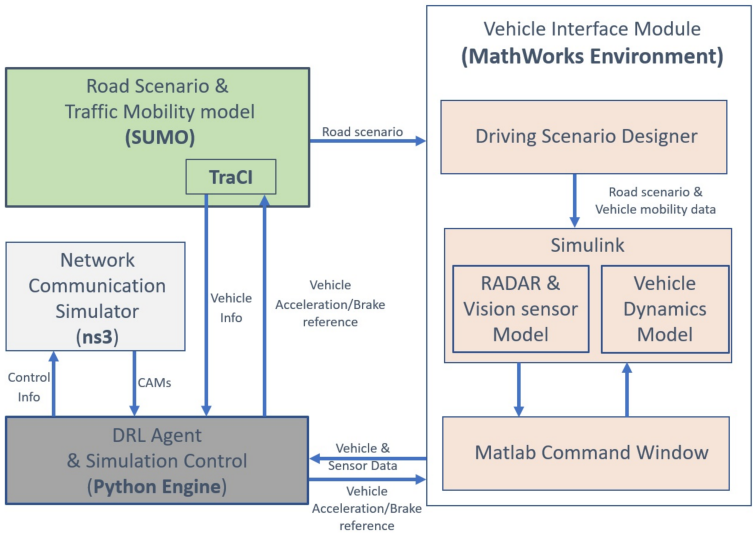

Figure 3.2. CoMoVe framework

The details are as follows:

- Vehicle mobility module:use SUMO simulator to design road scenarios, simulate car motion and visualize the vehicle driving.
- Vehicle dynamics module: use MATLAB/Simulink to simulate vehicle dynamics model and on-board sensors.
- Communication module: use ns-3 simulator to simulate the cellular network based V2X communication.
- Vehicle control module: use pytorch to build deep neural networks and use python to provide environment interfaces for DRL agent such as *step* (execute one time step within the environment), *reset* (reset the state of the environment to an initial state) and *close* (end the simulation process) functions based on the openAI gym standards to realize the model training and action selection.

Chapter 4

Performance evaluation

4.1 Simulation scenario

In this thesis, we focus on the longitudinal control of the DRL model in platooning scenario. In each episode, the total simulation duration is 25 seconds with a total number of 250 steps and an interval of 0.1 second between each step. We use CoMoVe framework to train the DRL model mainly in two scenarios:

- Two vehicles following scenario
- Three vehicles cut-out scenario

4.1.1 Two vehicles following scenario

In this scenario, the initial condition is shown in Figure 4.1:

- **Ego vehicle** (controlled by the DRL agent) departs at position = 5m with initial velocity = 22.5m/s;
- **Lead vehicle** departs at position = 50m with initial velocity = 22.5m/s.

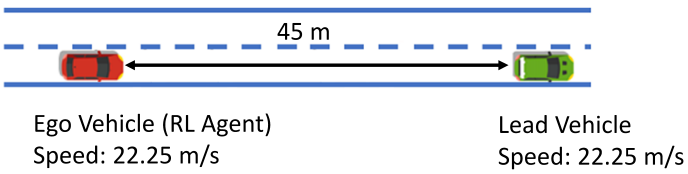

Figure 4.1. Two vehicles following scenario

During simulation, the variation of lead vehicle velocity is shown in Figure 4.2. The DRL agent learns how to control ego vehicle velocity by maximizing the accumulated rewards.

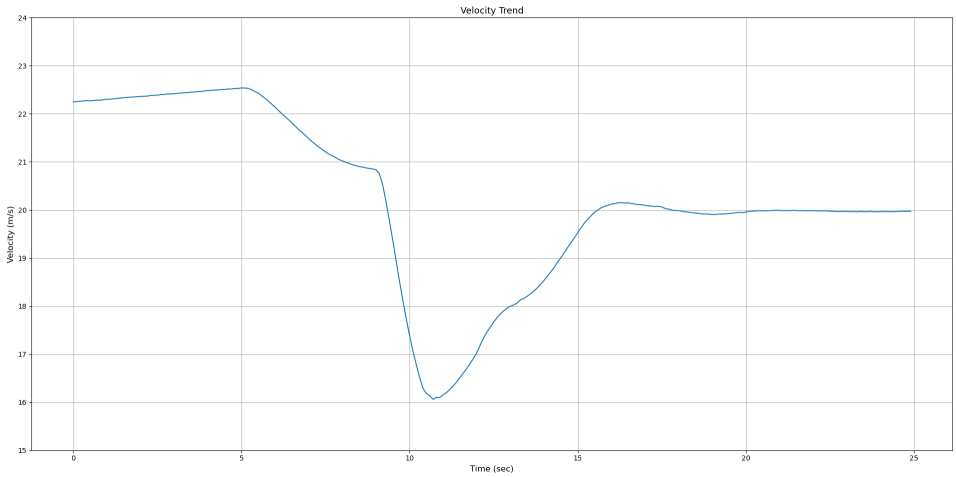

Figure 4.2. Lead vehicle velocity trend

4.1.2 Three vehicles cut-out scenario

In this scenario, the initial condition is shown in Figure 4.3:

- **Ego vehicle** (controlled by the DRL agent) departs at position = 5m with initial velocity = 18m/s;
- **Lead vehicle A** departs at position = 35m with constant velocity = 18m/s and changes line at position = 130m;
- **Lead vehicle B** departs at position = 130m with constant velocity = 12m/s.

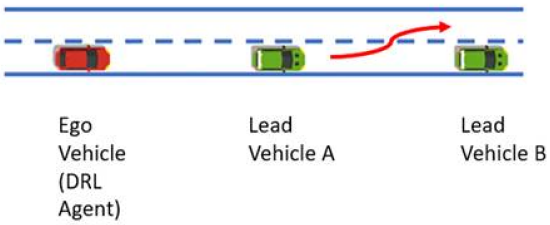

Figure 4.3. Three vehicles cut-out scenario

4.2 Experiments and results analytics

4.2.1 DDPG in two vehicles following scenario

We preliminarily train the DDPG algorithm with 200 episodes in two vehicles following scenario and test its performance. The hyperparameters of DDPG are listed in Table 4.1

Parameter	**Value**
optimization method	Adam
actor learning rate	0.0001
critic learning rate	0.001
discount factor γ	0.99
soft update factor τ	0.01
buffer size	10000
number of hidden layers	2
number of perceptron in each hidden layer	256
activation function	ReLU
sampling batch size	128
dropout probability	0
l2norm penalty	0

Table 4.1. DDPG hyperparameters

The training results are shown as followed:
Figure 4.4 reflects the DDPG agent training history and Figure 4.5 $\sim$ Figure 4.8 present specific performances of the well-trained model.

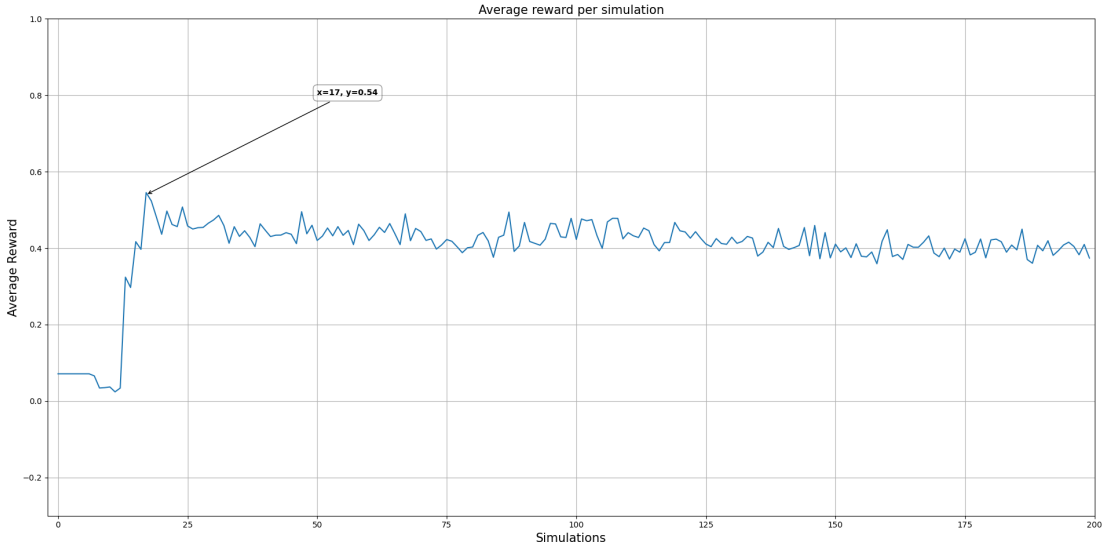

Figure 4.4. Average reward per simulation

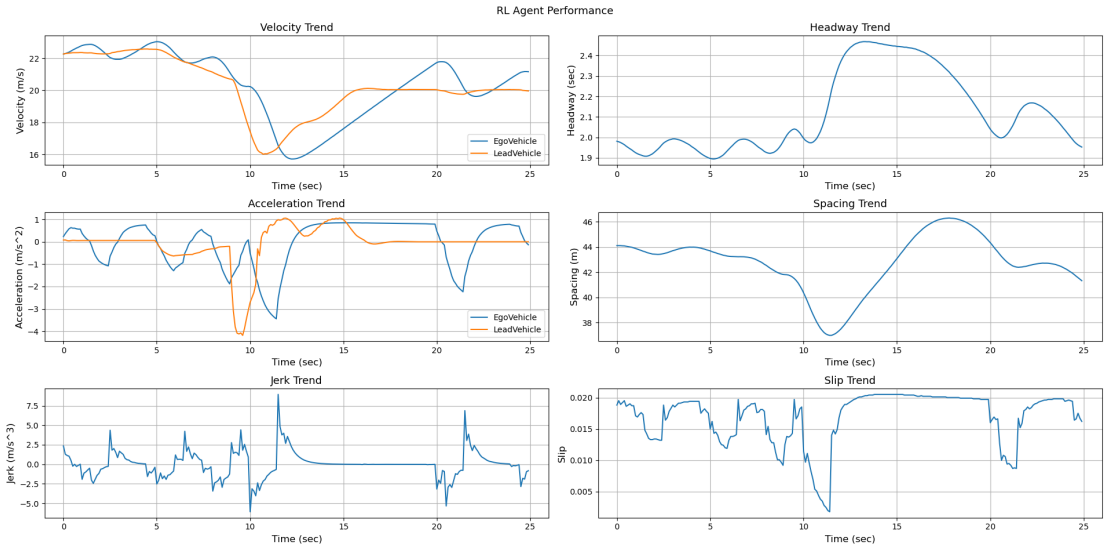

Figure 4.5. RL agent performance

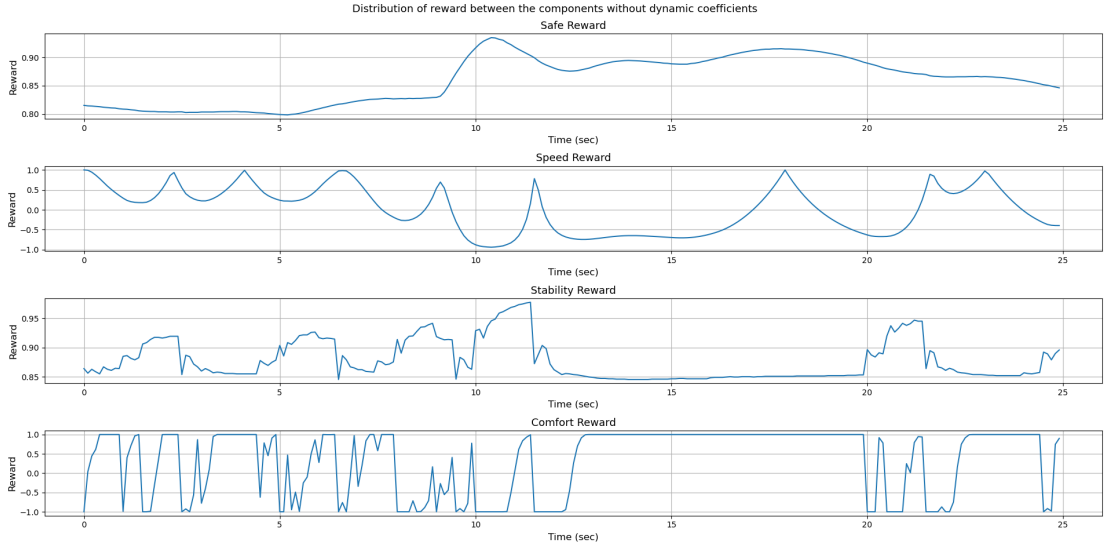

Figure 4.6. Reward components variation

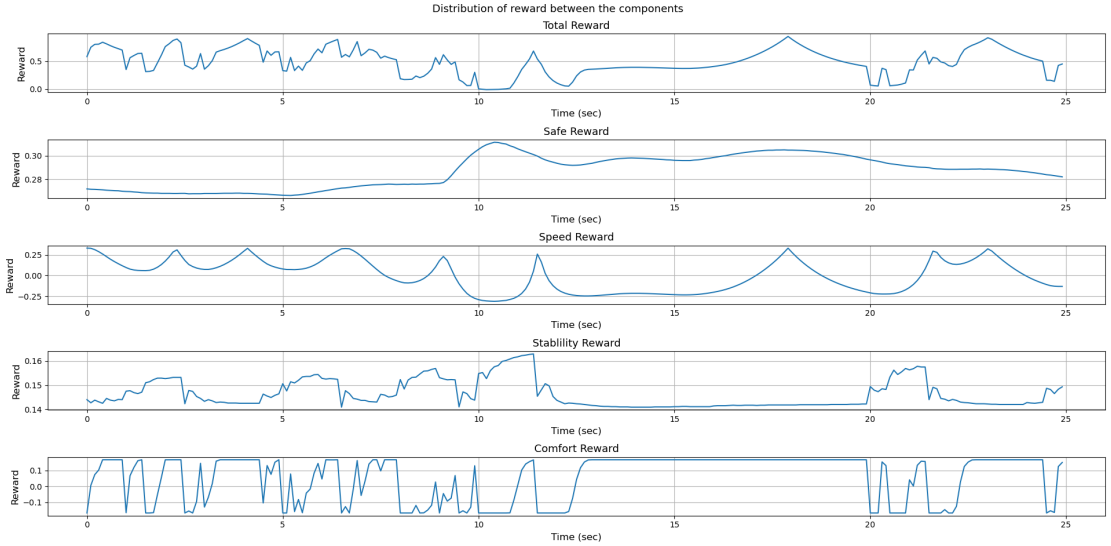

Figure 4.7. Reward components weighted variation

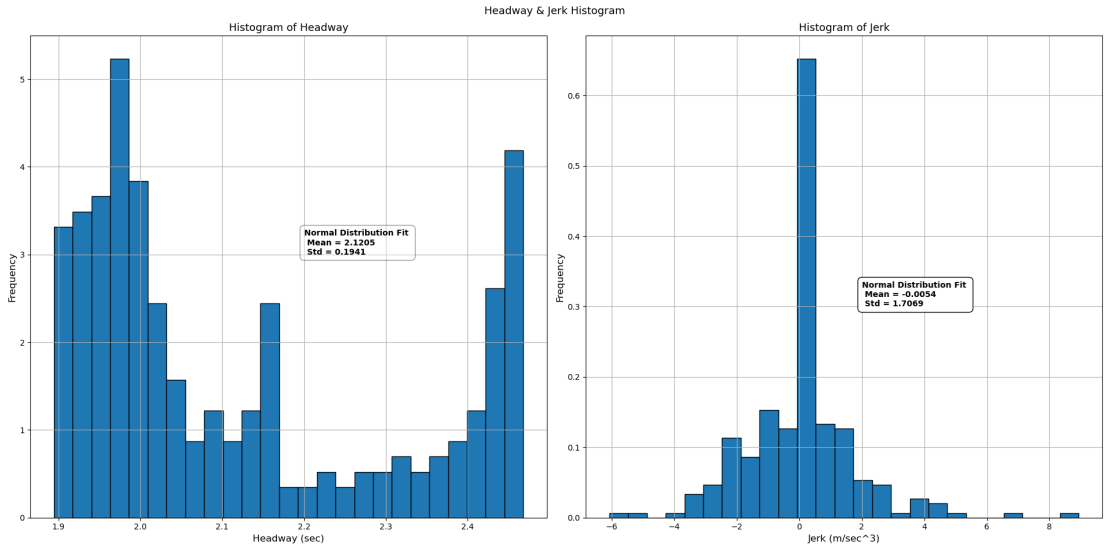

Figure 4.8. Distribution of Headway & Jerk

Figure 4.4 represents the variation of average step reward respect to simulation episodes. As it shows, after 25 episodes, the average reward of RL agent begins to stabilize at around 0.4, which is not a very high score(the average reward range is [-1,1]). This phenomenon indicates that the model is trapped in a local optimum during the training process and can no longer improve the cumulative reward value as training episode increases.
Figure 4.5 contains variations of several key performance indexes (KPIs) with respect to simulation time in one episode, which is generated from the DRL agent with the highest average reward. Based on the velocity, headway, acceleration and spacing trend information, it shows that, in most of the time, the ego vehicle can follow the lead vehicle at a similar speed and maintain the spacing above 40 meters. However, due to lead vehicle performs emergency braking in the interval [10,15]sec, the spacing is reduced below 40 meters. Referring to the jerk and slip trend information, we can see the reason is that jerk and longitudinal slip rate fluctuate drastically in this interval, and in order to maintain the passenger comfort and longitudinal stability, the RL agent sacrifices more spacing to smooth the deceleration. Adopting such control strategy demonstrates how the model maintains vehicle safety while considering comfort and stability.
Figure 4.6 shows the variation of four reward components: the safety and stability reward basically maintain at high level; the speed reward obtains some penalty in the interval [10,15]sec, which is mainly caused by the emergency braking of the

lead vehicle; the comfort reward was the most penalized one and has high level of oscillations, which mainly because of the low component weight in total reward, DRL agent doesn't prioritize maintaining comfort. Figure 4.7 shows their weighted variation.
Figure 4.8 presents the distribution of headway and jerk, which shows that the headway is relatively concentrated with small standard deviation, while jerk has some larger discrete values that makes its standard deviation higher.
To sum up, the preliminarily training results of the DDPG algorithm are not ideal and there is still a great potential for improvement. In **Sec** 4.2.2, various methods will be applied to improve the performance of DDPG algorithm.

4.2.2 Fine-tuning DDPG in two vehicles following scenario

In the previous section, we train the model using the DDPG algorithm in the two vehicles following scenario. The results show that there are some problems: the model falls into a local optimum during the training process, which results in low average step reward; the jerk has high level of oscillation, which causes negative impact on human comfort.
In this section, we will apply some methods to optimize the performance of the DDPG algorithm, including:

- fine-tune DDPG hyperparameters: batch size & dropout probability & L2 normalization penalty (weight decay);
- randomize initial state: the initial spacing and lead vehicle velocity;
- apply prioritized experience replay instead of sampling from buffer randomly.

- **Batch size & Dropout & L2 normalization**
  First, in each simulation, we only change the batch size to find the value that makes the model converge fastest. Then with optimal batch size, we separately adjust the dropout probability and l2norm penalty to compare the agent performance of each simulation.

a. **batch size** The value of batch size is chosen from list [1, 2, 4, 8, 16, 32, 64, 128, 256] and simulation results are as followed: Figure 4.9 reflects the DDPG agent training history with different batch size and Figure 4.10 ∼ Figure 4.11 present test performance of model with the batch size with highest learning rate.

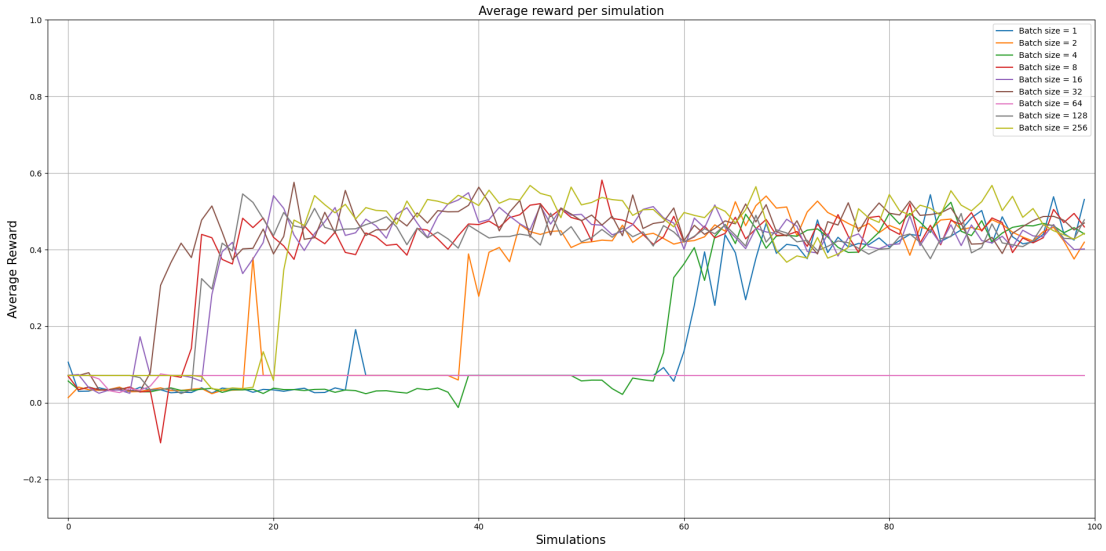

Figure 4.9. Average reward per simulation with different batch size

By comparing the learning rate of different batch size, Figure 4.9 shows that batch size = 32 is the best. Because its convergence speed is the fastest, and the number of sampling is relatively small in every step. So, batch size = 32 is used in all following simulations. Figure 4.10 and Figure 4.11 are the test results of batch size = 32, which show that jerk still has a relatively large oscillation.

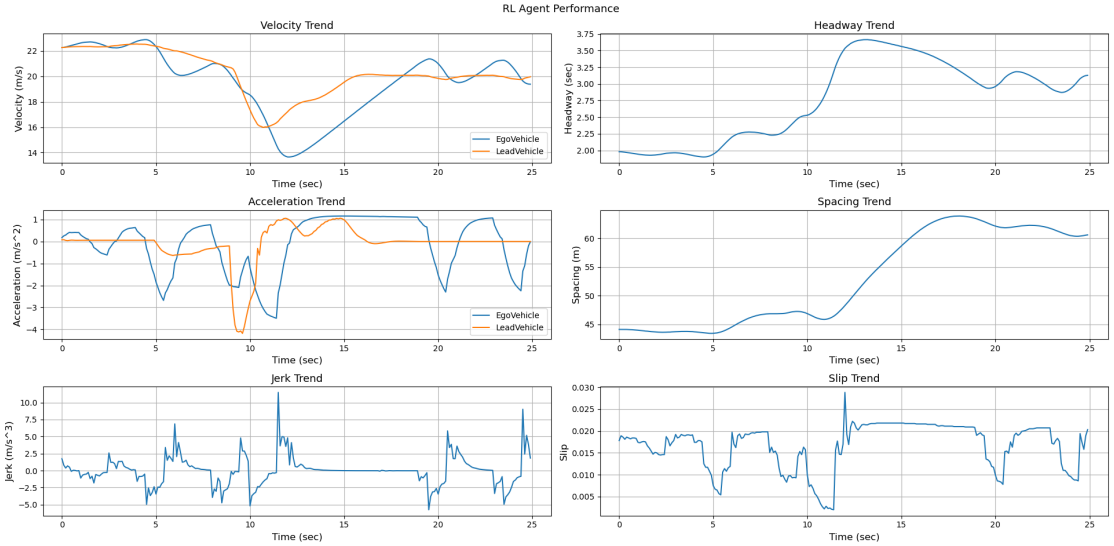

Figure 4.10. RL agent performance (batch size=32)

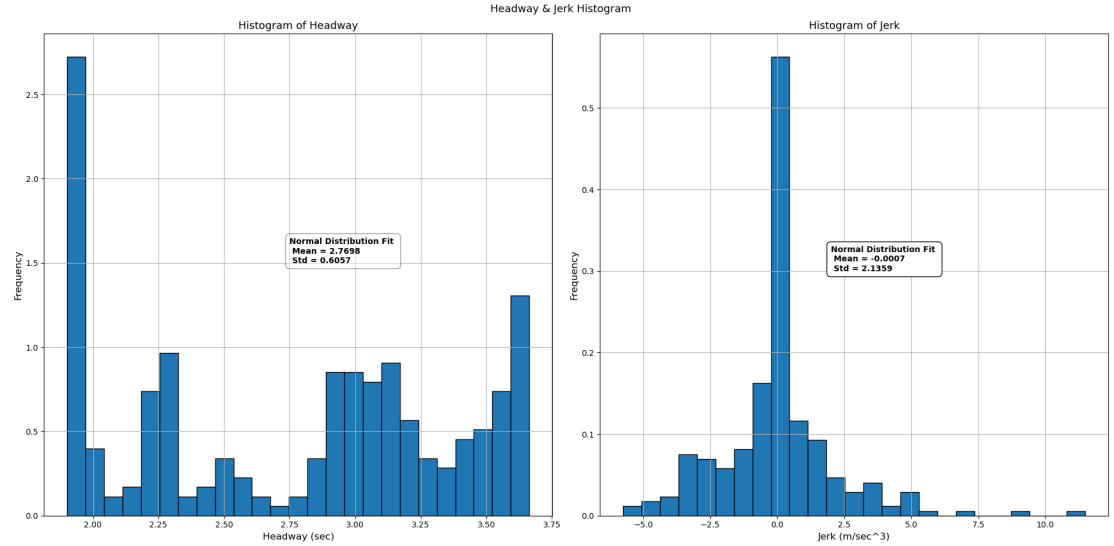

Figure 4.11. Distribution of Headway & Jerk (batch size=32)

b. **dropout**
Dropout probability is chosen from list [0, 0.1, 0.2, 0.3, 0.4, 0.5] and simulation results are as followed: Figure 4.12 reflects the DDPG agent training history with different dropout probability and 4.13 ∼ Figure 4.18 present relative test performances.

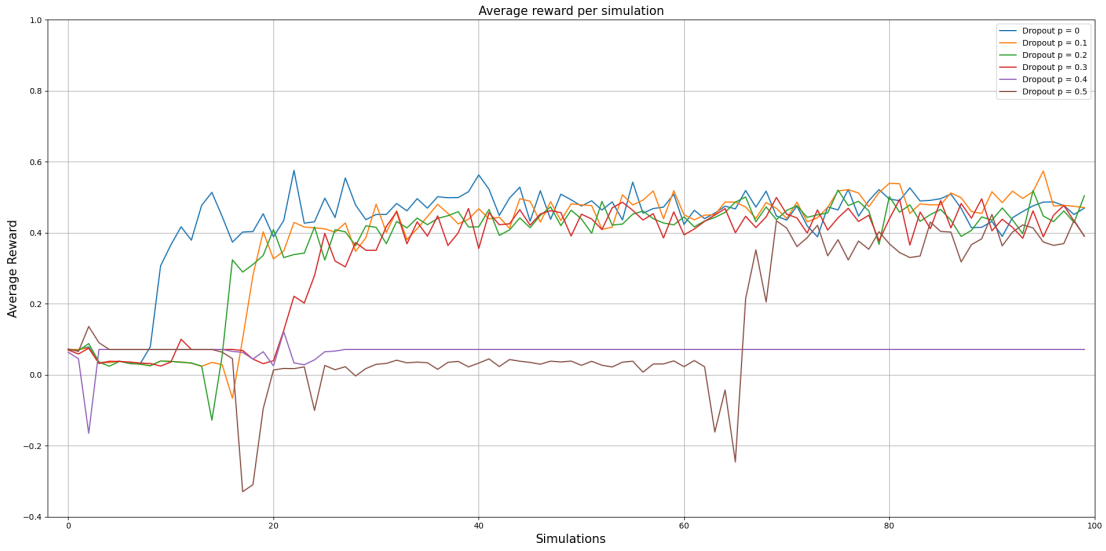

Figure 4.12. Average reward per simulation with different dropout probability

Figure 4.12∼Figure 4.18 show that when the dropout probability is in range of 0.1∼0.3, the model's learning rate is almost the same, as well as its performance in test scenario. But when the probability is greater than 0.4, the model will fail to learn, and always end up with a collision.
Since the test performance of the model is not significantly improved after setting dropout>0 and the learning rate is also slowed down, we do not recommend using this method and choose to keep dropout probability = 0.

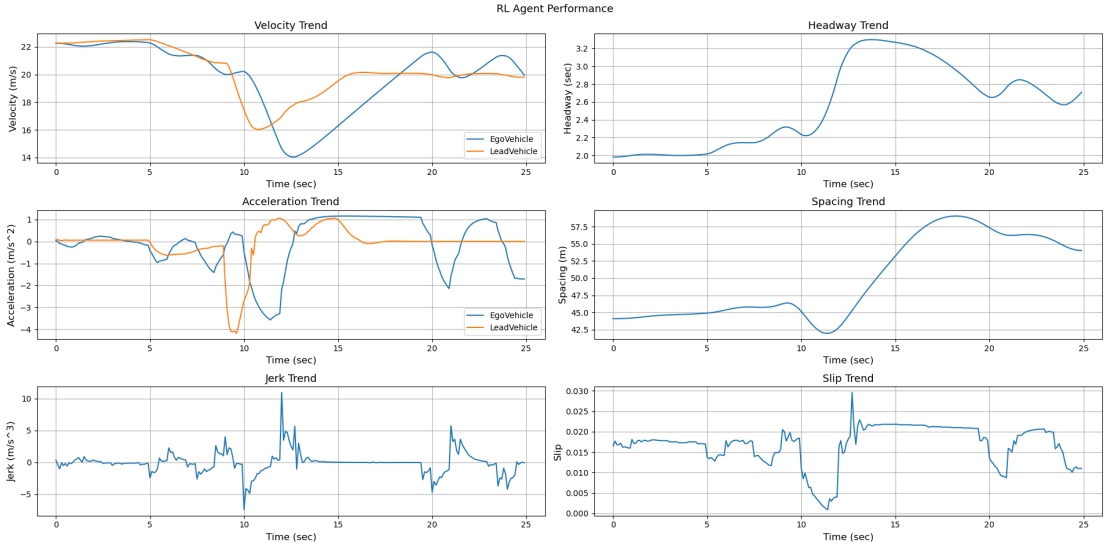

Figure 4.13. RL agent performance (dropout=0.1)

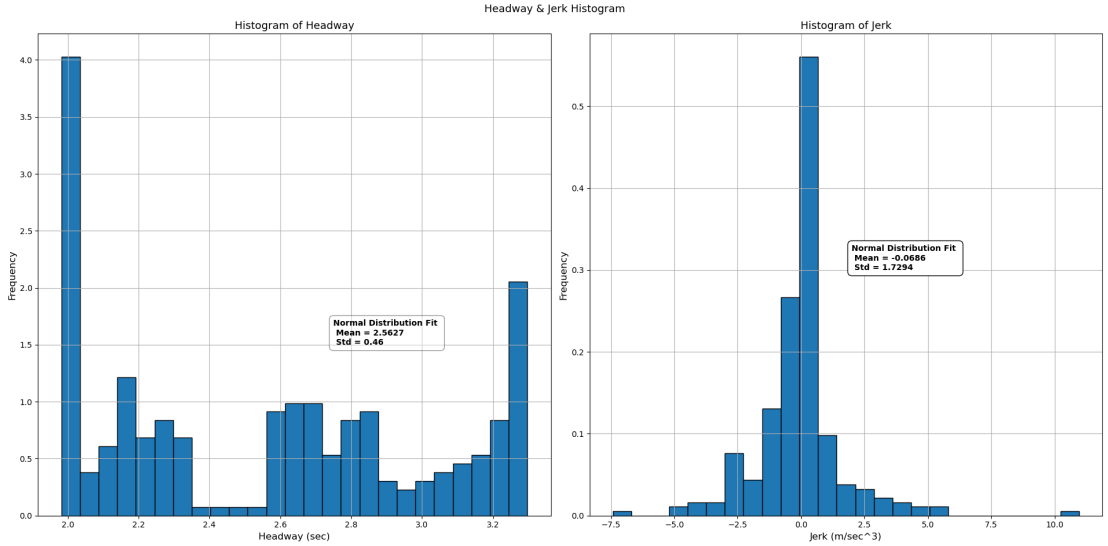

Figure 4.14. Headway & Jerk Histogram (dropout=0.1)

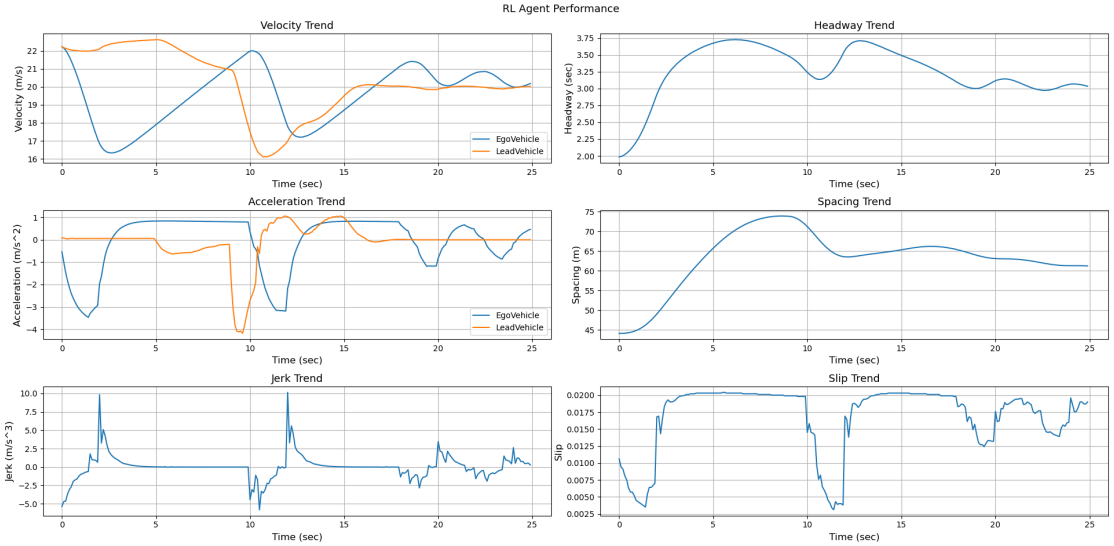

Figure 4.15. RL agent performance (dropout=0.2)

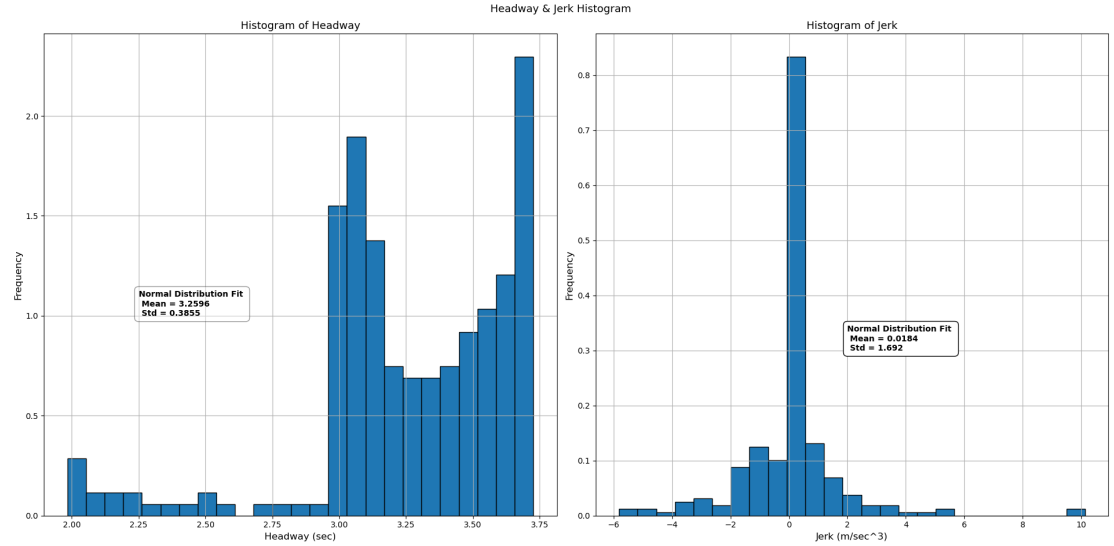

Figure 4.16. Headway & Jerk Histogram (dropout=0.2)

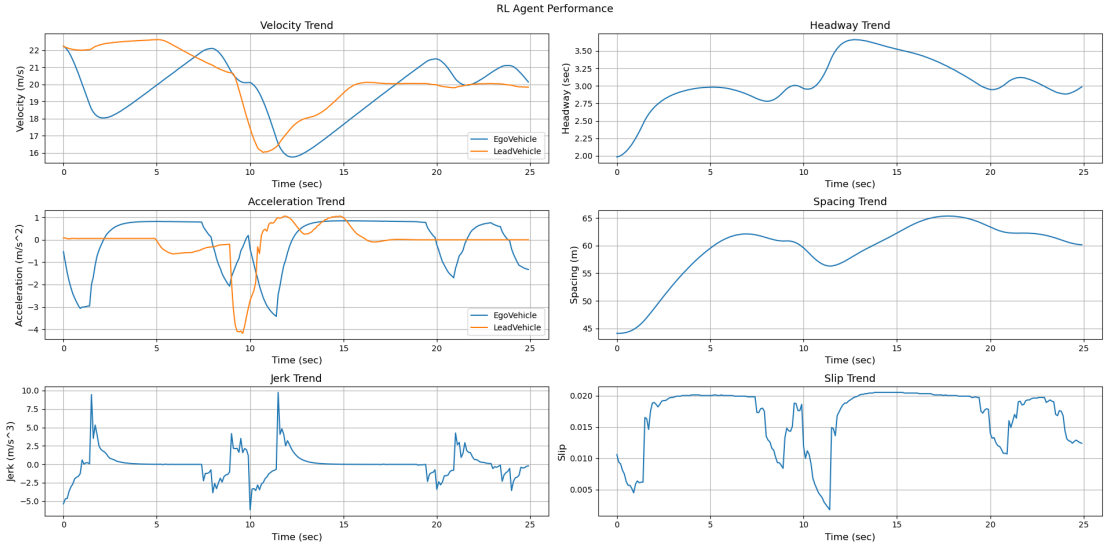

Figure 4.17. RL agent performance (dropout=0.3)

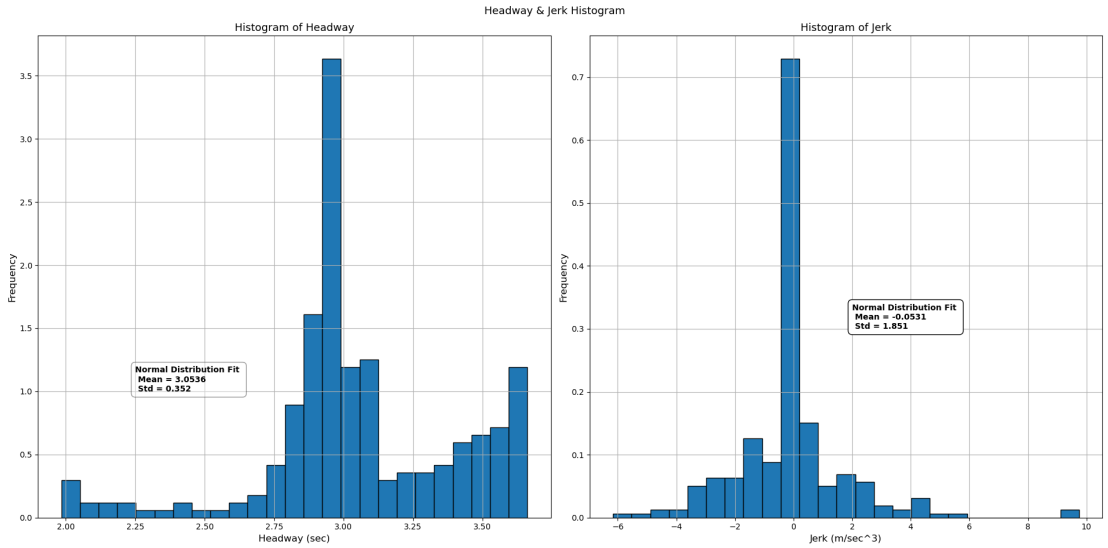

Figure 4.18. Headway & Jerk Histogram (dropout=0.3)

c. **l2 normalization**
L2norm penalty is chosen from list [0, 0.1, 0.2, 0.3, 0.4, 0.5] and simulation results are as followed: Figure 4.19 reflects the DDPG agent training history with different l2norm penalty and Figure 4.20 ∼ Figure 4.25 present relative test performances.

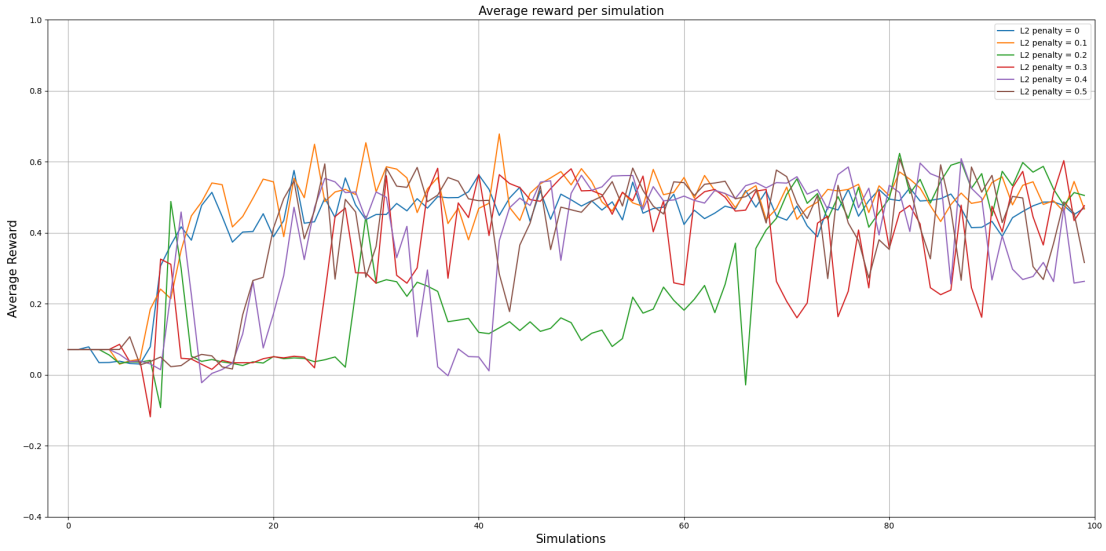

Figure 4.19. Average reward per simulation with different l2norm penalty

When the l2norm penalty is greater than 0.4, the model will fail to learn. Figure 4.20 ∼ Figure 4.25 show that when l2norm penalty = 0.3, the model can achieve excellent performance: The oscillation of jerk is significantly suppressed and model can quickly adjust the velocity close to lead vehicle with good human comfort and vehicle stability.

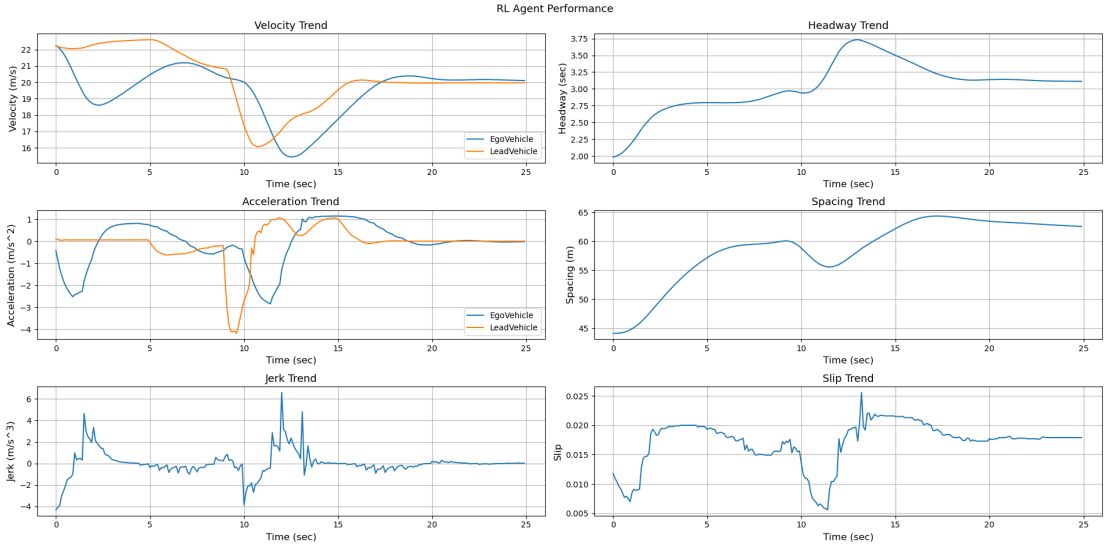

Figure 4.20. RL agent performance (L2norm penalty=0.1)

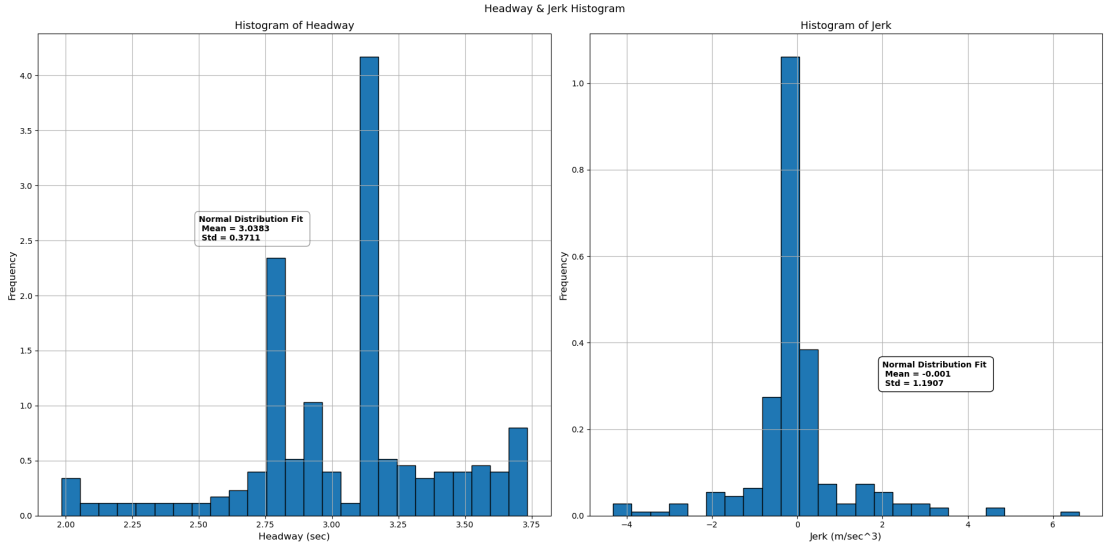

Figure 4.21. Headway & Jerk Histogram (L2norm penalty=0.1)

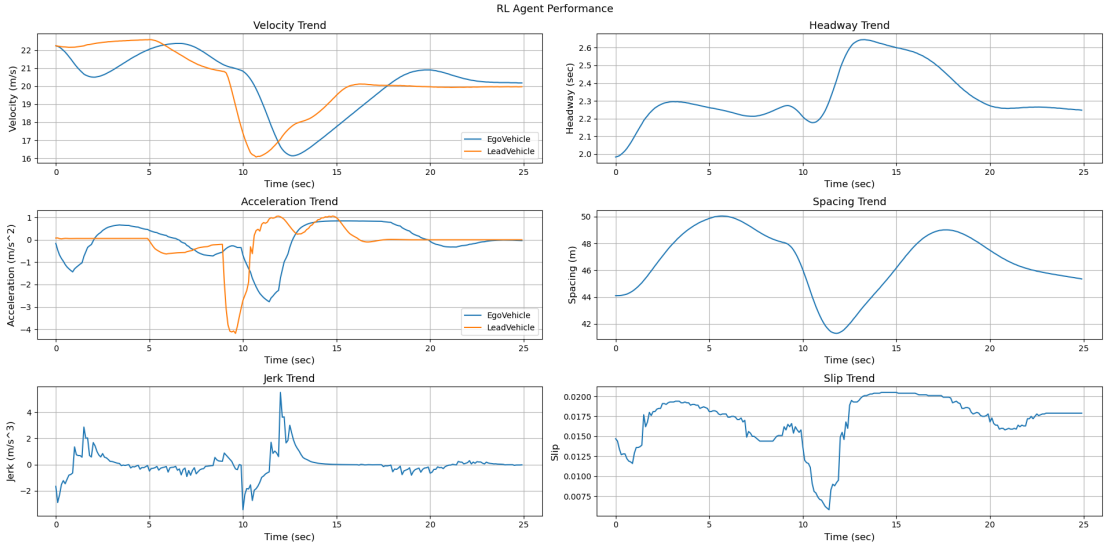

Figure 4.22. RL agent performance (L2norm penalty=0.2)

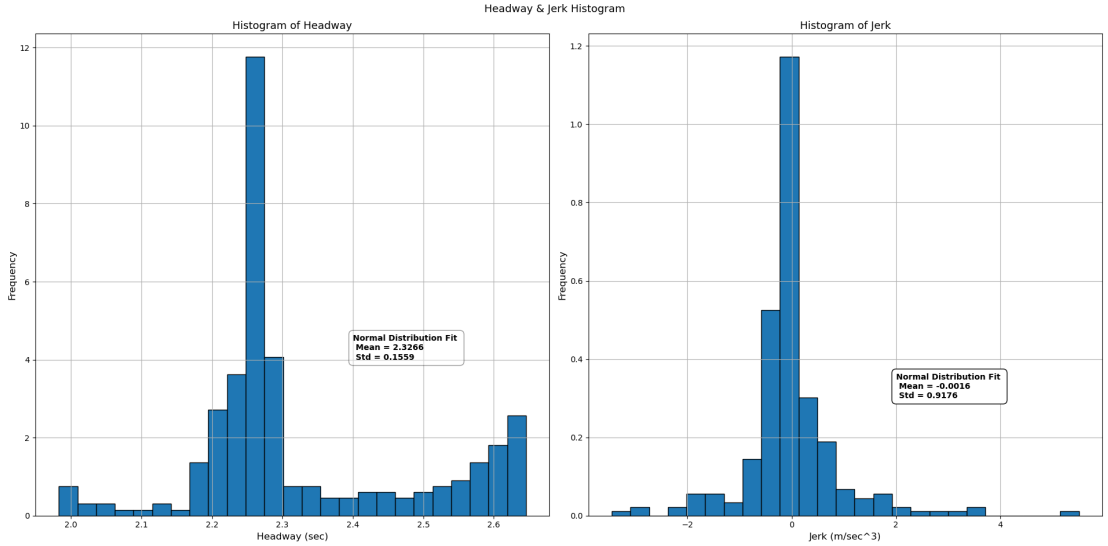

Figure 4.23. Headway & Jerk Histogram (L2norm penalty=0.2)

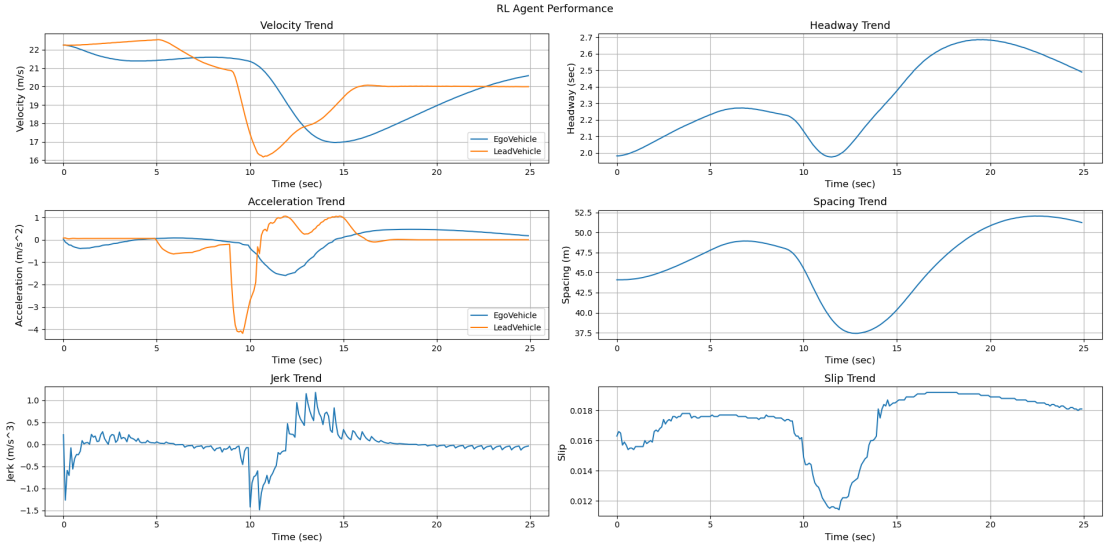

Figure 4.24. RL agent performance (L2norm penalty=0.3)

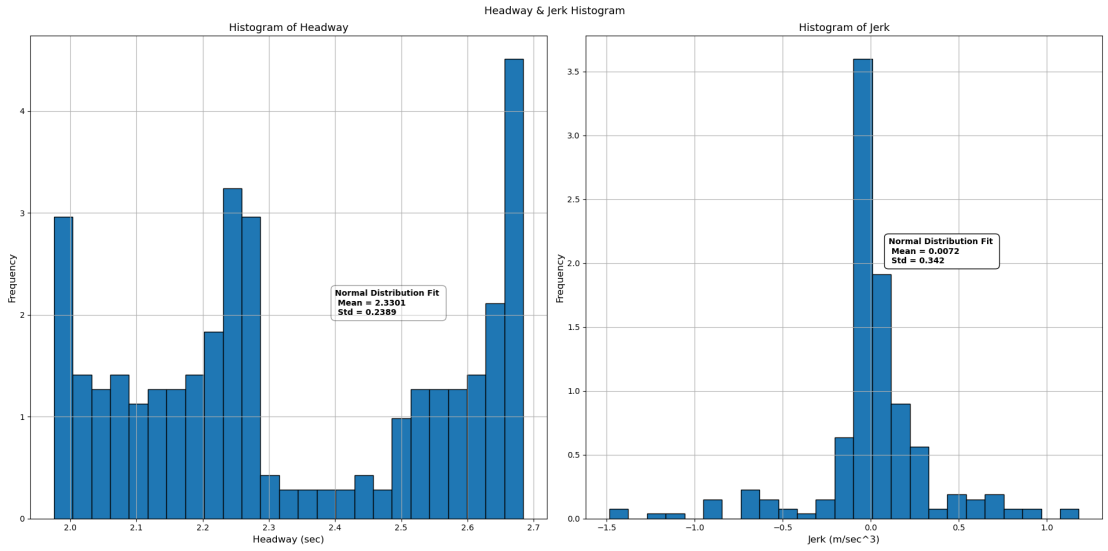

Figure 4.25. Headway & Jerk Histogram (L2norm penalty=0.3)

In summary, the best hyperparameter combination is considered as {batch size = 32, dropout = 0, l2norm penalty = 0.3}

- **Randomize initial state** In this method, we randomize the initial spacing and lead vehicle velocity separately in simulation.
  Spacing varies in range 45 ± 10m.
  Velocity varies in range 22.25 ± 3m/s.
  The relative results are as followed: Figure 4.26 reflects the DDPG agent training history with different parameters randomization and Figure 4.27 ∼ Figure 4.30 present relative test performances.

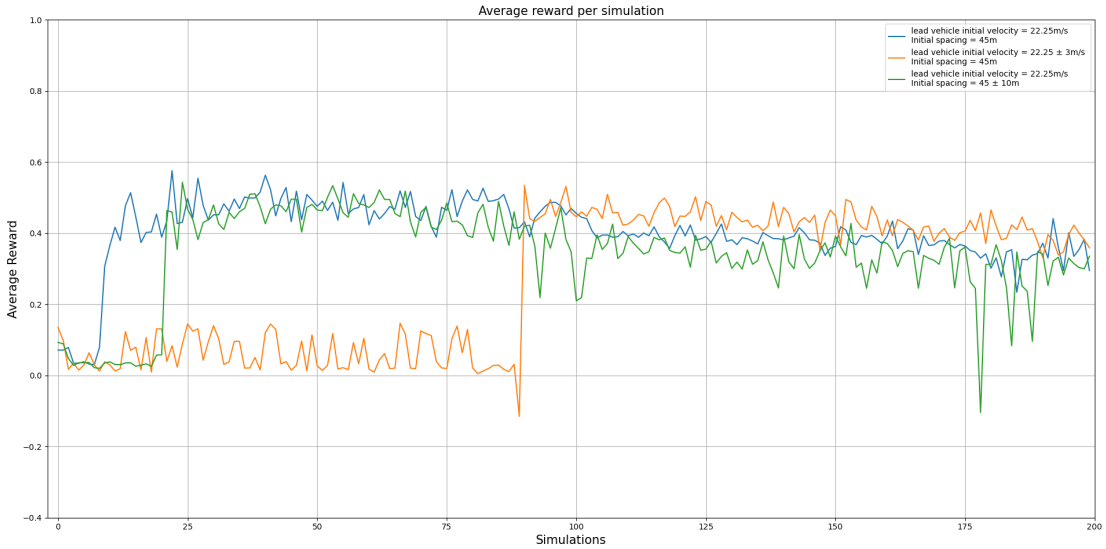

Figure 4.26. Average reward per simulation with different parameters randomization

Figure 4.26 ∼ Figure 4.30 show that the model learning rate is faster with random initial position while their test performances are similar: ego vehicle's velocity and jerk fluctuate a lot and cause discomfort and instability.
Simulation results expose the drawback of deep reinforcement learning, which has poor generalization for random initial configurations. But in these two settings, random initial spacing seems better than velocity. So, we recommend randomizing the initial position only.

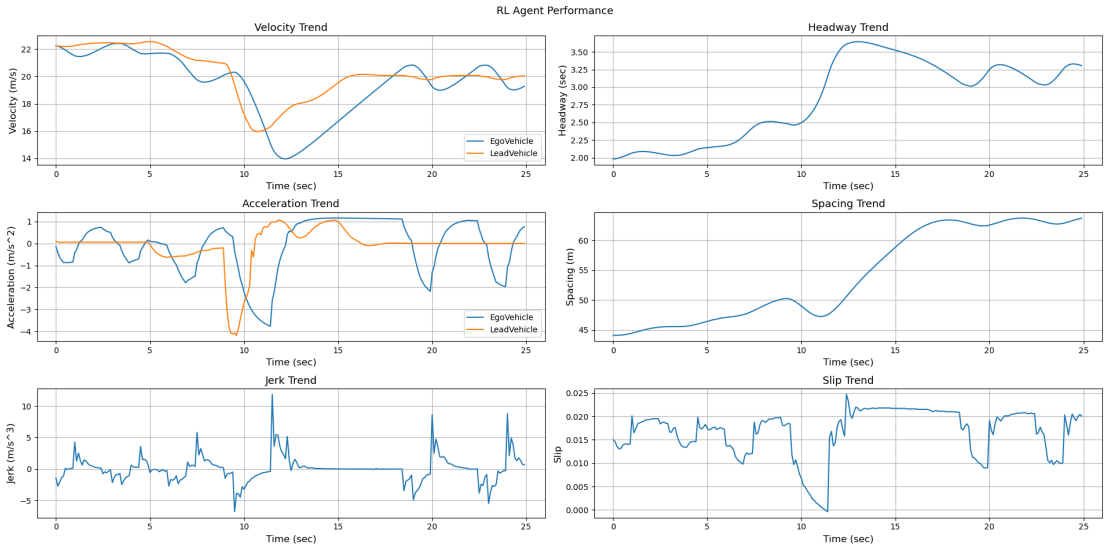

Figure 4.27. RL agent performance (Random initial spacing)

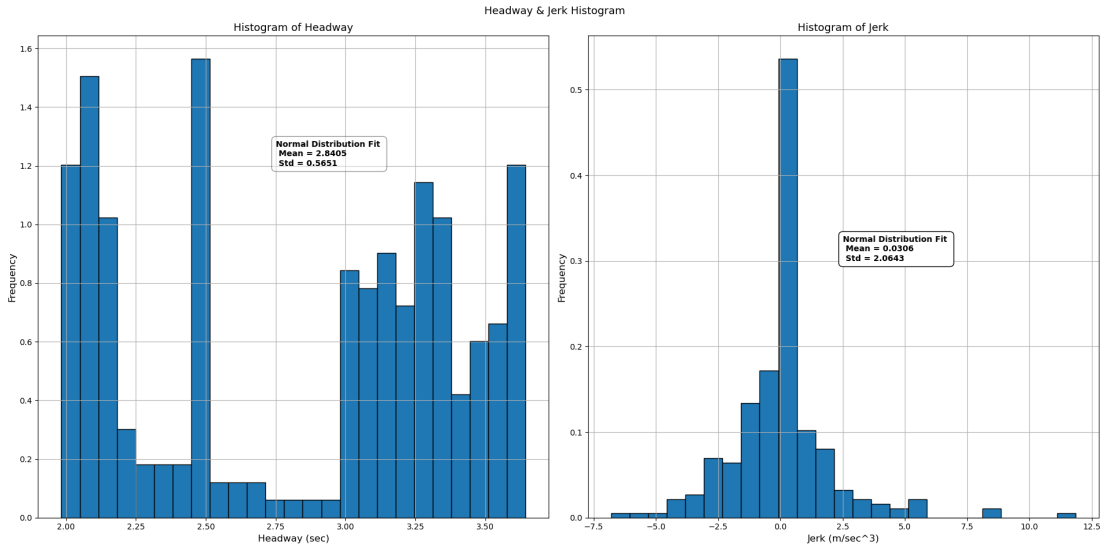

Figure 4.28. Headway & Jerk Histogram (Random initial spacing)

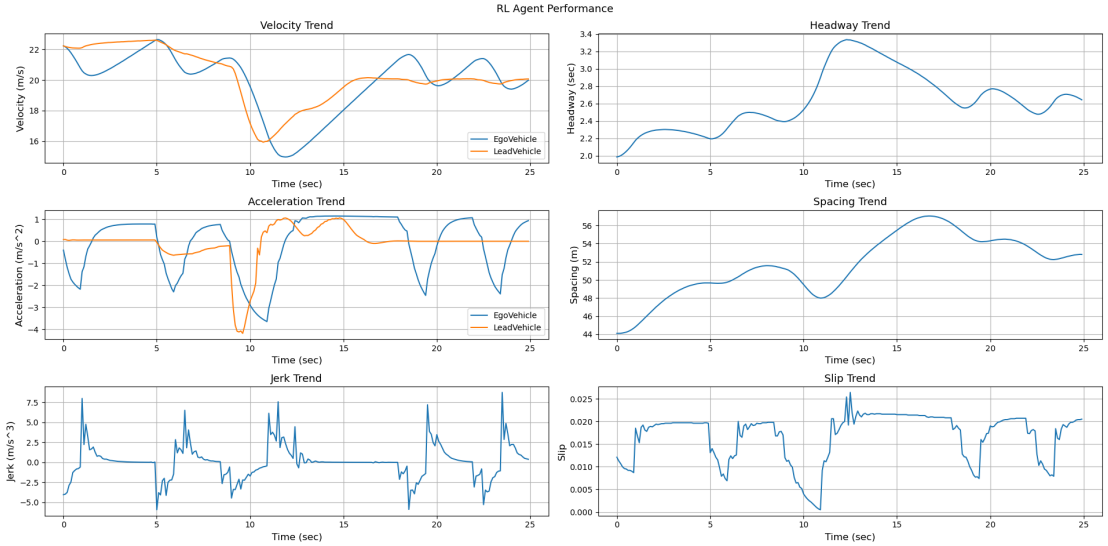

Figure 4.29. RL agent performance (Random initial lead vehicle velocity)

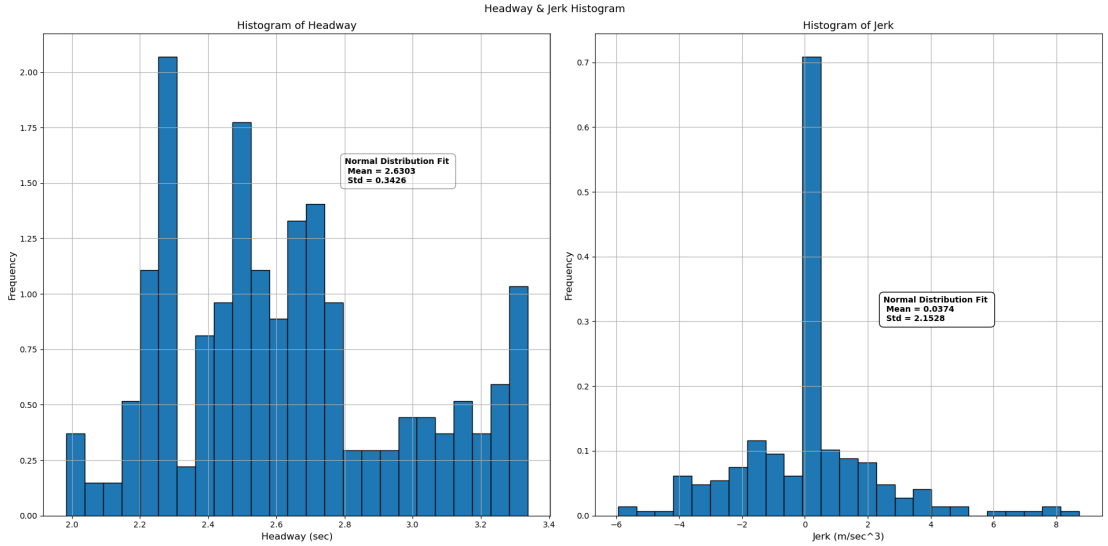

Figure 4.30. Headway & Jerk Histogram (Random initial lead vehicle velocity)

- **Prioritized experience replay**
  In this method, we change the sampling strategy of DDPG to prioritized experience replay and compared with the original DDPG algorithm to verify whether the test performance can be improved. The hyperparameters of DDPG with PER are listed in Table 4.2.

Parameter	**Value**
optimization method	Adam
actor learning rate	0.0001
critic learning rate	0.001
discount factor γ	0.99
soft update factor τ	0.01
buffer size	10000
number of hidden layers	2
number of perceptrons in each hidden layer	256
activation function	ReLU
sampling batch size	32
dropout probability	0
l2norm penalty	0
exponent α	0.7
exponent β	initial value = 0.5 and is linearly increased to 1 at the end of simulation

Table 4.2. DDPG with PER hyperparameters

The simulation results are as followed:
Figure 4.31 reflects the training history and Figure 4.32 $\sim$ Figure 4.33 present test performances of DDPG with PER algorithm.

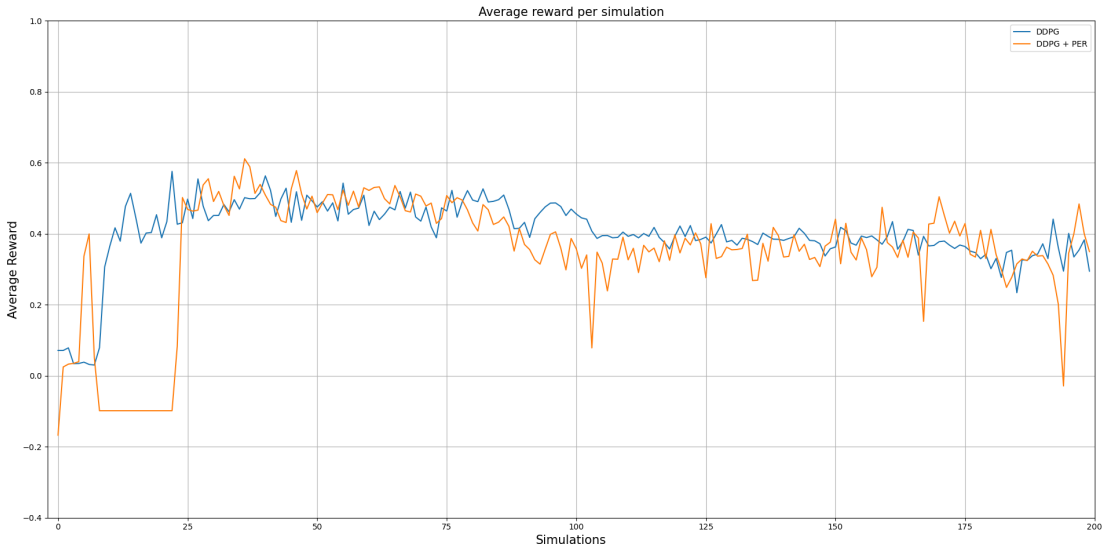

Figure 4.31. Average reward per simulation with different sampling methods

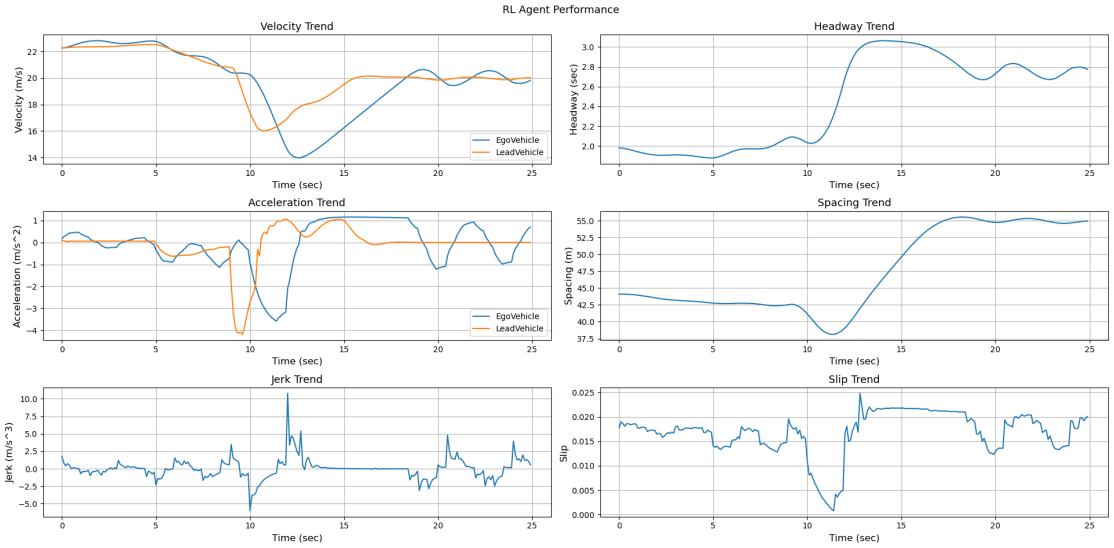

Figure 4.32. RL agent performance (DDPG with PER method)

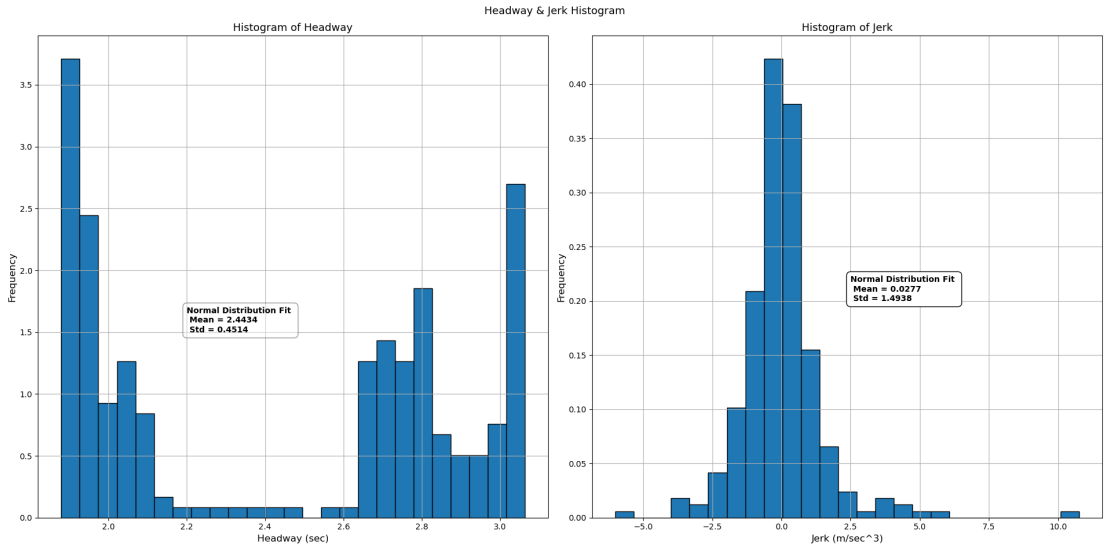

Figure 4.33. Headway & Jerk Histogram (DDPG with PER method)

Figure 4.31 shows that the model using PER method can get a high score with fewer episodes(episode=6) and the average reward upper limit is higher than the original algorithm(episode=36). However, according to the variation of the average reward, it's obvious that DDPG with PER has lower convergence stability.

After comparing the test performance of DDPG with PER(Figure 4.32 $\sim$ Figure 4.33) and the original one(Figure 4.10 $\sim$ Figure 4.11), we can see that PER method does help DDPG achieve slightly better result because the oscillation of jerk is reduced, but to be honest, the improvement is not significant. So we use another actor-critic algorithm TD3 in Section 4.2.3 and attempt to get better results.

4.2.3 Comparison between DDPG and TD3

A common problem with DDPG is that it tends to overestimates the Q value[42], i.e. exploits the wrong Q value, which may cause policy breaking. Twin Delayed DDPG (TD3) is an algorithm that solves this issue. So in this section, we train the TD3 algorithm in two vehicles following scenario and compare its test performance with the DDPG algorithm.
The hyperparameters of TD3 are listed in Table 4.3.

Parameter	**Value**
optimization method	Adam
actor learning rate	0.001
critic learning rate	0.001
discount factor γ	0.99
soft update factor τ	0.001
policy noise standard deviation σ	0.1
policy noise absolute value limit c	0.5
buffer size	10000
number of hidden layers	2
number of perceptrons in each hidden layer	64
activation function	ReLU
delay frequency	2
sampling batch size	32
dropout probability	0
l2norm penalty	0

Table 4.3. TD3 hyperparameters

The simulation results are as followed:
Figure 4.34 reflects the training history and Figure 4.35 $\sim$ Figure 4.36 present test performances of TD3 algorithm.

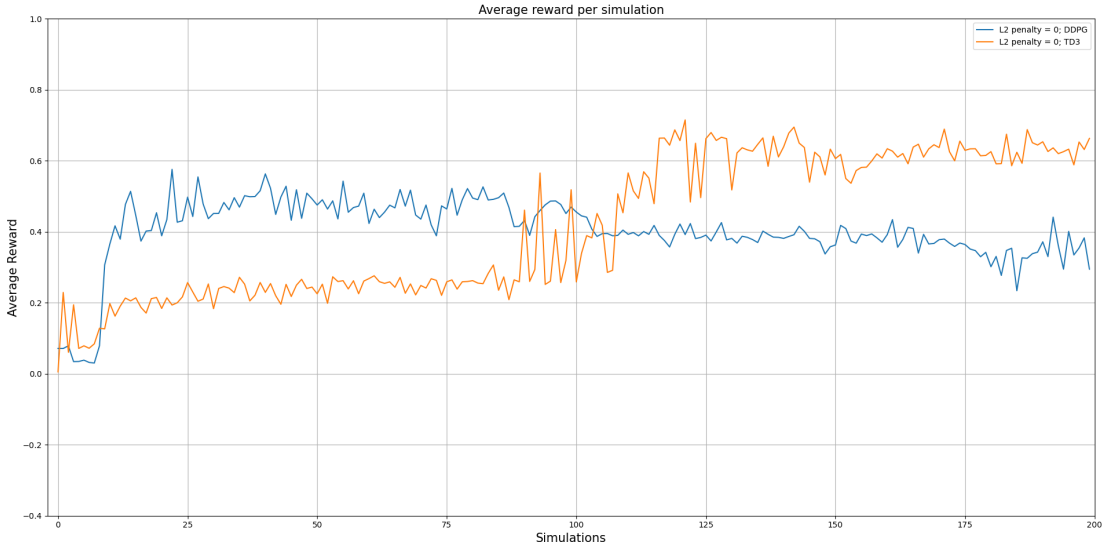

Figure 4.34. Average reward per simulation of TD3

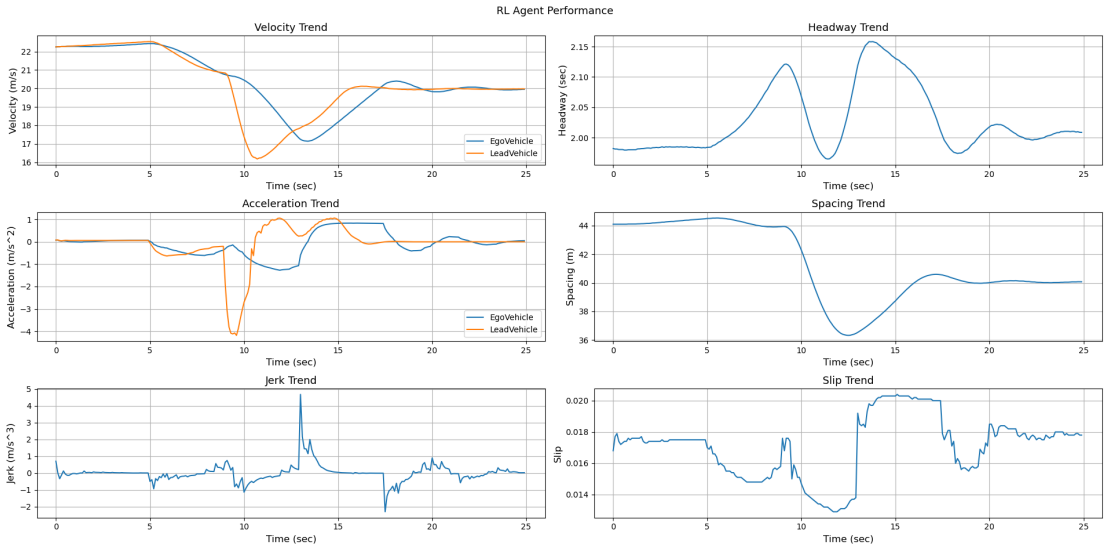

Figure 4.35. RL agent performance of TD3

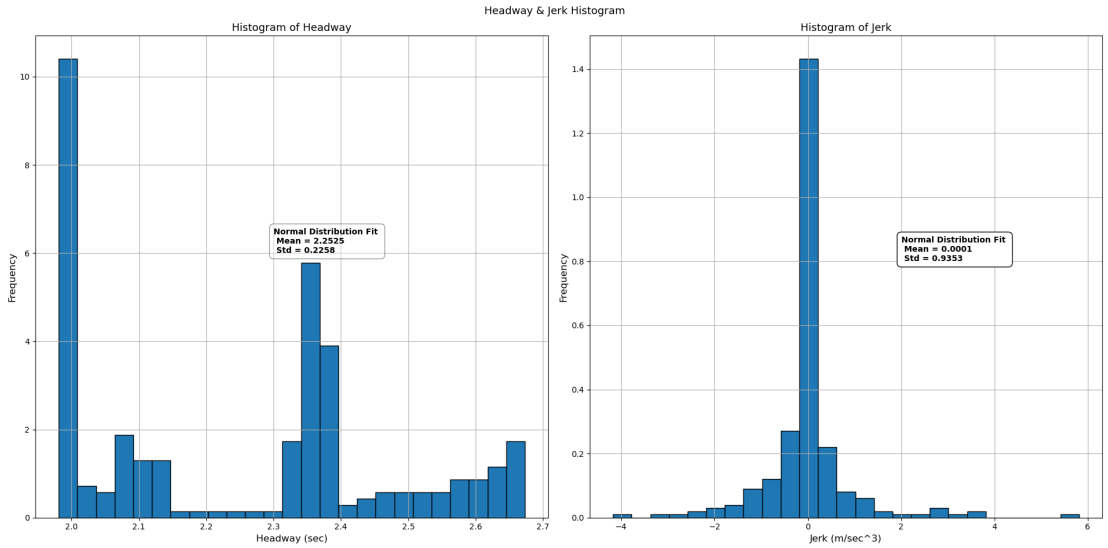

Figure 4.36. Headway & Jerk Histogram of TD3

Figure 4.34 shows that since TD3 adopts a delayed policy learning strategy (updates the policy network less frequently than the Q network), the learning rate of TD3 is slower than that of DDPG in the first 100 episodes. However, as the number of episode increases, the average reward of TD3 algorithm becomes obviously higher than DDPG and its convergence is quite stable, which is aligned with our expectation that TD3 makes a significant improvement over DDPG indeed.
The comparison between Figure 4.35 ∼ Figure 4.36 and Figure 4.10 ∼ Figure 4.11 shows that TD3 also has better performance in the test scenario: the standard deviation of jerk is reduced to less than half of DDPG and the acceleration oscillation is significantly suppressed, which result in high level of human comfort; the model can control the ego vehicle speed basically the same as the lead vehicle, which greatly improve driving safety and stability.

4.2.4 DDPG in three vehicles cut-out scenario

In order to verify the generalization of the DRL algorithm, we also train DDPG in three vehicles cut-out scenario and test its performance. The simulation results are as followed: Figure 4.37 reflects the training history and Figure 4.38 ∼ Figure 4.41 present test performances of TD3 algorithm.

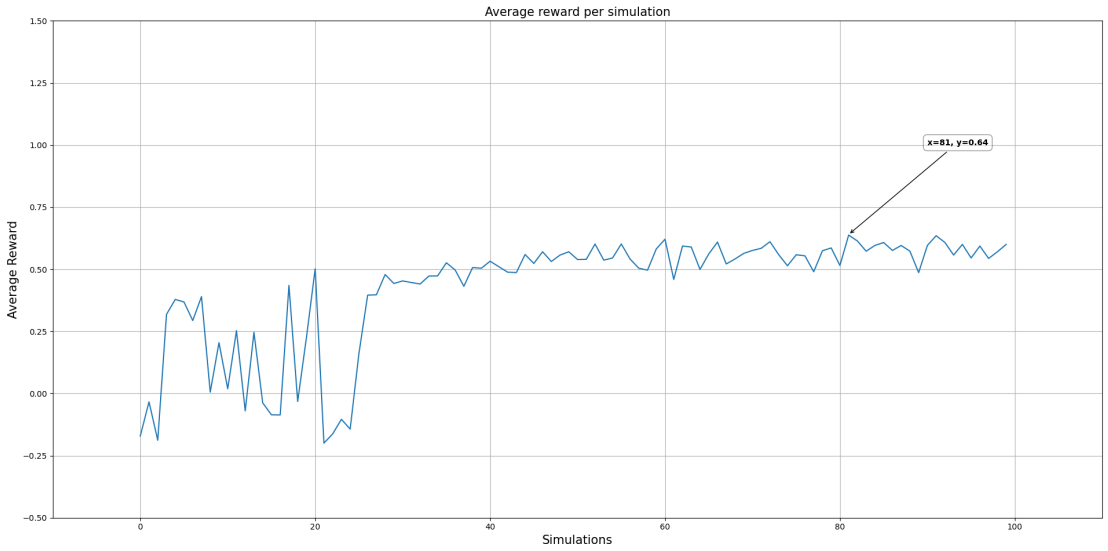

Figure 4.37. Average reward per simulation of cut-out scenario

Figure 4.37 shows the model converges quickly in the training process. Although the step average reward is not very close to 1 after stabilization, based on the information of Figure 4.38, we can find that the DRL agent actually controls the vehicle well: during the whole simulation, the model can quickly adjust velocity close to vehicle B after vehicle A changes lane and keep the spacing above 20m, which means ego vehicle successfully identifies the front slow vehicle B and avoids the collision; and according to position variation plot, we can see that the model imitates lead vehicle driving with approximately the same linear variation trend.
Figure 4.39 shows speed and comfort reward obtain penalty in the interval [15,20]sec due to necessary reaction for the line change of vehicle A. Figure 4.40 shows their weighted variation.
Figure 4.41 presents the distribution of headway and jerk, which shows that the headway is relatively concentrated, while jerk has some larger discrete values that makes its standard deviation higher.
In summary, the results show that model can well adapt to the cut-out scenario.

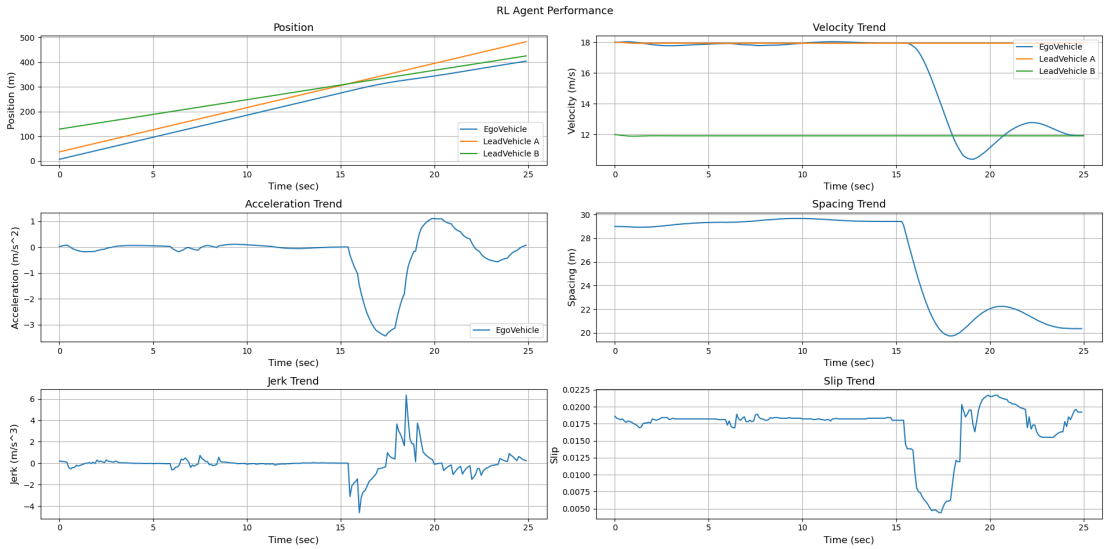

Figure 4.38. RL agent performance of cut-out scenario

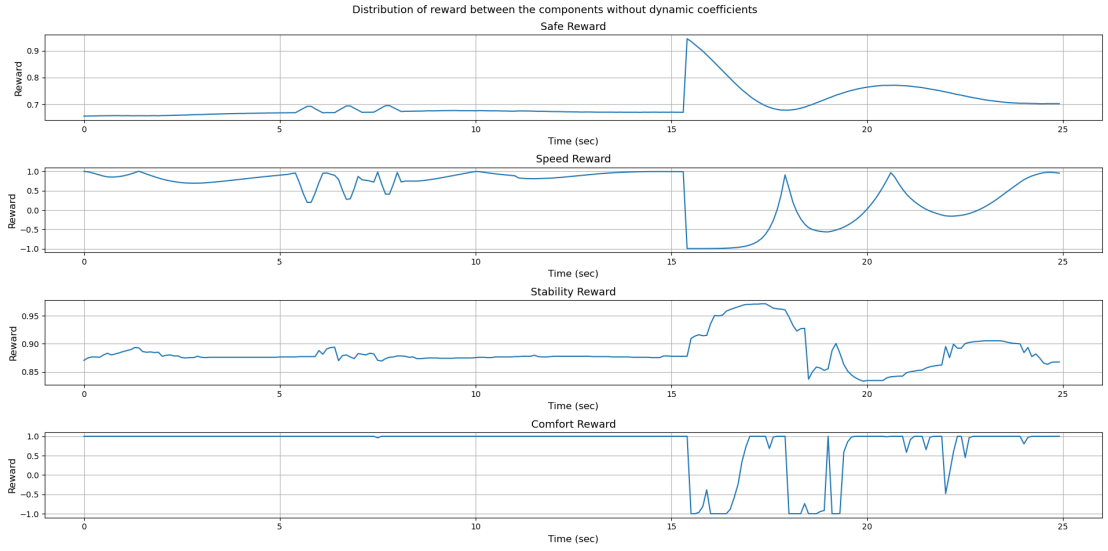

Figure 4.39. Reward components variation of cut-out scenario

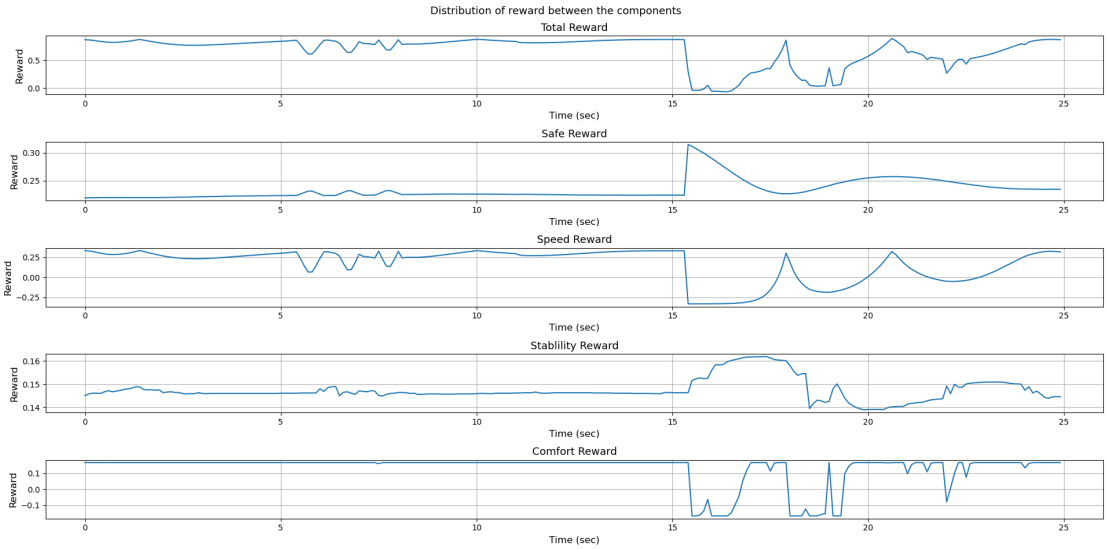

Figure 4.40. Reward components weighted variation of cut-out scenario

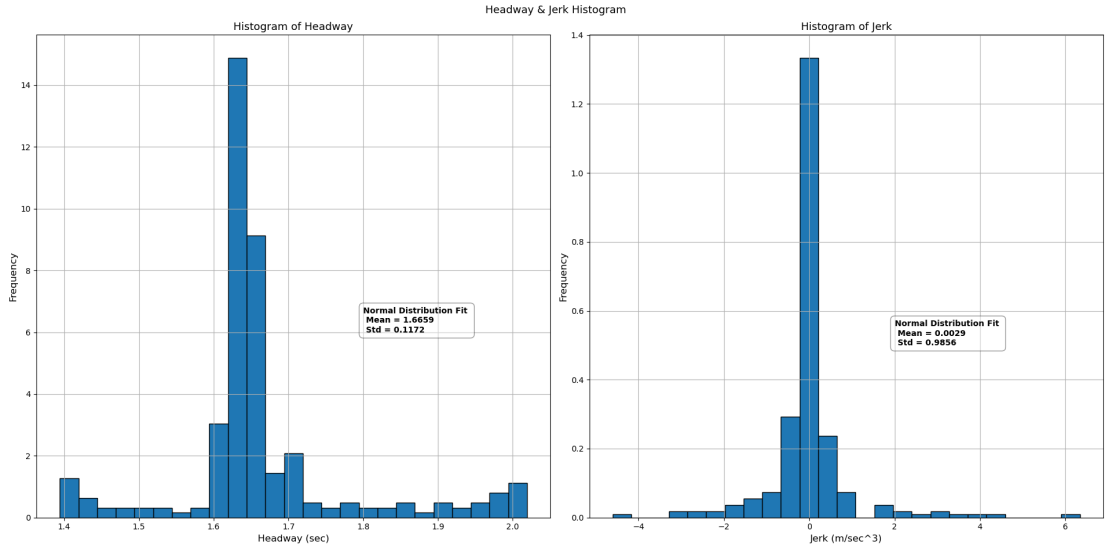

Figure 4.41. Distribution of Headway & Jerk of cut-out scenario

Chapter 5

Conclusion

In this thesis, we applied several deep reinforcement learning algorithms to achieve vehicle longitudinal control in CoMoVe framework. Through a proper design of reward functions, we guided the model to learn in the direction of achieving safe driving while taking into account passenger comfort and vehicle stability. As for algorithm optimization, we have significantly improved the performance of the DDPG algorithm and suppressed its oscillation in test scenario by fine-tuning hyperparameters, regularization and prioritized experience replay methods. In the meanwhile, by expanding the training scenarios of the DDPG algorithm, we improved its generalisation and made the algorithm become more robust. Due to the inherent drawbacks of the DDPG algorithm (such as overestimate of Q value), we also tested its improved version TD3 and successfully increased the average reward by 50%, which enriches the available options of control algorithms for CoMoVe framework and provides valuable reference for further improvement.
There are also some deficiencies during the process of algorithm deployment. So I think the following aspects can be improved in the future: while ensuring the lightness, the complexity of the neural network can be appropriately increased, such as the number of layers and perceptrons in the network; on the basis of achieving vehicle longitudinal control, we can increase the dimension of DRL agent's action, such as the steering; explore the interaction behavior and algorithm performance in the scenario where multiple DRL agents are coexisting to assist the realization of multi-agent cooperative driving system.

Bibliography

[1] Stéphanie Bouckaert, Araceli Fernandez Pales, Christophe McGlade, Uwe Remme, Brent Wanner, Laszlo Varro, Davide D'Ambrosio, and Thomas Spencer. Net zero by 2050: A roadmap for the global energy sector. 2021.

[2] World Health Organization. Global plan for the decade of action for road safety 2021-2030. 2021.

[3] DINESH CYRIL SELVARAJ, SHAILESH SUDHAKARA HEGDE, NICOLA AMATI, CARLA FABIANA CHIASSERINI, and FRANCESCO PAOLO DE-FLORIO. A reinforcement learning approach for efficient, safe and comfortable driving. http://hdl.handle.net/11583/2910374, 2021.

[4] Matthew Veres and Medhat Moussa. Deep learning for intelligent transportation systems: A survey of emerging trends. *IEEE Transactions on Intelligent transportation systems*, 21(8):3152–3168, 2019.

[5] Fangchun Yang, Shangguang Wang, Jinglin Li, Zhihan Liu, and Qibo Sun. An overview of internet of vehicles. *China communications*, 11(10):1–15, 2014.

[6] Thomas Nolte, Hans Hansson, and Lucia Lo Bello. Automotive communications-past, current and future. In *2005 IEEE Conference on Emerging Technologies and Factory Automation*, volume 1, pages 8–pp. IEEE, 2005.

[7] Fangchun Yang, Jinglin Li, Tao Lei, and Shangguang Wang. Architecture and key technologies for internet of vehicles: a survey. *Journal of Communications and Information Networks*, 2(2):1–17, 2017.

[8] Carlos Renato Storck and Fátima Duarte-Figueiredo. A survey of 5g technology evolution, standards, and infrastructure associated with vehicle-to-everything communications by internet of vehicles. *IEEE access*, 8:117593–117614, 2020.

[9] The Center for Advanced Automotive Technology. Connected and automated vehicles. http://autocaat.org/Technologies/Automated_and_Connected_Vehicles.

[10] Intelligent Transportation Systems. Vehicle-to-infrastructure (v2i) resources. https://www.its.dot.gov/v2i/.

[11] United States Department of Transportation. Vehicle-to-vehicle

communication. https://www.nhtsa.gov/technology-innovation/vehicle-vehicle-communication.

[12] Heejae Kim, Jiyong Han, Seong-Hwan Kim, Jisoo Choi, Dongsik Yoon, Minsu Jeon, Eunjoo Yang, Nhat Pham, Sungpil Woo, Jeongkyu Park, Daeyoung Kim, and Chan-Hyun Youn. Isv2c: An integrated road traffic-network-cloud simulator for v2c connected car services. In *2017 IEEE International Conference on Services Computing (SCC)*, pages 434–441, 2017.

[13] Intelligent Transportation Systems. Vehicle-to-pedesrian (v2p) communications for safety. https://www.its.dot.gov/research_archives/safety/v2p_comm_safety.htm.

[14] Khadige Abboud, Hassan Aboubakr Omar, and Weihua Zhuang. Interworking of dsrc and cellular network technologies for v2x communications: A survey. *IEEE transactions on vehicular technology*, 65(12):9457–9470, 2016.

[15] Zachary MacHardy, Ashiq Khan, Kazuaki Obana, and Shigeru Iwashina. V2x access technologies: Regulation, research, and remaining challenges. *IEEE Communications Surveys & Tutorials*, 20(3):1858–1877, 2018.

[16] Arne Kesting, Martin Treiber, Martin Schönhof, and Dirk Helbing. Adaptive cruise control design for active congestion avoidance. *Transportation Research Part C: Emerging Technologies*, 16(6):668–683, 2008.

[17] Vicente Milanés, Steven E Shladover, John Spring, Christopher Nowakowski, Hiroshi Kawazoe, and Masahide Nakamura. Cooperative adaptive cruise control in real traffic situations. *IEEE Transactions on intelligent transportation systems*, 15(1):296–305, 2013.

[18] Jeroen Ploeg, Bart TM Scheepers, Ellen Van Nunen, Nathan Van de Wouw, and Henk Nijmeijer. Design and experimental evaluation of cooperative adaptive cruise control. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 260–265. IEEE, 2011.

[19] Louis A Pipes. An operational analysis of traffic dynamics. *Journal of applied physics*, 24(3):274–281, 1953.

[20] Katsuya Hasebe, Akihiro Nakayama, and Yūki Sugiyama. Dynamical model of a cooperative driving system for freeway traffic. *Physical review E*, 68(2):026102, 2003.

[21] Rajesh Rajamani and Steven E Shladover. An experimental comparative study of autonomous and co-operative vehicle-follower control systems. *Transportation Research Part C: Emerging Technologies*, 9(1):15–31, 2001.

[22] Meng Wang, Winnie Daamen, Serge P Hoogendoorn, and Bart van Arem. Rolling horizon control framework for driver assistance systems. *Transportation research part C: emerging technologies*, 40:271–311, 2014.

[23] Yang Zhou, Soyoung Ahn, Madhav Chitturi, and David A Noyce. Rolling horizon stochastic optimal control strategy for acc and cacc under uncertainty. *Transportation Research Part C: Emerging Technologies*, 83:61–76, 2017.

[24] Hans Fritz. Longitudinal and lateral control of heavy duty trucks for automated vehicle following in mixed traffic: experimental results from the chauffeur project. In *Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No. 99CH36328)*, volume 2, pages 1348–1352. IEEE, 1999.

[25] Tom Robinson, Eric Chan, and Erik Coelingh. Operating platoons on public motorways: An introduction to the sartre platooning programme. In *17th world congress on intelligent transport systems*, volume 1, page 12, 2010.

[26] Dinesh Cyril Selvaraj, Shailesh Hegde, Carla Fabiana Chiasserini, Nicola Amati, Francesco Deflorio, and Giuliana Zennaro. A full-fledge simulation framework for the assessment of connected cars. 52:315–322, 2021.

[27] Bruce P Minaker. *Fundamentals of vehicle dynamics and modelling: A textbook for engineers with illustrations and examples*. John Wiley & Sons, 2019.

[28] Dinesh Cyril Selvaraj. Development of a framework for sensor- and communication-assisted vehicle dynamic., 2020.

[29] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[30] ID Jacobson, LG Richards, and AR Kuhlthau. Models of human comfort in vehicle environments. *Human Factors in Transport Research Edited by DJ Oborne, JA Levis*, 2, 1980.

[31] Vehicle Dynamics Standards Committee. Vehicle dynamics terminology. 2008.

[32] Genta. *Automotive Chassis: Volume 1: Components Design*. Springer International Publishing, 2020.

[33] OpenAI Gym and Nimish Sanghi. Deep reinforcement learning with python.

[34] Martijn van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement learning*, pages 3–42. Springer, 2012.

[35] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[36] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[38] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[39] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[40] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[42] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[43] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.