Human-Like Autonomous Car-Following Planning by Deep Reinforcement Learning

Meixin Zhu
Graduate Student
School of Transportation Engineering
Tongji University
4800 Cao'an Road, Jiading District
Shanghai, 201804, China
Email: 1151208@tongji.edu.cn

Xuesong Wang, Ph.D.
Professor
School of Transportation Engineering
Tongji University
4800 Cao'an Road, Jiading District
Shanghai, 201804, China
Phone: +86-21-69583946
Email: wangxs@tongji.edu.cn
(Corresponding author)

Yinhai Wang, Ph.D.
Professor
Department of Civil and Environmental Engineering
University of Washington
Seattle, WA 98195-2700
Email: yinhai@uw.edu

Word Count: 4,923
Tables and Figures (10 at 250 words each): 2,500
Total Count: 7,423

July 2017
Prepared for Presentation at the TRB Annual Meeting

ABSTRACT

This study proposes a framework for human-like autonomous car-following planning based on deep reinforcement learning (RL). In the framework, historical driving data are fed into a simulation environment, where an RL agent learns from trying and interaction, with a reward function signaling how much the agent deviates from the empirical data. An optimal policy (car-following model) that maps from speed, relative speed, and spacing to following-vehicle acceleration in a human-like way is finally obtained, and can be continuously updated when more data are fed in. Two thousand car-following periods extracted from the Shanghai Naturalistic Driving Study were used to train the model and compare its performance with that of four traditional car-following models.

Results show that this new model can reproduce human-like car-following behavior with significantly higher accuracy than traditional car-following models, especially in terms of speed replicating. Specifically, the model has a validation error of 18% on spacing and 5% on speed, which is generally 15 and 30 percentage points less, respectively, than that of traditional car-following models. Moreover, the model demonstrates good capability of generalization to different driving situations and can adapt to different drivers by continuously learning. These results can contribute to the development of human-like autonomous driving algorithms. Moreover, this study demonstrates that data-driven modeling and reinforcement learning methodology can contribute to the development of traffic-flow models and offer deeper insight into driver behavior.

Keywords: Autonomous Car Following, Human-Like Driving Planning, Deep Reinforcement Learning, Naturalistic Driving Study, and Deep Deterministic Policy Gradient

1 INTRODUCTION

Autonomous driving technology is capable of providing driver convenience and enhancing safety by avoiding some crashes caused by driver errors (*1*). While building autonomous driving algorithms, one significant challenge is to make the autonomous vehicles able to emulate human drivers' intelligence and driving styles while staying within the safety bounds, i.e., human-like driving (*2-4*).

Human-like driving will 1) make passengers of the autonomous vehicle feel more comfortable and convinced of the car's ability to drive itself, and 2) enable drivers of surrounding vehicles to better predict and understand the autonomous vehicle's behavior so that they can interact with it more naturally (*1*). Therefore, it is desirable to introduce a driver model that reproduces individual driver's behaviors or trajectories to achieve human-like driving.

Reproducing driving trajectories has been one of the main objectives for modeling longitudinal motion of vehicles in a traffic flow; these models are known as car-following models. Car-following models are one of the main sub-models for the microscopic simulation of traffic flows (*5*); they also represent a theoretical reference for autonomous car-following systems (*6*).

Since the early investigation of car-following dynamics in 1953 (*7*), numerous car-following models have been developed. Although these traditional car-following models have produced great achievements in microscopic traffic simulation, they have limitations in one or more of the following aspects when applied to autonomous car-following planning.

First, limited accuracy. Most existing car-following models are simplified to be parsimonious, i.e., containing only a small number of parameters (*8*). On the one hand, parsimony offers some desired practical properties, including good analytical conclusions and fast simulation speed. On the other hand, parsimony also restricts model flexibility and accuracy, because the inherently complex car-following process can hardly be fully modeled using only several parameters.

Second, poor generalization capability. The calibration of a car-following model using empirical data tries only to emulate human drivers' output (e.g., speed and spacing), rather than understanding the reason for making these decisions (*1*). Therefore, the calibrated car-following model has difficulty in dealing with traffic scenarios and drivers that were not presented in the calibration dataset.

Third, incapable of adaptive updating. Most parameters of car-following models are fixed to reflect the average driver's characteristics (*9*). If car-following models with such pre-set parameters are used for autonomous vehicles, they reflect neither the driving style of the drivers who actually use the vehicles nor the contexts in which they drive. In this sense, such an approach cannot automatically adapt to different drivers or different driving situations.

To answer the questions, we propose a deep reinforcement learning (deep RL) based car-following model that applies neural network and reinforcement learning to address the planning of human-like autonomous car following. Deep RL is a field that seeks to combine the

advances in deep neural networks with reinforcement learning algorithms to create agents capable of acting intelligently in complex environments (*10*), and exciting breakthroughs have been witnessed, like deep *Q*-network (*11*) and AlphaGo (*12*).

Deep RL has the promise to address the limitations of traditional car-following models in that 1) neural networks, known as adept general purpose function approximators (*13*), can achieve higher accuracy in approximating the complicated relationship between stimulus (e.g., relative speed) and reaction (acceleration of the following vehicle) during car-following; 2) reinforcement learning can achieve better generalization capability because it learns decision-making mechanisms from training data rather than parameter estimation through fitting the data (*11*); 3) by continuously learning from historical driving data, deep RL can enable the car to move appropriately in accordance with the behavioral features of its regular drivers.

Figure 1 shows the schematic diagram of human-like car-following framework based on deep RL. During the manual driving phase, data are collected and stored as historical driving data in the database. These data are then fed into a simulation environment, where an RL agent learns from trying and interaction, with a reward function signaling how much the agent deviates from the empirical data. An optimal policy (car-following model) that maps from speed, relative speed, and spacing to following-vehicle acceleration in a human-like way is finally obtained, and can be continuously updated when more data are fed in. This optimal policy will act as the executing policy in the autonomous driving phase.

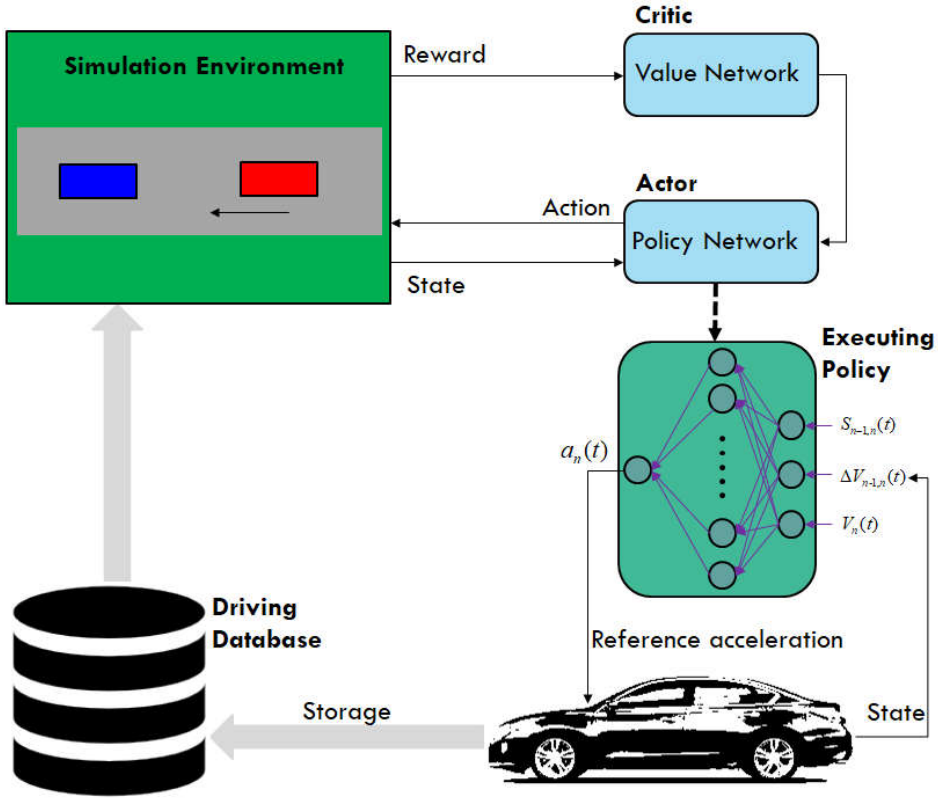


FIGURE 1 Conceptual diagram of human-like car following by deep RL.

To evaluate the proposed model and compare its performance with that of traditional car-following models, real-world driving data collected in the Shanghai Naturalistic Driving Study (*14*) were used to train and test the proposed deep RL model and also several typical traditional car-following models. Their performance in terms of trajectory-reproducing accuracy, generalization, and adaptivity was then compared.

In this paper, a general background is given on car-following models and reinforcement learning. Then, the research data are described and the proposed deep RL based car-following model is specified, followed by training and performance evaluation of the model. The final section is devoted to discussion and conclusion.

2 BACKGROUND

2.1 Car-following Models

A car-following model describes the movements of a following vehicle in response to the actions of the lead vehicle(s). As one of the most important traffic analysis tools, the first car-following models (*7, 15*) were proposed in the middle 1950s. From then on, a large number of car-following models were developed, for example, the Gaxis-Herman-Rothery (GHR) model (*16*), the intelligent driver model (IDM) (*17*), the optimal velocity model (*18*), and the models proposed by Helly (19), Gipps (*20*), and Wiedemann (*21*). For a review and historical development of the subject, consult Brackstone and McDonald (*5*), Olstam and Tapani (*22*), Panwai and Dia (*23*), and Saifuzzaman and Zheng (*24*). Generally, however, the car-following models developed can be divided into four categories: stimulus-based models, safety-distance models, desired-measures models and psycho-physical models (see a review in Saifuzzaman and Zheng (*24*)).

The main idea of a stimulus-based model is that the acceleration of a following vehicle is determined by the driver's reaction to the speed and position differences of the vehicle in front (*25*). The General Motors models are some of the best known stimulus-based models, which have been developed since the late 1950s, with one of their latest modifications proposed by Ozaki (*26*).

The safety-distance models are based on the idea that the driver of a following vehicle would adopt a speed and maintain a distance such that he/she can bring the vehicle to a safe stop should the vehicle in front brake to a sudden stop. The Gipps model (*20*) is based on the safety-distance idea.

The desired-measures models assume that a driver has some preferred situation represented with certain measures (e.g., space headway and following speed), and that a driver continuously attempts to eliminate the difference between the actual situation and the preferred. The intelligent driver model (*27*) is one of the most widely used models using desired measures.

The psycho-physical models suggest that a driver's driving behavior would vary depending on the traffic state he/she is in, such as whether the driver is in the free-flow condition, approaching the vehicle in front, following the vehicle in front or braking. The boundary

conditions defining the different states are usually expressed as a combination of relative speed and relative distance to the front vehicle (*21*).

This study compared four traditional car-following models with the proposed deep RL based model: GHR, Gipps, IDM, and Wiedemann. These specific models were chosen because they were found most representative of those reviewed above.

2.2 Reinforcement Learning

Reinforcement learning (RL) usually solves sequential decision making problems. An RL agent interacts with an environment over time. At each time step t, the agent receives a state s_t and selects an action a_t from some action space A, following a policy $\pi(a_t|s_t)$, which is the agent's behavior, i.e., a mapping from state s_t to actions a_t, receives a scalar reward r_t, and transitions to the next state s_{t+1}, according to the environmental dynamics, or model. In an episodic problem, this process continues until the agent reaches a terminal state, and then it restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the discounted, accumulated reward with the discount factor $\gamma \in (0,1]$. The agent aims to maximize the expectation of such long-term returns from each state (*28*). In general, there are two types of RL methods: value-based and policy-based (*29*).

2.2.1 Value-Based Reinforcement Learning

A value function is a prediction of the expected, accumulative, discounted, future reward, measuring how good each state, or state-action pair is. The action value $Q^{\pi}(s,a) = E[R_t \mid s_t = s, a_t = a]$ is the expected return for selecting action a in state s and then following policy π. Tt represents the "quality" of a certain action a in a given state s. The task of value-based RL methods is to learn the action value function from experience. One example of such an algorithm is Q-learning. Starting from a random Q-function, the agent continuously updates its Q-values by playing the game and obtaining rewards. The iterative updates are derived from the Bellman equation (*29*):

$$Q(s,a) = E[r + \gamma \max_{a'} Q(s',a')] \qquad (1)$$

This equation is based on the following intuition: maximum future reward for this state s and action a is the immediate reward r plus maximum future reward for the next state s'. Using these evolving Q-values, the agent chooses the action with the highest $Q(s,a)$ to maximize its expected future rewards.

2.2.2 Policy-Based Reinforcement Learning

In contrast to value-based methods, policy-based methods optimize the policy $\pi(a \mid s;\theta)$ with function approximation directly, and update the parameters θ by gradient ascent on $E[R_t]$. REINFORCE is a policy-gradient method, updating θ in the direction of $\nabla_\theta \log \pi(a_t \mid s_t;\theta) R_t$

(28).

In order to update the gradients with lower variance and speed up learning, an actor-critic method is usually used. In an actor-critic algorithm, the learning agent is split into two separate entities: the actor (policy) and the critic (value function). The actor is only responsible for generating an action *a*, given the current state *s*. The critic is responsible for evaluating the current policy prescribed by the actor. The critic approximates and updates the value function using samples. The value function is then used to update the actor's policy parameters in the direction of performance improvement (*30*).

2.3 Deep Reinforcement Learning

Deep reinforcement learning refers to reinforcement learning algorithms that use neural networks to approximate any of the following component of reinforcement learning: value function $V(s;\theta)$, policy $\pi(a \mid s;\theta)$, and model (state transition and reward). Here, the parameters θ are the weights in deep neural networks. This field seeks to combine the advances in deep neural networks with reinforcement learning to create agents capable of acting intelligently in complex environments.

2.3.1 Deep Q-Network

Deep RL in discrete action spaces can be performed using the Deep *Q*-Learning method introduced by Mnih et al. (*11*), which employs a single deep network to estimate the value function of each discrete action and, when acting, selects the maximally valued output for a given state input. Deep *Q* networks (*DQN*) work well when there are only a few possible actions but do not function in continuous action spaces. To resolve such difficulties, Lillicrap et al. (*31*) proposed an actor-critic algorithm called deep deterministic policy gradient (DDPG) that builds on *DQN* but can be applied to continuous-control problems.

2.3.2 Deep Deterministic Policy Gradient

Like *DQN*, DDPG uses deep neural network function approximators. Unlike *DQN*, however, DDPG uses two separate actor and critic networks (*31*). The critic network with weights θ^Q approximates the action-value function $Q(s,a|\theta^Q)$. The actor network with weights θ^μ explicitly represents the agent's current policy $\mu(s|\theta^\mu)$ of the *Q*-function, which maps from a state of the environment *s* to an action *a*. To make the learning stable and robust, similar to *DQN*, DDPG deploys experience replay and an idea similar to a target network.

(1) Experience replay

One challenge when using neural networks for reinforcement learning is that most optimization algorithms assume that the samples are independently and identically distributed. Obviously, when the samples are generated by exploring sequentially in an environment this assumption no longer holds. A replay buffer was applied to address these issues.

The replay buffer is a cache D of finite size. Transitions were sampled from the environment according to the exploration policy, and the tuple (s_t, a_t, r_t, s_{t+1}) was stored in the replay buffer. When the replay buffer was full the oldest samples were discarded. At each time step the actor and critic were updated by sampling a minibatch uniformly from the buffer. Because DDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions.

(2) Target network

Directly implementing Q learning with neural networks proved to be unstable in many environments. Because the network $Q(s,a|\theta^Q)$ being updated is also used in calculating the target value, the Q update is prone to divergence. A solution to this problem is to use a separate network (the target network) for calculating the target values. In the DDPG algorithm, two target networks, $Q'(s,a\,|\,\theta^{Q'})$ and $\mu'(s\,|\,\theta^{\mu'})$, were created for the main critic and actor networks, respectively. They are identical in shape to the main networks but have different network weights θ'. The weights of these target networks are updated by having them slowly track the learned networks: $\theta' = \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. This means that the target values are constrained to change slowly, greatly improving the stability of learning.

The full DDPG algorithm is listed in Algorithm 1. DDPG starts by initializing the replay buffer as well as its actor, critic, and corresponding target networks. For each episode step, an action is then selected according to the exploratory policy. Afterwards, the reward r_t and new state s_{t+1} are observed and the information is stored in the replay memory D.

During training, minibatches are sampled from replay memory. The critic is then updated with the same loss function as in *DQN*. In the next step the actor is updated by performing a gradient ascent step on the sampled policy gradient. Finally, the target networks with weights $\theta^{Q'}$ and $\theta^{\mu'}$ are slowly moved in the direction of the updated weights of the actor and critic networks.

Algorithm 1 Deep deterministic policy gradient (*31*)

Randomly initialize critic network $Q(s,a \mid \theta^Q)$ and actor $\mu(s \mid \theta^\mu)$ with weights θ^Q and θ^μ.

Initialize target network $Q'(s,a \mid \theta^{Q'})$ and $\mu'(s \mid \theta^{\mu'})$ with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer *D*

for episode = 1 to M **do**

Initialize a random process N for action exploration

Receive initial observation state s_1

for t = 1 to T **do**

Select action $a_t = \mu(s_t \mid \theta^\mu) + \mathbb{N}_t$ according to the current policy and exploration noise $\mathbb{N}_t$.

Execute action a_t and observe reward r_t and observe new state s_{t+1}

Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer *D*

Sample a random minibatch of *N* transitions (s_i, a_i, r_i, s_{i+1}) from *D*

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

Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i \mid \theta^Q))^2$

Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a \mid \theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s \mid \theta^\mu)|_{s_i}$$

Update the target networks:

$$\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

end for

end for

3 DATA PREPARATION

3.1 Shanghai Naturalistic Driving Study

The data used in this study were collected in the Shanghai Naturalistic Driving Study (SH-NDS) (*14*) jointly conducted by Tongji University, General Motors, and the Virginia Tech Transportation Institute. The SH-NDS aims to learn more about the vehicle use, vehicle handling, and safety consciousness of Chinese drivers.

Five GM light vehicles equipped with Strategic Highway Research Program 2 (SHRP2) NextGen data acquisition systems (DAS) were used to collect real-world driving data. The three-year data collection procedure started in December 2012 and ended in December 2015. Driving data were collected daily from 60 licensed Shanghai drivers who, altogether, traveled 161,055 km during the study period.

3.2 Data Acquisition System

The DAS uses an interface box to collect vehicle controller area network (CAN) data, an accelerometer for longitudinal and lateral acceleration, a radar system that measures range and range rate to the lead vehicle and vehicles in adjacent lanes, a light meter, a temperature/humidity sensor, a global positioning system (GPS) sensor, and four synchronized camera views to validate the sensor-based findings.

As shown in Figure 2, the four camera views monitor the driver's face, the forward roadway, the roadway behind the vehicle, and the driver's hand maneuvers. The data collection frequency ranges from 10 to 50 Hz. The DAS automatically starts when the vehicle's ignition is turned on and automatically powers down when the ignition is turned off.

FIGURE 2 Four camera views for the SH-NDS.

3.3 Car-Following Periods Extraction

Car-following periods were automatically extracted for this analysis from the massive volume of collected data by applying a car-following filter described in a previous study (*14*). A car-following period was extracted if the following criteria were met simultaneously:

- Radar target's identification number > 0 and remaining constant: this criterion guaranteed that the same LV was being detected.
- 7m < range < 120m, and speed of the research vehicle > 5m/s: these two criteria eliminated free-flow and traffic jam conditions.
- −2.5m < lateral distance < 2.5m: this criterion guaranteed that the following and lead vehicles were driving in the same lane.
- −2.5m/s < relative speed < 2.5m/s: this criterion eliminated cases where the subject vehicle was rapidly closing in or falling back from the LV (unstable state).
- Length of car-following period > 15s: this criterion guaranteed that the car-following persisted long enough to reach a stable state.

The results of the automatic extraction process were confirmed with video viewed by an analyst to ensure validity of the car-following periods selected for analysis. The roadway type was identified during the validation phase.

This study focused on drivers' car-following behavior on roadways with limited access. Therefore, twenty drivers' car-following periods on expressways and freeways were used. For each driver, 100 car-following periods were randomly selected; 70 of those periods were randomly selected for model calibration and 30 for model validation. The total 2,000 selected car-following periods represented 827 minutes of driving.

4 PROPOSED APPROACH

Because the acceleration of a vehicle is continuous, a deep RL method called deep deterministic policy gradient (DDPG) (*31*), which performs well in continuous action space, was used. In this section, the proposed approach to modeling car-following behavior using DDPG is explained.

4.1 State and Action

At a certain time step *t*, the state of a car-following process is described by the following vehicle speed $V_n(t)$, spacing $S_{n-1,n}(t)$, and relative speed $\Delta V_{n\text{-}1,n}(t)$. The action is the longitudinal acceleration of the following vehicle $a_n(t)$. The acceleration was confined between -3 m/s^2 and 3 m/s^2 based on the observed acceleration of all the car-following events. With state and action at time step *t*, a kinematic point-mass model was used for state updating:

$$\begin{aligned} V_n(t+1) &= V_n(t) + a_n(t)\cdot\Delta T \\ \Delta V_{n\text{-}1,n}(t+1) &= V_{n-1}(t+1) - V_n(t+1) \\ S_{n\text{-}1,n}(t+1) &= S_{n-1,n}(t) + \Delta V_{n-1,n}(t+1)\cdot\Delta T \end{aligned} \quad (2)$$

where ΔT is the update time interval, set as 0.1s in this study, and $V_{n\text{-}1}$ is the velocity of LV, which is externally inputted.

4.2 Simulation Setup

A simple car-following simulation environment was implemented to enable the RL agent to interact with the environment through a sequence of states, actions, and rewards. The NDS data contains velocities of both the lead and the following vehicles. These data allow for a direct comparison between the measured driver behavior and trajectories simulated by an RL agent with the lead vehicle serving as externally controlled input.

Initialized with the empirically given following vehicle speed, spacing and velocity differences, $V_n(t=0) = V_n^{data}(t=0)$ and $S_{n-1,n}(t=0) = S_{n-1,n}^{data}(t=0)$, the RL agent is used to compute

the acceleration $a_n(t)$. The future states of the following car are then generated iteratively based on the state-updating rules defined in section 4.1. The simulated spacing can be directly compared to the empirical spacing provided by the NDS data to calculate reward value and simulation errors. The state is reset to the value in the empirical dataset when the simulating car-following event terminates at its maximum time step.

4.3 Evaluation Metric and Reward function

As suggested in Punzo and Montanino (*32*), the empirical and simulated inter-vehicle spacings were compared in order to evaluate the performance of a car-following model. Specifically, the root mean square percentage error (RMSPE) of spacing was used as the evaluation metric to consider relative errors:

$$RMSPE = \sqrt{\frac{\sum_{i=1}^{N}(S_i^{sim} - S_i^{obs})^2}{\sum_{i=1}^{N}(S_i^{obs})^2}} \quad (3)$$

where *i* denotes observation, S_i^{sim} is the *i*th modeled spacing, S_i^{obs} is the *i*th observed spacing, and *N* is the number of observations.

In RL, the reward function *r*(*s*, *a*), is used as a training signal to encourage or discourage behaviors in the context of a desired task. The reward provides a scalar value reflecting the desirability of a particular state transition that is observed by performing action *a* starting in the initial state *s* and resulting in a successor state s'. For the task of modeling drivers' car-following behavior, because the aim is to minimize the difference between values of simulated and observed spacing, the reward at time step *t* is provided by:

$$r_t = \log\left(\left|\frac{S_{n-1,n}(t) - S_{n-1,n}^{data}(t)}{S_{n-1,n}^{data}(t)}\right|\right) \quad (4)$$

where $S_{n-1,n}(t)$ is the simulated inter-vehicle spacing in the RL environment at time step *t*, and S_t^{obs} is the observed spacing in the real data set at time step *t*.

4.4 Network Architecture

Two separate neural networks were used to represent the actor and critic, respectively. At time step *t*, the actor network takes a state $s_t = (v_n(t), \Delta v_{n-1,n}(t), \Delta S_{n-1,n}(t))$ as input and outputs a continuous action: the following vehicle acceleration $a_n(t)$. The critic network takes as input a state s_t and an action a_t and outputs a scalar *Q*-value $Q(s_t, a_t)$.

As shown in Figure 3, both the actor and critic networks have three layers: an input layer taking the input signals to the whole neural network, an output layer generating the output signal,

and a hidden layer containing 30 neurons between the former two layers. We had tested even deeper neural networks, but experiment results showed that considering neural networks deeper than one hidden layers was unnecessary for our problem, which has only three or four input variables.

For the hidden layers, each neuron has a Rectified Linear Unit (RLU) activation function that transfers its input to its output signal. The RLU function computes as $f(x)=\max(0,x)$ and has been shown to accelerate the convergence of network parameter optimization significantly (*33*). The final output layer of the actor network used a tanh activation function, which maps a real-valued number to the range [-1, 1], and thus can be used to bound the output actions.

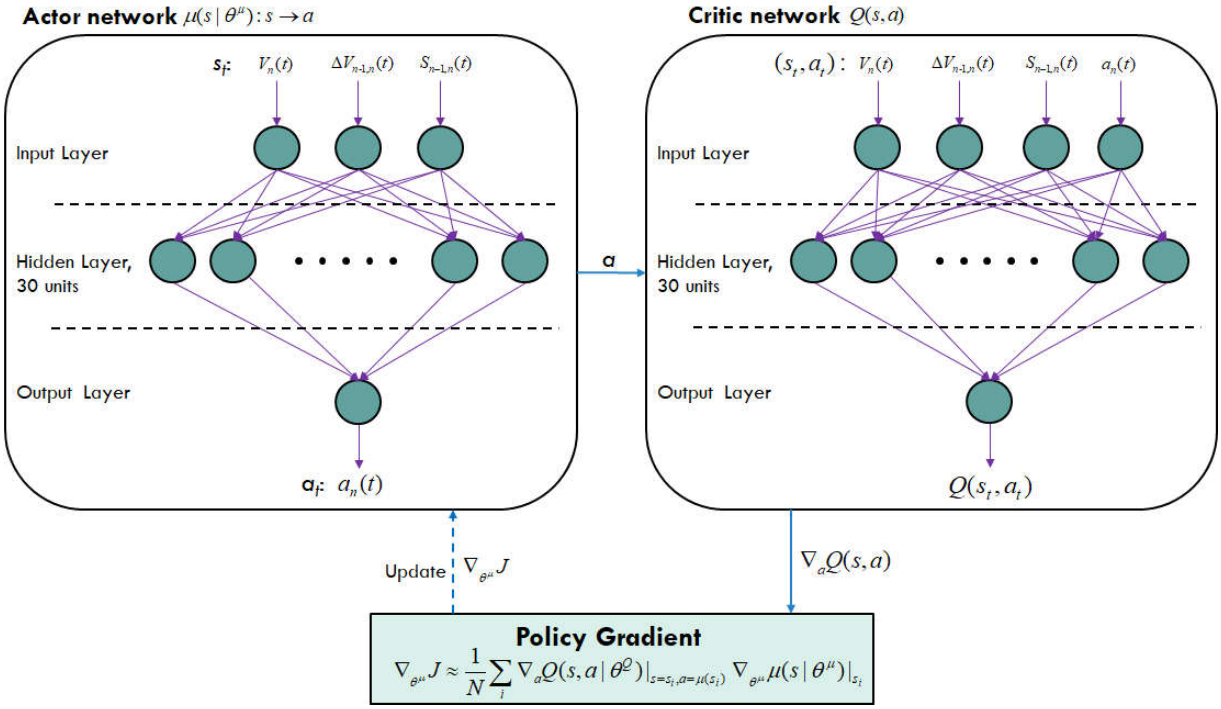


FIGURE 3 Architecture of the actor and critic networks.

4.5 Network Update and Hyperparameters

At each learning step, the weight coefficients of the critic network were updated by the stochastic gradient descent algorithm together with the adaptive learning rate trick Adam (*34*) to minimize the loss function $L=\frac{1}{N}\sum_i (y_i - Q(s_i,a_i|\theta^Q))^2$. As shown in Figure 3, the actor network parameters were updated as follows: on a forward pass, the actor's output is passed forward into the critic and evaluated. Next, the estimated Q-Value is back-propagated through the critic, producing gradients $\nabla_a Q(s,a)$ that indicate how the action should change in order to increase the Q-Value. On the backwards pass, these gradients flow from the critic through the actor. An

update is then performed over only the actor's parameters θ^{μ} according to the gradient $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i} \nabla_{a} Q(s,a \mid \theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s \mid \theta^{\mu})|_{s_i}$.

The hyperparameters (parameters whose values are set prior to the beginning of the learning process) adopted are shown in Table 1. The values of these hyperparameters were selected according to Lillicrap et al. (*31*) and also by performing a test on a randomly sampled training dataset.

TABLE 1 Hyperparameters and Corresponding Descriptions

Hyperparameter	Value	Description
Learning rate	0.001	Learning rate used by Adam
Discount factor	0.99	Discount factor gamma used in the Q-learning update
Minibatch size	32	Number of training cases over which each stochastic gradient descent update is computed
Replay memory size	7000	Number of training cases in replay memory
Replay start size	7000	Random policy run for this number of time steps before learning starts; the resulting experience is used to populate the replay memory
Soft target update τ	0.001	Update rate of target networks

4.6 Exploration Noise of Action

A major challenge of learning in continuous action spaces is exploration. An exploration policy was constructed by adding noise sampled from a noise process to the original actor policy. An Ornstein-Uhlenbeck process (*35*) with $\theta = 0.15$ and $\sigma = 0.2$ were used as suggested by Lillicrap et al. (*31*). The Ornstein-Uhlenbeck process models the velocity of a Brownian particle with friction, which results in temporally correlated values centered around zero. The temporally correlated noise enables the agent to explore well in physical environments that have momentum.

5 TRAINING AND TEST

5.1 Test Procedure

The car-following models were calibrated/trained at an individual-driver level, i.e., the calibration process was repeated for each driver, and each driver had his/her own set of parameters. As shown in Figure 4, the calibration and validation proceeded as follows for each driver:

1) Seventy (70%) of the total 100 car-following periods were randomly selected as the calibration/training dataset;
2) Car-following models were calibrated or trained based on the calibration dataset, outputting model parameters;
3) Two levels of validation—intra-driver and inter-driver—were conducted for the

calibrated/trained model parameters;

A. Intra-driver validation aimed to assess how the calibrated models would perform when applied to car-following periods that belonged to the investigated, or calibration, driver, but that were not used for calibration. That is, the remaining 30 car-following periods of the calibration driver were used for intra-driver validation. Intra-driver validation error was used to measure the trajectory reproducing accuracy of a car-following model.

B. Inter-driver validation aimed to assess how the calibrated models would perform when applied to other drivers. Inter-driver validation was repeated for each of the remaining 19 validation drivers, and all 100 car-following periods of the driver were used in each repetition. Inter-driver validation error was used to measure the generalization capability a car-following model.

4) For the DDPG car-following model, a model originally trained with data from driver A, was then retrained with data from driver B. The error reduction induced by the retraining was used to show how the DDPG model can adapt to different drivers.

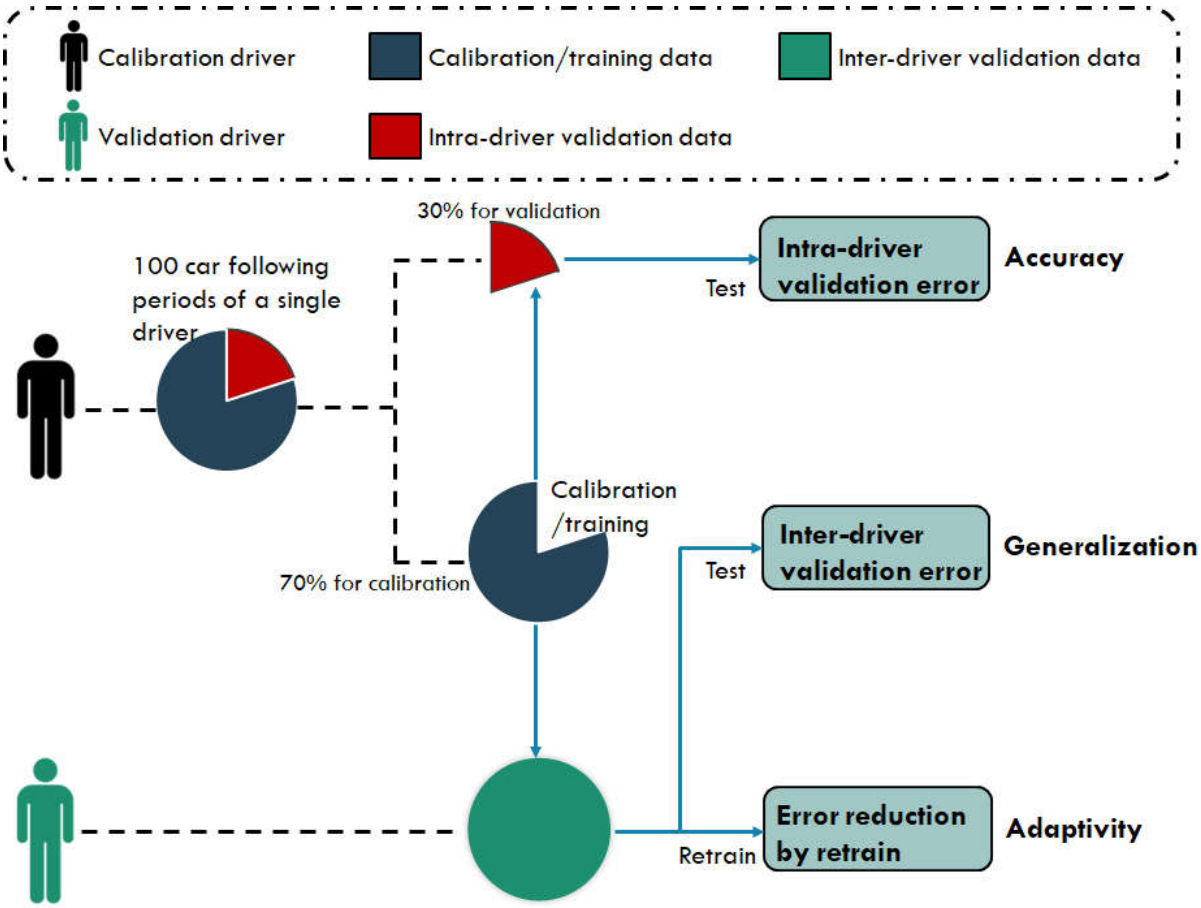


FIGURE 4 Schematic representation of the calibration and validation process for a single driver.

5.2 Calibrating Traditional Car-Following Models

Calibration of car-following models involves finding a set of model parameters that minimize the difference between values of simulated and observed variables (*36*). In this study, the RMSPE of spacing defined in Equation (*3*) was used as the error measure, and a genetic algorithm (GA) was implemented to find the optimum parameter values that minimize the errors. Genetic algorithms are widely used for calibrating microscopic traffic-simulation models because they can avoid local minima and reach the global optimum with a stochastic global search method applicable to constrained and unconstrained optimization problems (*37*).

The Genetic Algorithm Toolbox in MATLAB® was used to calibrate the traditional car-following models in this study. The relevant GA parameters for calibration of the GHR, Gipps, and IDM models were specified as follows: population size was 300, maximum number of generations was 300 and number of stall generations was 100. Because the number of parameters of the Wiedemann model (13 parameters) was significantly larger than in other models, the population size, maximum number of generations, and number of stall generations were set as 500, 1300, and 150, respectively.

The algorithm calculates the relative weighted average change in the fitness function value over stall generations, and if the change is less than function tolerance (10^{-6} in this study), the algorithm stops. The maximum number of generations controls the number of iterations.

Because the GA uses a stochastic process, it finds slightly different solutions in each optimization run. To find a solution closer to the global optimum, the optimization process was repeated 12 times for each driver, and the set of parameters with the minimum error (i.e., RMSPE) was selected. For detailed information about the calibration, readers can refer to Zhu et al. (*14*).

5.3 Training DDPG Car-Following Model

For each driver, 70 car-following periods were randomly selected to train the DDPG car-following model, and the remaining 30 periods were used as testing data. For each episode of training, the 70 training car-following periods were simulated sequentially by the RL agent. The state was initialized according to the empirical data whenever a new car-following period was to be simulated. The RMSPE of spacing on both training and testing data was calculated during training. The training was repeated for 60 episodes for each driver, and the RL agent that generated the smallest sum of training and testing errors was selected. Figure 5 shows the reward and error curves for a randomly selected driver. As can be seen, the performance of the DDPG model starts to converge when the training episode reaches 20.

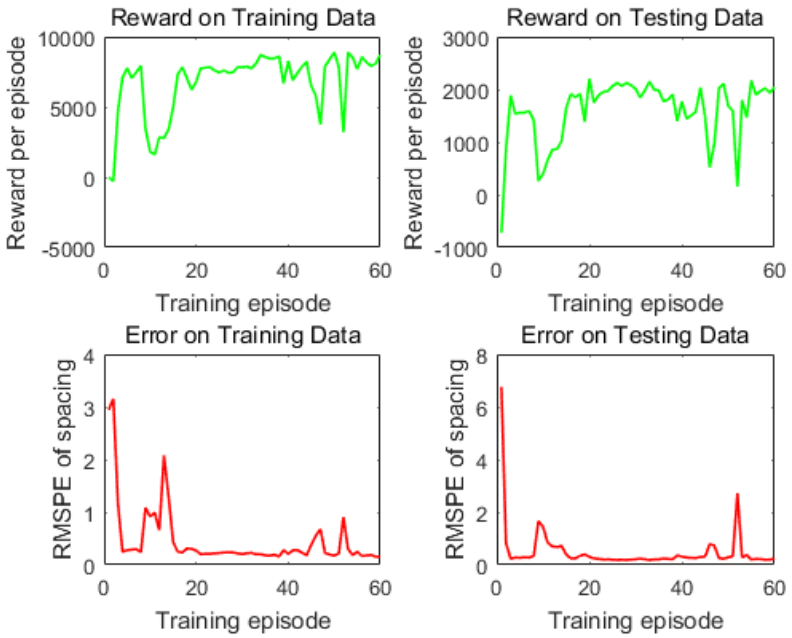


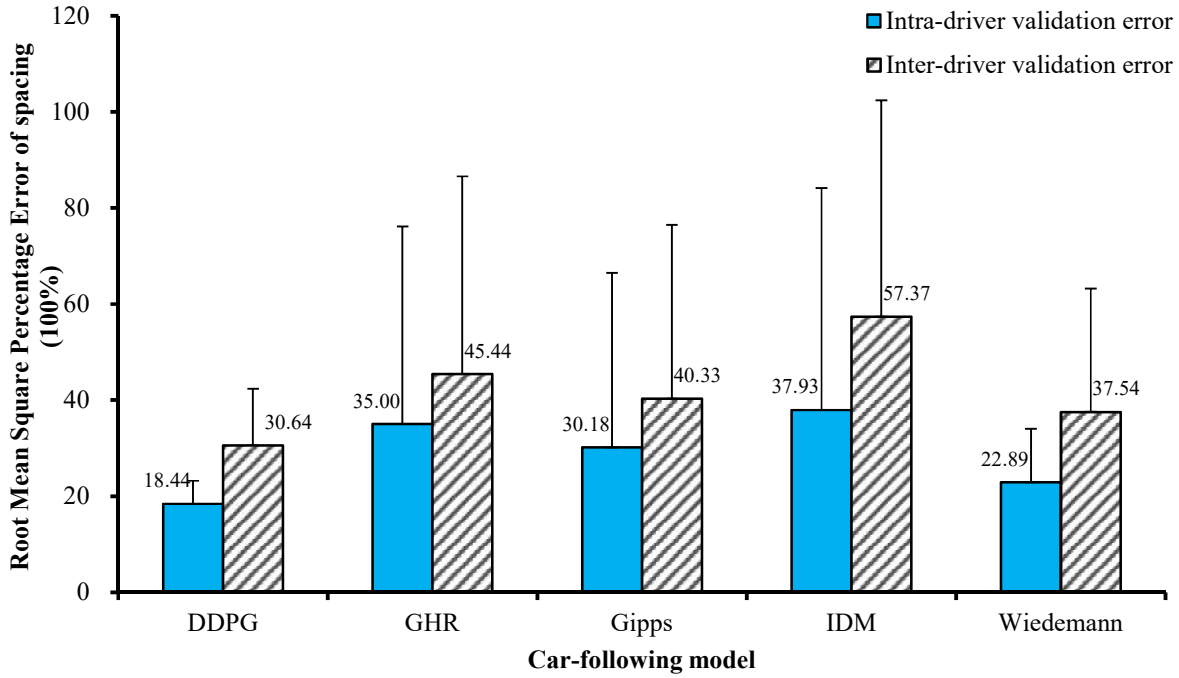
FIGURE 5 Reward and error curves during training for a selected driver.

6 RESULTS

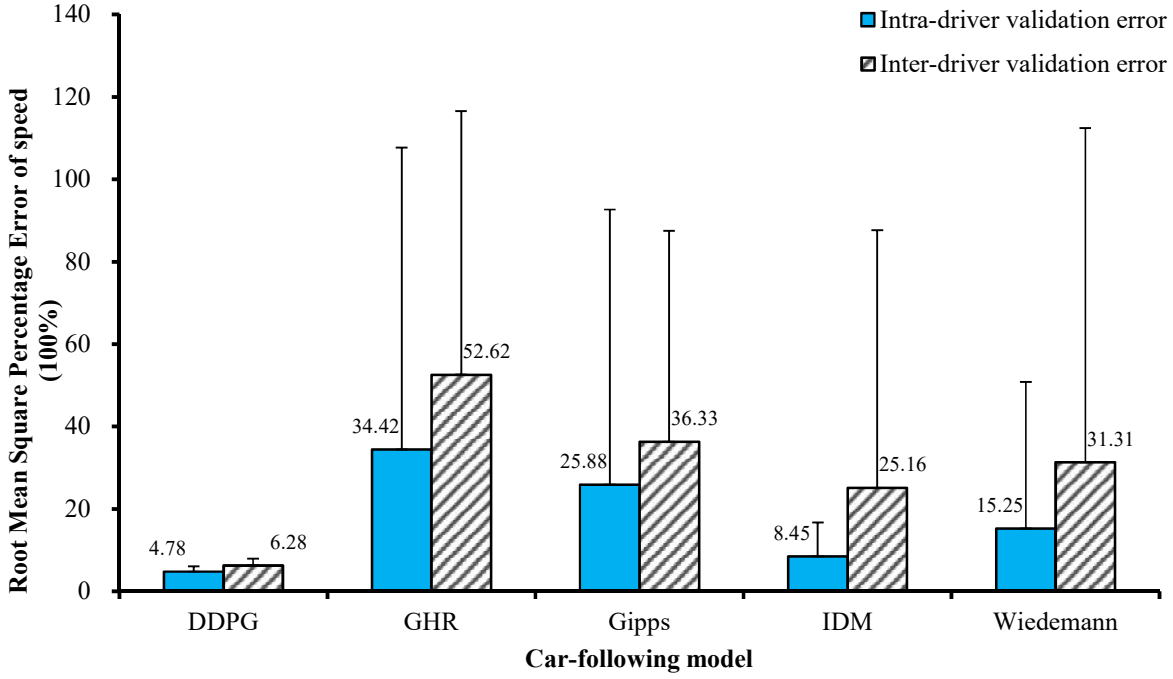
6.1 Trajectory Reproducing Accuracy

To examine the driving trajectory reproducing accuracy of the proposed DDPG car-following model, we compared the RMSPEs of spacing and speed in the intra-driver validation dataset when different models are used. The RMSPE of speed was defined as similar to that of spacing. The speed results are particularly meaningful since this variable was not used to train the models, and for this reason is an indicator of a model's ability to capture the system dynamics.

Figure 6 presents the mean values and standard deviations of the intra-driver validation errors for the five models. The DDPG model outperformed all of the investigated traditional car-following models, as evidenced by its having the lowest mean and standard deviation of errors on both spacing and, especially, speed.



(a) Spacing



(b) Speed

FIGURE 6 Mean standard deviation (line over the bar) of intra- and inter-driver validation errors on (a) spacing and (b) speed for the five models.

To illustrate the spacing and speed reproducing errors, we randomly selected one car-following period in the NDS dataset and calibrated it with the five models. Figure 7 shows the observed spacing and speed, and the spacing and speed replicated by the five models. The DDPG model (in red) best tracked the empirical spacing and speed variations.

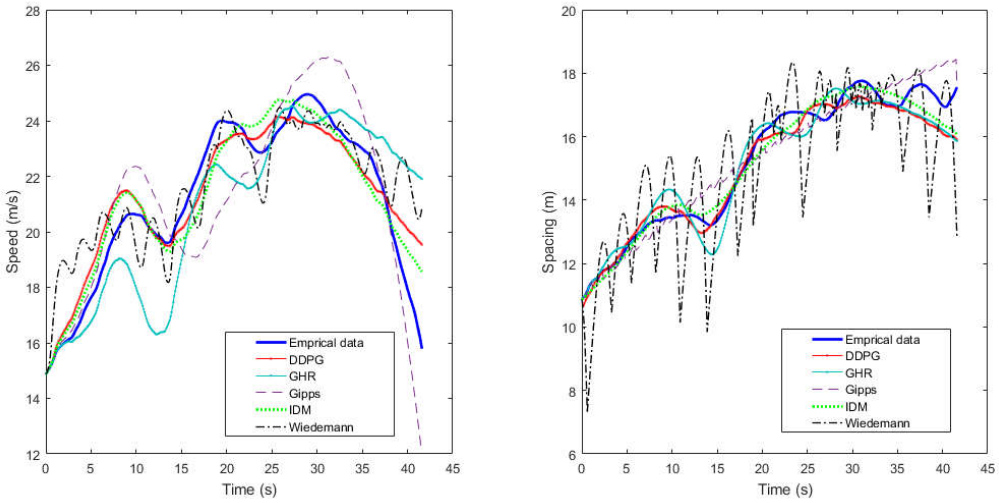


FIGURE 7 Spacing replicated by the five models.

To provide a more insightful interpretation of the reproducing-accuracy differences between the models, the observed and simulated trajectories for all car-following periods were plotted on a speed-spacing diagram. As presented in Figure 8, the DDPG model had the most similar replication of speed-spacing relationships as compared with empirical data, while other models such as GHR and Wiedemann generated a number of trajectories not observed in the empirical data (e.g., trajectories in low speed situations).

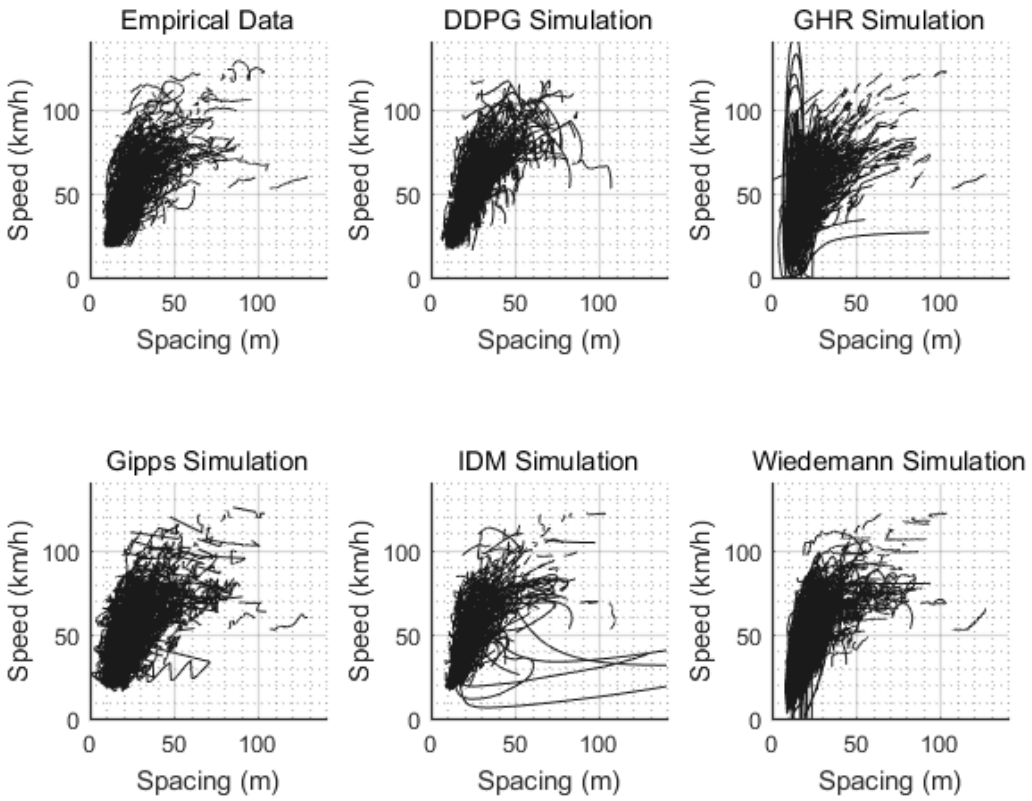
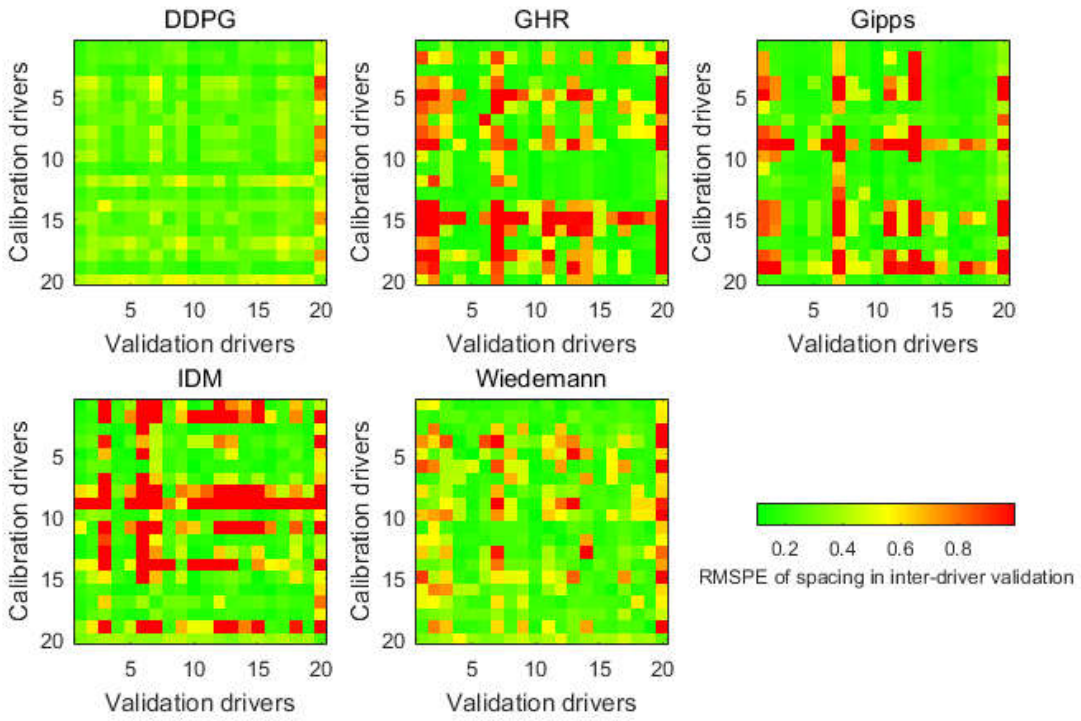


FIGURE 8 Speed-spacing relationships replicated by different models.

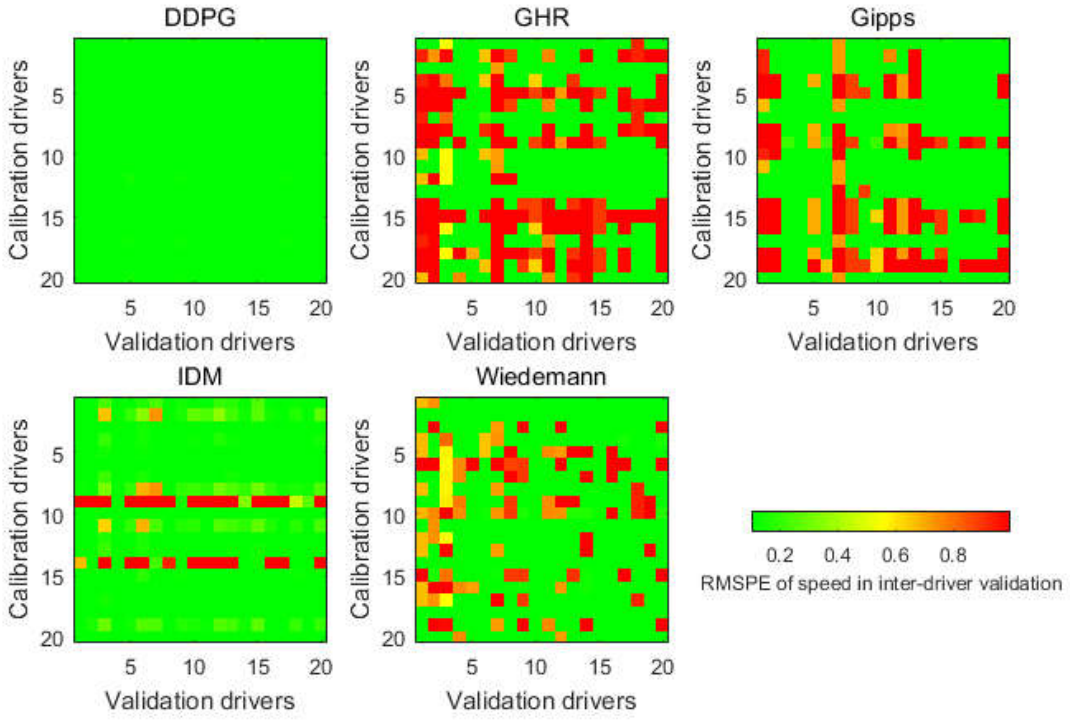
6.2 Generalization Capability

In the inter-driver validation phase, the car-following models were first calibrated or trained for an investigated driver and were then applied to each of the remaining 19 drivers. Therefore, two 20×20 error matrices (one for spacing and the other for speed) were obtained for each model, as shown in Figure 9. The DDPG model demonstrated the best generalization capability across different drivers, with almost all errors less than 40%.

Figure 6 present mean values and the respective standard deviations of the inter-driver validation errors for the five models. The DDPG model again outperformed all of the investigated traditional car-following models because it had the lowest means and standard deviations of errors on both spacing and speed.



(a) Spacing



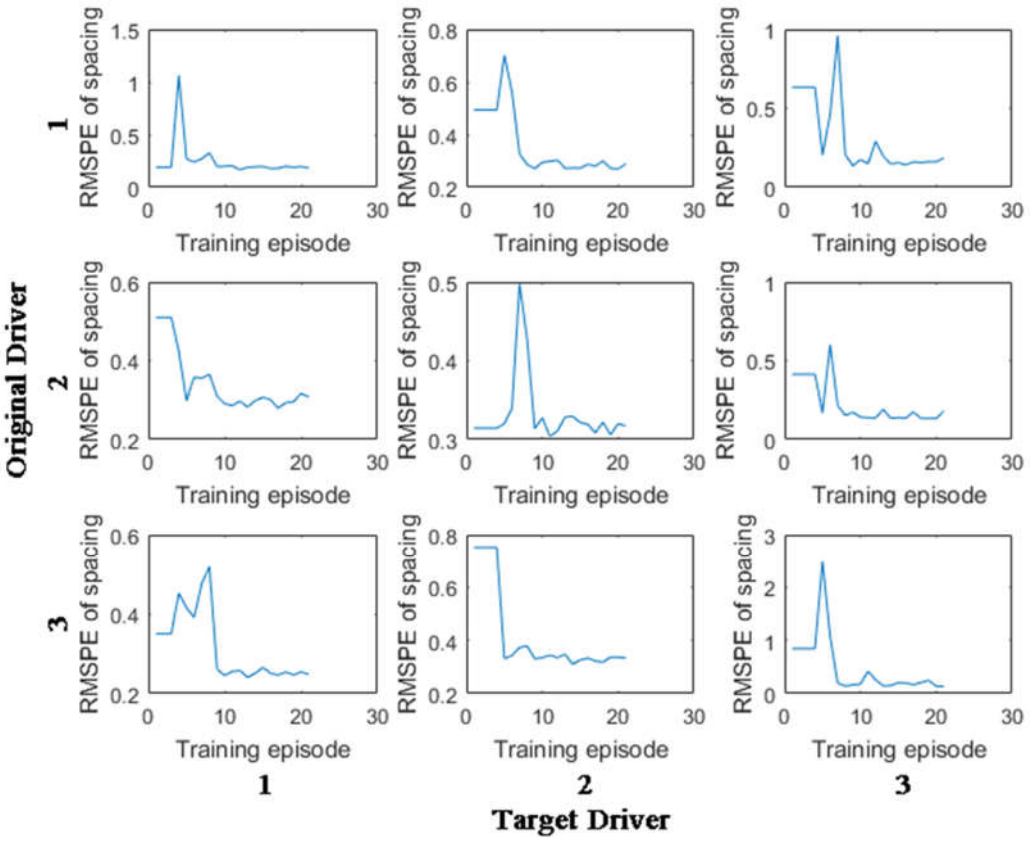
(b) Speed

FIGURE 9 Inter-driver validation errors of (a) spacing and (b) speed for all pairs of drivers.

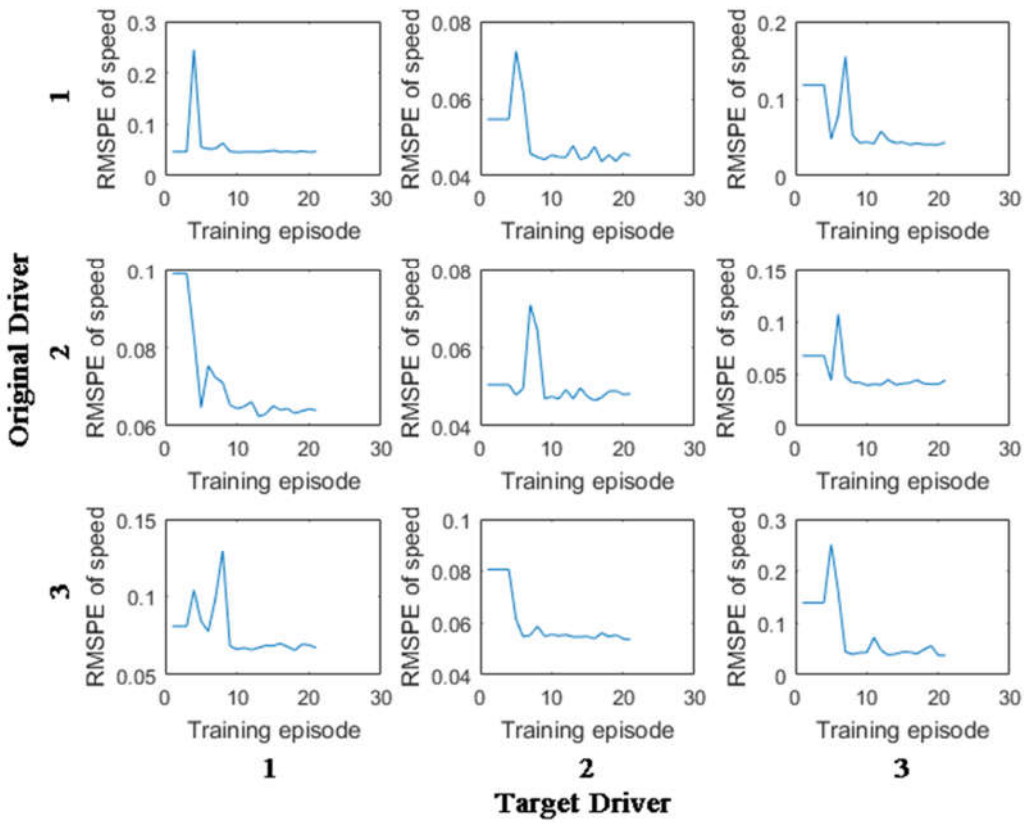
6.3 Adaptivity

This section investigates how a pre-trained DDPG model can adapt to a new driver when it is retrained with that driver's driving data. Three drivers were randomly selected, together with three corresponding DDPG models trained with their individual driving data. Each model corresponding to a driver was then retrained with driving data from other drivers, and, for completeness, with the corresponding driver. The retraining proceeded as follows for each model: 1) clear the old training cases in the replay buffer; 2) then feed the agent with driving data of a new driver and implement model training as described in Algorithm 1.

Figure 10 presents the changes of error in spacing and speed during the retraining process. The vertical axis refers to drivers on whose data the model was initially trained, and the horizontal axis refers to drivers on whose data the model was retrained. As shown in the figure, the error curves begin with a stable horizontal line. This is because the RL agent did not start learning (updating network parameters) until the replay buffer was filled with new experience data. During this time, the RL agent adopted a random policy to conduct exploration (trial and error) on the new set of data. Therefore, there is usually a sharp rise in error succeeding the stable horizontal line, which is caused by exploration noise. Then the errors decline and start to converge after 8 episodes of training. The final errors were smaller than those of initial models, which demonstrates that the DDPG model can adapt to different drivers when new data were fed.



(a) Spacing



(b) Speed

FIGURE 10 Changes of errors on (a) spacing and (b) speed for during retrain.

To summarize, the DDPG model outperformed the tested traditional car-following models in terms of trajectory-reproducing accuracy and generalization capability, and demonstrated its ability to adapt to different drivers when their driving data were fed in.

7 DISCUSSION AND CONCLUSION

This study proposes a framework for human-like autonomous car-following planning based on deep RL. The framework uses a deep deterministic policy gradient (DDPG) algorithm to learn a car-following model from historical driving data. Results show that the newly proposed DDPG car-following model outperforms the tested traditional models in terms of trajectory-reproducing accuracy and generalization capability, and demonstrates its ability to adapt to different drivers.

The following reasons may account for the better performance of the DDPG car-following model:

1) Rule-based traditional car-following models have strong built-in, specific assumptions about road and driver behavior, which may not be consistent with reality. In contrast, the DDPG model only generally assumes that drivers select actions in a fashion that maximizes cumulative future reward, which agrees with psychological and neuroscientific perspectives of human

behavior.

2) The small set of parameters used in parametric traditional car-following models may not generalize well to diverse driving scenarios, while the DDPG model uses two neural networks, one for policy generation and the other for policy improvement. Since neural networks have significantly more parameters, this would certainly improve the generalization capability of the DDPG model.

Although the network structure utilized in our study with only one hidden layer cannot be regarded as a truly "deep" neural network, it fully exploits the main idea of deep reinforcement learning methods and can be easily extended once more input variables are provided. Indeed, a possible improvement to our model is to take as inputs the velocities, velocity differences, and position differences that were observed in the last few time intervals, rather than instantaneous moment (*8*). The prediction capability, or memory effect, of human drivers would thus be considered, and more input variables would be given where neural networks with deeper or more complicated structure (such as recurrent neural networks) may exert their power.

Another potential improvement to our model is related to experience replay. Experience replay lets RL agents remember and reuse experiences from the past. In the currently adopted DDPG algorithm, experience transitions were uniformly sampled from a replay memory. However, this approach simply replays transitions at the same frequency that they were originally experienced, regardless of their significance (*38*). In future work, we can utilize a framework for prioritizing experience, so as to replay important transitions more frequently, and therefore learn more efficiently.

Moreover, since the DDPG algorithm can learn policies based on low-dimension inputs and also in an end-to-end fashion (*31*) (directly from raw pixel inputs to final actions), we would expect to compare the performance of these two paradigms in future studies. By comparing the low-dimension input-based DDPG car-following model, which accepts the relative position and speed information, with the end-to-end DDPG car-following model which accepts image information as input, the latent relationship between environment perception and control action may also be revealed.

As for the training time issue, first, the model only needs a few seconds to converge for data collected on trajectories lasting many seconds. Second, when an initial optimal policy has been learned, that policy is responsible for real-time reference acceleration generation, which is feasible because it is just a forward pass on the policy network; the policy updating work, however, does not necessarily need to be conducted in real-time.

It is worth noting that the proposed model is intended to reproduce a human-like driving behavior and that absolute safety issues are not addressed in our approach, but rather are within the province of a coupled collision avoidance system. The results of this study give only feasibility indications; the work must be continued in order to refine the approach and to validate it against field experiments. Moreover, prototypal implementation of the system and further studies with more emphasis on practical implementation aspects would clarify if those kinds of

framework can have efficiency and robustness characteristics.

To sum up, this study has shown the feasibility of a deep RL based car-following model for autonomous driving planning. Real world driving data from SH-NDS was used to train the model and compare its performance with that of traditional car-following models. Results show that this new model can 1) reproduce quite well a human-like car-following behavior, especially in terms of speed replicating; 2) generalize to different driving situations easily; 3) and adapt to different drivers by continuously learning. These results can contribute to the development of human-like autonomous driving algorithms. Moreover, this study demonstrates that data-driven modeling and reinforcement learning methodology can contribute to the development of traffic flow models and offer deeper insight into driver behavior.

ACKNOWLEDGEMENTS

The authors are grateful to Marilyn Morgan and Barbara Rau Kyle for their helpful edit. This study was sponsored by the Chinese National Science Foundation (51522810).

REFERENCES

1. Wei, J., J. M. Dolan, and B. A. Litkouhi. Learning-Based Autonomous Driver: Emulate Human Driver's Intelligence in Low-Speed Car Following. *Proceeding. of SPIE*, Vol. 7693, pp. 76930L–1.
2. Gu, T., and J. M. Dolan. Toward Human-Like Motion Planning in Urban Environments. *Intelligent Vehicles Symposium Proceedings*, 2014, pp. 350–355.
3. Lefèvre, S., A. Carvalho, and F. Borrelli. Autonomous Car Following: A Learning-Based Approach. *Intelligent Vehicles Symposium*, 2015, pp. 920–926.
4. Kuderer, M., S., and W. Burgard. Learning Driving Styles for Autonomous Vehicles From Demonstration. Robotics and Automation, *IEEE International Conference*, 2015, pp. 2641–2646.
5. Brackstone, M., and M. McDonald. Car-Following: A Historical Review. *Transportation Research F,* Vol. 2, 1999, pp. 181–196.
6. Simonelli, F., G. Bifulco, V. De Martinis, and V. Punzo. Human-Like Adaptive Cruise Control Systems through a Learning Machine Approach. *Applications of Soft Computing*, 2009, pp. 240–249.
7. Pipes, L. An Operational Analysis of Traffic Dynamics. *Journal of Applied Physics*, Vol. 24, No. 3, 1953, pp.274–281.
8. Wang, X., R., Jiang, L., Li, Y., Lin, X., Zheng, and F. Y. Wang. Capturing Car-Following Behaviors by Deep Learning. *IEEE Transactions on Intelligent Transportation,* Vol. 99, 2017, pp. 1–11.
9. Wang, J., L. Zhang, D. Zhang, and K. Li. An Adaptive Longitudinal Driving Assistance System Based on Driver Characteristics. *IEEE Transactions on Intelligent Transportation Systems*, Vol.14, No.1, 2013, pp. 1–12.

10. Hausknecht, M., and P., Stone. Deep Reinforcement Learning in Parameterized Action Space. International Conference on Learning Representations, 2016.
11. Mnih, V., K., Kavukcuoglu, D., Silver, A. A., Rusu, J., Veness, M. G., Bellemare, and S., Petersen. Human-Level Control through Deep Reinforcement Learning. *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533.
12. Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, and Dieleman, S. Mastering the Game of Go With Deep Neural Networks And Tree Search. *Nature*, Vol. 529 No.7587, 2016, pp. 484–489.
13. Hornik, K., M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, Vol. 2, No. 5, 1989, pp. 359–366.
14. Zhu, M., X. S. Wang, A, P, Tarko. Calibrating Car-Following Models on Urban Expressways for Chinese Drivers Using Naturalistic Driving Data. Transportation Research Board 96th Annual Meeting, Washington D.C., USA, 2017. pp. 1–12.
15. Chandler, R. E., R. Herman, and W. Montroll. Traffic Dynamics: Studies in Car-Following. *Operations Research*, Vol. 6, 1958, pp. 165–184.
16. Gazis, D.C., R. Herman, and R. W. Rothery. Nonlinear Follow-the Leader Models of Traffic Flow. *Operations Research,* Vol. 9, 1961, pp. 545–567.
17. Treiber, M., A. Hennecke, and D. Helbing. Congested Traffic States in Empirical Observations and Microscopic Simulations. *Physical Review E,* Vol. 62, 2000, pp. 1805–1824.
18. Bando, M., and K. Hasebe. A Dynamical Model of Traffic Congestion and Numerical Simulation. *Physical Review E,* Vol. 51, No. 2, 1995, pp. 1035–1042.
19. Helly, W. Simulation of Bottlenecks in Single-Lane Traffic Flow. In: Proceedings of the Symposium on Theory of Traffic Flow, Research Laboratories, General Motors, New York, 1959.
20. Gipps, P.G. A Behavioural Car-Following Model for Computer Simulation. *Transportation Research Part B,* Vol. 15, No. 2, 1981, pp. 105–111.
21. Wiedemann, R. Simulation des StraBenverkehrsflusses. In: Proceedings of the Schriftenreihe des tnstituts fir Verkehrswesen der Universitiit Karlsruhe, Germany, 1974.
22. Olstam, J.J. and A. Tapani. *Comparison of Car-following models*. VTI, Linköping, 2004.
23. Panwai, S. and H. Dia. Comparative Evaluation of Microscopic Car Following Behavior. *IEEE Transaction on Intelligent Transportation System*, Vol. 6 No. 3, September 2005, pp. 314–325.
24. Saifuzzaman, M., and Z.D. Zheng. Incorporating Human-Factors in Car-Following Models: A Review of Recent Developments and Research Needs. *Transportation Research Part C: Emerging Technologies*, Vol. 48, 2014, pp. 379–403.
25. May, A.D. *Traffic Flow Fundamentals.* Prentice Hall, Eaglewood Cliffs, New Jersey, 1990.
26. Ozaki, H. Reaction and Anticipation in Car-Following Behavior. *Transportation and Traffic Theory*. (C. F. Daganzo, ed.), Elsevier Science Publishers, Amsterdam, Netherlands, 1993,

pp. 349–362.
27. Treiber, M., and A. Kesting. *Traffic Flow Dynamics: Data, Models and Simulation*, Springer-Verlag, Berlin, Heidelberg, 2013.
28. Li, Y. Deep Reinforcement Learning: An Overview. 2017.
29. Sutton, R. S., and A. G. Barto. Reinforcement Learning: An introduction. Cambridge: MIT press, 1998.
30. Grondman, I., L Busoniu, G. A. Lopes, and R. Babuska. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 42, No. 6, 2012, pp. 1291–1307.
31. Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, and, Y. Tassa. Continuous Control with Deep Reinforcement Learning. International Conference on Learning Representations, 2016.
32. Punzo, V., and M. Montanino. Speed or Spacing? Cumulative Variables, And Convolution of Model Errors and Time in Traffic Flow Models Validation and Calibration. *Transportation Research Part B: Methodological*, Vol. 91, 2016, pp. 21–33.
33. Krizhevsky, A., I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
34. Kingma, D., and J. Ba. Adam: A Method for Stochastic Optimization. Computer Science, 2014, pp. 1412–6980.
35. Morikazu, T. On the Theory of the Brownian Motion. *Journal of the Physical Society of Japan*. Vol. 13, No. 11, 2007, pp. 1266–1280.
36. Kesting, A., and M. Treiber, Calibrating Car-Following Models by Using Trajectory Data: a Methodological Study. *Transportation Research Record: Journal of the Transportation Research Board*, No. 2088, Transportation Research Board of the National Academies, Washington, D.C., 2008, pp. 148–156.
37. Saifuzzaman, M., Z.D. Zheng, M.M. Haque, S. Washington. Revisiting the Task-Capability Interface Model for Incorporating Human Factors into Car-Following Models. *Transportation Research Part B: Methodological*, Vol. 82, 2015, pp. 1–19.
38. Schaul, T., J. Quan, I. Antonoglou, and D. Silver. Prioritized Experience Replay. International Conference on Learning Representations, 2016.