Dynamic Car-following Model Calibration with Deep Reinforcement Learning

Htet Naing[1]

I. INTRODUCTION

This document contains supplementary materials of our paper, *"Dynamic Car-following Model Calibration with Deep Reinforcement Learning"*, submitted to the IEEE International Conference on Intelligent Transportation Systems (IEEE ITSC 2022). The paper provides all related supplementary information regarding literature review, model formulation, model implementation, etc. Moreover, model training parameters (or hyperparameters) that were used in calibrating the parameters of Intelligent Driver Model (IDM), are also reported. Two deep reinforcement learning (DRL) architectures: Deep Q-Network (DQN) [1], [2] and Deep Deterministic Policy Gradient (DDPG) [3], are adapted for this calibration task. All the DRL models used in our paper and the car-following simulation environment were implemented in Python language with the help of Pytorch [4]. The baseline model, Genetic Algorithm (GA), was implemented using a Python library distributed on Pypi [5].

II. REVIEW ON MODEL CALIBRATION WITH DRL

A relatively new research direction has emerged to tackle the issues inherent in the current state-of-the-art calibration approaches. Under this direction, researchers have developed a variety of deep reinforcement learning-based calibration methods [6]–[9]. Generally, the DRL-based models learn the desired optimal behaviour by interacting through a simulation environment. Unlike deep learning models, they are more suitable for model calibration task because they do not require explicitly labelled ground-truth parameter values as their training data, except an appropriate simulation environment and observation data.

In [6], an extended DRL-based calibration method has been proposed to calibrate the diagnostics-related parameters of turbofan engines in real-time. The proposed method outperformed both Unscented Kalman Filter (UKF) and Multi-layer Perceptron (MLP) in terms of prediction accuracy, robustness to uncertainty and inference time. The same model was used in [8] to calibrate battery models and compared against similar baseline methods. In [7], a hierarchical parameter tuning process involving Deep Deterministic Policy Gradient (DDPG) is proposed to calibrate power system models. The experimental results show that their proposed model outperforms other calibration methods such as Bayesian Optimization (BO) and Random Search (RS). Finally, in [9], Deep Q-Network (DQN) was used to calibrate the speed limit of different types of vehicles in a microscopic traffic simulator for reconstructing incomplete vehicle trajectories for unmonitored road segments.

[1]Htet Naing is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore

III. DQN FOR MODEL CALIBRATION

An attempt to calibrate the speed limit of different types of vehicles in a microscopic traffic simulator (SUMO [10]) with the help of Deep Q-Network (DQN) has been made in [9]. However, their objective is different from ours in that they aim to calibrate the speed limit of different vehicle types in the simulator for vehicle trajectory reconstruction based on aggregated traffic features. Furthermore, it neither mentioned anything specific to the calibration of the car-following model used in the simulator, nor attempted to perform dynamic calibration for each individual vehicle. Nevertheless, inspired by this work, DQN is selected, as our initial step, to be used in our framework, and it is shown in Figure 1.

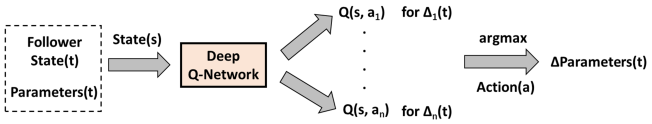

Fig. 1. Deep Q-Network for dynamic car-following model calibration

1) Agent State Space: For DQN, the agent's environment state, for a given time step t, is defined as s_t which consists of the current follower state and the car-following model (CFM) parameter values. In order to keep our follower state representation generic, the set of additional features veh_t^a is defined to be empty. However, future works may consider adding more relevant features as necessary according to their application context.

2) Agent Action Space: The inherent limitation of DQN is that it can only handle only discrete action spaces. The implication of this limitation on our calibration task is that we are confined to define a fixed set of discretized $\Delta\lambda$ values in spite of all IDM parameters being continuous. It is also obvious that there can be many possible combinations of $\Delta\lambda$ values with more number of parameters considered for calibration. To tackle this issue, only a reduced set of λ with three most important parameters according to [11]: acc_{max}, T, δ, are chosen for calibration. Then, the action space for DQN can be defined as:

$$A_{DQN} = \{\Delta^- acc_{max}, \Delta^0 acc_{max}, \Delta^+ acc_{max}\} \times \{\Delta^- T, \Delta^0 T, \Delta^+ T\} \times \{\Delta^- \delta, \Delta^0 \delta, \Delta^+ \delta\} \quad (1)$$

where $\{\Delta^{-}\lambda, \Delta^{0}\lambda, \Delta^{+}\lambda\} \in \Delta\lambda;\ \lambda \in \{acc_{max}, T, \delta\}$ are the parameter change values with their change direction: decrease $(-)$; no change (0); increase $(+)$, respectively. A_{DQN} is the Cartesian product of these possible Δ value sets. In this paper, these values are fixed to {-0.01, 0, 0.01} for Δacc_{max}; {-0.1, 0, -0.1} for ΔT; and {-1, 0, 1} for $\Delta\delta$, respectively. This results in the DQN action space with 27 dimensions. Initially, the IDM parameters λ are defined using their default parameters. At each simulation step, DQN selects the best calibration action (with the highest Q-value) out of these 27 possible sets. The three selected IDM parameters are then updated with their corresponding $\Delta\lambda$ subject to $[LB_{\lambda}, UB_{\lambda}]$. If there is any parameter value exceeds their valid intervals, it is clipped to be within their respective bounds.

3) Follower State Update: Based on the newly calibrated parameters values, the follower state is updated by advancing to the next simulation step using the IDM predicted dynamics along with the ballistic update integration scheme.

4) Agent Reward: After observing the new environment state, the DQN agent receives the reward for executing a particular action. A reward is a scalar signal generated by a specific reward function. The reward function should be designed carefully depending on the type of behaviour to which our agent should be trained to converge. The purpose of a calibration task is to adjust a set of parameters that can help the response of the underlying physics-based model match, as closely as possible, to the observation data. Thus, an appropriate measure of performance should be incorporated in crafting the reward function.

For car-following model calibration, this is analogous to the selection of performance measure for a typical optimization algorithm. Among popular performance measures chosen for the calibration task (i.e., spacing, speed or acceleration), Punzo and et al. has proved empirically in [12] and theoretically in [13] that the choice of spacing (gap distance between a follower and its leader) as the calibration performance measure can produce more optimal results than that of vehicle speed. In their proof, it has been shown that the optimization over spacing, which is the cumulative sum of speed, is more robust than optimization over the speed itself [14]. By the same reasoning, this conclusion applies to acceleration as well. Thus, the reward signal at each simulation step (r_t) can be calculated through the reward function R, which is modelled as the negative exponential function of the squared error between the simulated gap (gap_t^{sim}) and the ground-truth gap (gap_t^{truth}). The intuition behind this reward function is that the lower the squared error, the higher the reward value. The maximum achievable reward for our agent in a given time step is a scalar 1 (which occurs when there is no difference between the simulated and actual gaps).

$$r_t = R(gap_t^{sim}, gap_t^{truth}) = e^{-Error_{data}} \tag{2}$$

$$Error_{data} = (gap_t^{sim} - gap_t^{truth})^2 \tag{3}$$

5) Agent Experience Replay Memory: An agent's transition experience gained through the interaction with the environment is usually summarized as a tuple, (s_t, a_t, s_{t+1}, r_{t+1}). These transitions are stored in a replay memory configured in advance with a predefined size. A batch of transitions are sampled randomly from this replay memory and used as training samples for the DQN agent.

6) Agent Training: Most of the notations used here follows the convention of the original DQN paper [1]. The optimal action-value function $Q^*(s, a)$ obeys an important identity as follows [1]:

$$Q^*(s,a) = \mathbb{E}\Big[r + \gamma \max_{a'} Q^*(s', a')\Big] \tag{4}$$

where s' and a' are potential next state and action respectively, after taking action a at the state s. Consequently, the DQN agent can employ any suitable deep neural network architecture as an approximator, $Q(.|\theta)$ (also known as "Q-Network") and trains the parameters θ of this approximator so that it can learn how to estimate $Q^*(s, a) \approx Q^*(s, a|\theta)$ optimally [1]. At each simulation step, the agent samples a batch of transition experiences from the replay memory to update θ by minimizing the following training error loss.

$$Loss_{DQN} = \mathbb{E}\Big[(Q^*(s,a) - Q(s,a|\theta))^2\Big] \tag{5}$$

where $Q^*(s, a) \approx r + \gamma \max_{a'} Q(s', a'|\theta^{-})$, implying that the target optimal Q-value is also approximated by a time-lagged copy of the main Q-Network $Q(.|\theta^{-})$, namely "target Q-Network". All of the above derivations can be found in the original works [1], [2] for more details. With reference to them, the ϵ-greedy strategy is also adopted in this paper to encourage agent exploration behaviour. This means that the DQN agent selects a random action with ϵ probability and chooses the optimal action with $1 - \epsilon$ probability. This ϵ value will be decayed during the training process as the agent behaviour improves with more training episodes.

7) Agent Testing: At the end of each training episode, an updated set of IDM parameters are saved so that they can be selected later for testing. The agent performance for car-following model calibration can be tested in two ways as follows:

- Static (offline) calibration: The final set of calibrated IDM parameters are used for individual follower trajectory simulation. In general, we provided three heuristic ways to select the final set of parameters for testing – 1) the best set of parameters that gives the highest scores during the training process, 2) the average of the last-K parameter sets generated after each of the last-K training episodes, and 3) the average of the parameter sets selected within the local window size (W) with the best set of parameters being at the center of the window. No Q-Network is deployed during the test stage.
- Dynamic (online) calibration: Each test follower is initialised with the final set of calibrated parameters chosen earlier. The pre-trained Q-Network is deployed during the test stage in order to perform online calibration of the IDM parameters for each follower trajectory. The

DQN agent selects the best calibration action dynamically at each simulation step based on the environment state. No learning (training) is involved during the test stage.

This dynamic model calibration capability of DRL, without relying on any online optimization or learning during the deployment stage, is the major advantage of DRL over other widely used optimisation algorithms such as GA. Although the DRL training process may take longer than others due to its inherent trial-and-error learning approach, the computational speed of the DRL model during the inference stage is considerably fast subject to the model complexity.

IV. IDM Related Parameters

Before the DRL-based calibration process begins, the default set of IDM parameters λ is inferred from the existing studies [15]–[17], and used as the initial starting point for the calibration task. Since these λ values can only range within their valid intervals, their respective parameter bounds are inferred from the training datasets, the road speed limit of the study area, and most importantly, the acceleration/deceleration bounds are based on a systematic study conducted in [18].

The default parameter values are then defined as: [33.3, 1.6, 2, 0.73, 1.67, 4] for $[v_0, T, gap_{jam}, acc_{max}, dacc_{max}, \delta]$, respectively. These values were initialized at the beginning of the training process of both DQN and DDPG. Accordingly, the lower (LB_λ) and upper bounds (UB_λ) are:

- set to = [10, 33.3333] m/s for maximum desired velocity v_0.
- set to [0.3, 6] s for desired time headway T.
- set to [1, 5] m for jam spacing distance gap_{jam}.
- set to [0.28, 3.41] m/s^2 for maximum desired acceleration acc_{max}.
- set to [0.47, 3.41] m/s^2 for maximum desired deceleration $dacc_{max}$.
- set to [0, 10] for exponent parameter δ.

V. DRL Training Hyperparamters

The training hyperparameters are referred to [19] and [20] for DDPG and [1], [9] for DQN.

A. DQN Training Hyperparameters

The architecture of the DQN network used in our paper is shown in Figure 2. DQN is used to calibrate three IDM paramters: acc_{max}, T, δ in our paper. The hyperparamters used for DQN is defined as follows.

- Number of training episodes E is set to 150.
- Learning rate (α) used for Deep Q-Network parameter update during the training process is set to 0.0005.
- Discount factor (γ) used to discount future rewards is set to 0.95.
- Replay memory size used to store transition experiences is set to 100000.
- Learning start point is set to 20000, i.e., the agent only starts learning when there are 20000 transitions in the memory.
- Batch size for transition samples (B) used for training Q-Network parameters is set to 256.
- The exploration probability ϵ is set to 1.0 initially and is decreased after each simulation step by the amount, $\epsilon_{dec} = 0.00001$ until it reaches the $\epsilon_{min} = 0.01$.
- Target network update frequency is set to once per 30 training episodes.
- Loss function is set to mean squared error loss.
- Optimization algorithm is set to Adam.

1) Model Selection for Testing: As for testing the DQN agent, the best Q-Network saved after the training episode that achieves the highest score was used for dynamic calibration. Similarly, for the best set of IDM parameters generated after that particular episode was also used for both static calibration and parameter initialization for dynamic calibration.

B. DDPG Training Hyperparameters

The architecture of the DDPG network used in our paper is shown in Figure 3. DDPG is used to calibrate all IDM parameters: $v_0, T, gap_{jam}, acc_{max}, dacc_{max}, \delta$ in our paper. The hyperparamters used for DDPG is defined as follows.

- Number of training episodes E is set to 150.
- Learning rate (α) used for the parameter update of both actor and critic networks during the training process is set to 0.0005.
- Discount factor (γ) used to discount future rewards is set to 0.95.
- Replay memory size used to store transition experiences is set to 100000.
- Learning start point is set to 20000, i.e., the agent only starts learning when there are 20000 transitions in the memory.
- Batch size for transition samples (B) used for training actor and critic network parameters is set to 256.
- OU-Noise parameters added for action exploration are set to 0.15 and 0.2 for θ_{OU} and σ_{OU} respectively. Unlike DQN exploration, the OU-Noise used here is not decayed throughout the training process. Nonetheless, based on the training performance scores, the DDPG agent seems to converge to optimal behaviour without decaying this noise.
- Soft target update rate τ used for updating target network parameters is set to 0.001.
- Loss function is set to mean squared error loss for critic network and negative average Q-values, for actor network.
- Optimization algorithm is set to Adam.

1) Model Selection for Testing: As for testing the DDPG agent, the last actor network saved after the end of training process was used for dynamic calibration. The average IDM parameter values taken from the last 30 training episodes was used for both static calibration and parameter initialization for dynamic calibration.

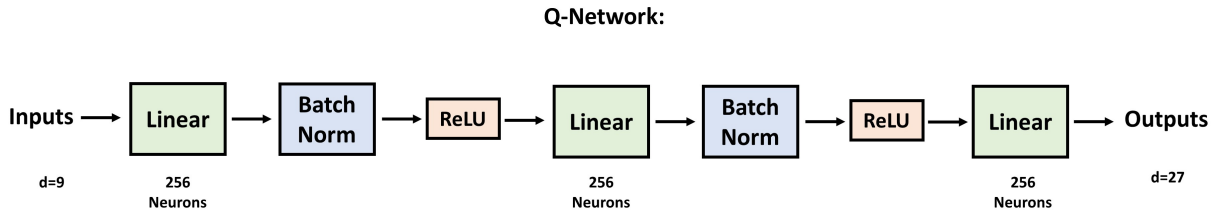

Fig. 2. DQN architecture used for car-following model calibration

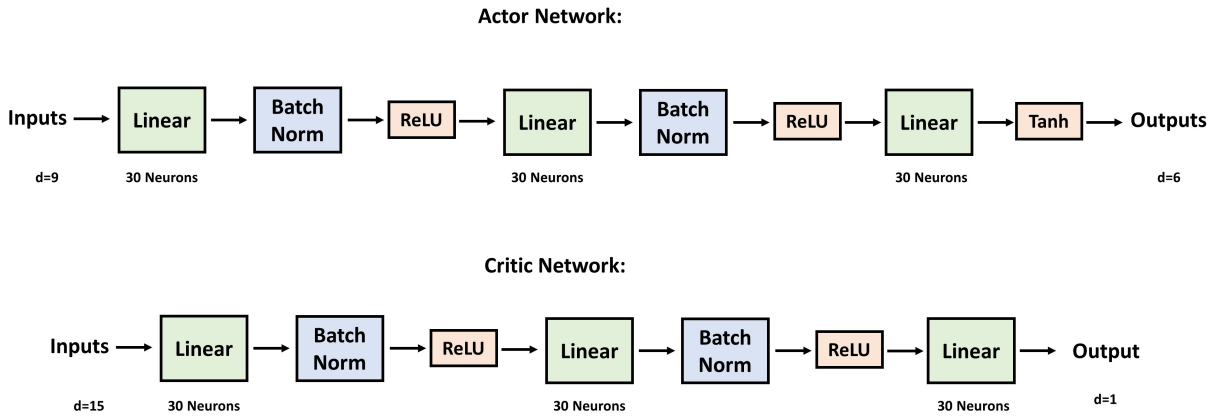

Fig. 3. DDPG architecture used for car-following model calibration

VI. Genetic Algorithm Training Parameters

The GA is used to calibrate all IDM parameters: $v_0, T, gap_{jam}, acc_{max}, dacc_{max}, \delta$ in our paper. The GA training parameters used for static (offline) calibration of the IDM parameters are described below. The above-mentioned IDM parameter bounds were also applied to GA as constraints during the optimisation process.

- Maximum number of generation is set to 150.
- Population size is set to 50.
- Mutation probability is set to 0.3.
- Elite ratio (for selecting the number of elites out of a given population) is set to 0.01.
- Crossover probability is set to 0.5.
- Crossover method is set to be uniform crossover.
- Proportion of the number of parents in the population is set to 0.3.
- Stopping criterion is set as follows. When the solution for the last 10 generations has not improved anymore, the GA is terminated.

All training follower-leader trajectories were used to calculate the fitness score at each generation. The spacing root mean squared error for all simulated trajectories was used as the minimization objective subject to the IDM parameter bounds. Despite 150 generations set for optimisation, the early convergence was observed at the generation 20 after which the algorithm was terminated. Another optimisation run without any early termination introduced was also tested, however, the final solution generated after completing 150 generations, was observed to underperform that of the first run with early termination.

1) Model Selection for Testing: As for testing the GA performance, the final set of parameters generated after the algorithm convergence was selected for static calibration.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[4] M. (a.k.a. Facebook), "Pytorch," https://pytorch.org, 2021.

[5] R. M. Solgi, "Genetic algorithm," https://pypi.org/project/geneticalgorithm/, 2020.

[6] Y. Tian, M. A. Chao, C. Kulkarni, K. Goebel, and O. Fink, "Real-time model calibration with deep reinforcement learning," *arXiv preprint arXiv:2006.04001*, 2020.

[7] L. Lin, W. Wu, Z. Shangguan, S. Wshah, R. Elmoudi, and B. Xu, "Hpt-rl: Calibrating power system models based on hierarchical parameter tuning and reinforcement learning," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 1231–1237.

[8] A. Unagar, Y. Tian, M. A. Chao, and O. Fink, "Learning to calibrate battery models in real-time with deep reinforcement learning," *Energies*, vol. 14, no. 5, p. 1361, 2021.

[9] X. Tang, B. Gong, Y. Yu, H. Yao, Y. Li, H. Xie, and X. Wang, "Joint modeling of dense and incomplete trajectories for citywide traffic volume inference," in *The World Wide Web Conference*, 2019, pp. 1806–1817.

[10] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic traffic simulation using sumo," in *2018 21st International*

Conference on Intelligent Transportation Systems (ITSC), 2018, pp. 2575–2582.

[11] V. Punzo, M. Montanino, and B. Ciuffo, "Do we really need to calibrate all the parameters? variance-based sensitivity analysis to simplify microscopic traffic flow models," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 184–193, 2014.

[12] V. Punzo, B. Ciuffo, and M. Montanino, "Can results of car-following model calibration based on trajectory data be trusted?" *Transportation research record*, vol. 2315, no. 1, pp. 11–24, 2012.

[13] V. Punzo 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, pp. 21–33, 2016.

[14] A. Sharma, Z. Zheng, and A. Bhaskar, "Is more always better? the impact of vehicular trajectory completeness on car-following model calibration and validation," *Transportation research part B: methodological*, vol. 120, pp. 49–75, 2019.

[15] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[16] Z. Mo, R. Shi, and X. Di, "A physics-informed deep learning paradigm for car-following models," *Transportation research part C: emerging technologies*, vol. 130, p. 103240, 2021.

[17] M. Treiber, A. Kesting, and D. Helbing, "Delays, inaccuracies and anticipation in microscopic traffic models," *Physica A: Statistical Mechanics and its Applications*, vol. 360, no. 1, pp. 71–88, 2006.

[18] P. S. Bokare and A. K. Maurya, "Acceleration-deceleration behaviour of various vehicle types," *Transportation research procedia*, vol. 25, pp. 4733–4749, 2017.

[19] F. Hart, O. Okhrin, and M. Treiber, "Formulation and validation of a car-following model based on deep reinforcement learning," *arXiv preprint arXiv:2109.14268*, 2021.

[20] M. Zhu, X. Wang, and Y. Wang, "Human-like autonomous car-following model with deep reinforcement learning," *Transportation research part C: emerging technologies*, vol. 97, pp. 348–368, 2018.