Training an Intelligent Driver on Highway Using Reinforcement Learning

Sheng Li
Aeronautics and Astronautics
Engineering Department
Stanford University
Stanford, California 94305
lisheng@stanford.edu

Mengxuan Zhang
Aeronautics and Astronautics
Engineering Department
Stanford University
Stanford, California 94305
zhangmx@stanford.edu

Yuan Zhang
Aeronautics and Astronautics
Engineering Department
Stanford University
Stanford, California 94305
yuanz9@stanford.edu

***Abstract*—Traffic modeling is one of the top concerns in the field of autonomous driving. The traffic environment usually possesses high degrees of freedom and levels of uncertainty, which make the modeling process difficult. Meanwhile, the balancing among safety, comfort and efficiency greatly affects the control laws. In this paper, we propose a driver modeling process and its evaluation results of an intelligent autonomous driving policy, which is obtained through reinforcement learning techniques. Q-learning and deep Q-learning methods are applied to concise but descriptive state and action spaces, assuming a Markov Decision Process (MDP) model. So that a policy is developed within limited computational load. The driver could perform reasonable maneuvers, including acceleration, deceleration and lane-changes, under usual traffic conditions on a two-lane highway. A traffic simulator is also constructed to evaluate a given policy in terms of collision rate, average travelling speed, and lane-change times. Results show that the policy is well trained within reasonable training time. The intelligent driver that follows the policy acts interactively in the stochastic traffic environment, showing a low collision rate and a higher travelling speed than the average speed of the environment vehicles. Sample intelligent driver demonstration videos are posted on YouTube.**

Keywords*—*traffic simulator, autonomous driving, reinforcement learning, Markov decision process, decision making, Q-learning, deep Q-learning

I. Introduction

Autonomous driving has gained incredibly attractions, and is under fast development in recent years. Common approaches to the design of control systems includes localization, perceptron, decision making (path planning), and dynamics control. Several path planning and trajectory optimization methods have been proposed to the field of autonomous vehicles, including decision trees[1], and Stackelberg policies[2]. In addition, supervised learning methods are also used to modeling the interactive behaviors of human drivers and autonomous vehicles, based on real human driving data. In [3] and [4], real driving data is used in a Hidden Markov Model (HMM) based driver modeling. Oyler in [5] constructed a traffic simulator based on game theory, where each driver adopts one of the reasoning depth.

With respect to the existing approaches, this paper is distinguished by the detailed modeling of driver and vehicle interactions in traffic using MDP decision making techniques. The state space is limited to only most relevant environment observations so as to increase the learning speed. Moreover, we considered human driving habits when driving on highways, where drivers usually pursue higher travelling speed, avoid to take extra maneuver efforts and prefer not to stay in the leftmost lane.

This paper begins with the formulation process of the problem in Section II, where we introduce the traffic simulator construction, state and action spaces, and definition of the reward function. We then model the decision making process in Q-learning and deep Q-learning methods in Section III. The training evaluation and policy evaluation are demonstrated in Section IV and Section V correspondingly.

II. Problem Formulation

The problem we modeled in this paper is the maneuvers of intelligent drivers on a two-lane highway. We trained offline policies to model the behavior of a human driver. The policies include performing acceleration, deceleration as well as lane-changing. The traffic model is built on discrete state and action spaces, assuming full observability. The model is highly stochastic, where the randomness comes from the unobserved policy of environment vehicles, random initialization of traffics, and probability the state transitions. In general, the environment vehicles are only assumed to obey basic traffic laws and perform reasonable behaviors.

A. Traffic Model Construction

In order to simulate the traffic for arbitrarily long time horizon, we constructed a "circular" two-lane track model as demonstrated in Fig. 1. Note that the track is not physically circular. It can be intuitively understood as a straight track whose end connects to the start, so that all the vehicles can keep running until any form of a terminal state (reaching time limit, collision, etc.) is triggered. A closed-up of a part of the circular track is presented in Fig. 2, which is physically a straight track.

All the squares in the figure represent traffic vehicles that move in the same direction. The red square represents the ego vehicle, i.e. the vehicle of interest, whose driving policy we were trying to learn (introduced in Section III-B and III-C). The blue squares represent the environmental vehicles. They are called "environment vehicles" because they act as the environment to be observed by the ego vehicle.

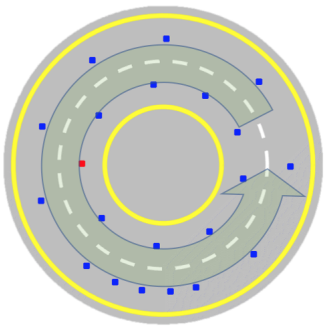

Fig. 1: Circular Track

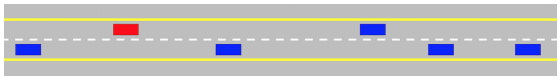

Fig. 2: Straight Track

B. Physical Model

We use discrete time equations of motion formula for the update of traffic. For any vehicle i in the traffic at time step t with observation o_i^t, we first convert the observation in continuous space to discretized state s_i^t. Then, obtaining an action from current policy at each step. The action for vehicles in this traffic model consists of acceleration in longitudinal direction (x-direction), $a_{i,x}^t$; and velocity in lateral direction (y-direction), $v_{i,y}^t$. The updating equations are shown below:

$$s_i^t \leftarrow \text{convert}(o_i^t), \tag{1}$$
$$(a_{i,x}^t, v_{i,y}^t) \leftarrow \text{policy}(s_i^t), \tag{2}$$
$$x_i^{t+1} \leftarrow x_i^t + v_{i,x}^t dt, \tag{3}$$
$$y_i^{t+1} \leftarrow y_i^t + v_{i,y}^t dt, \tag{4}$$
$$v_{i,x}^{t+1} \leftarrow v_{i,x}^t + a_{i,x}^t dt. \tag{5}$$

The position of the vehicle x_i^t and y_i^t can then be updated to next time step $t+1$ using kinematic equations.

Considering real world highway traffic regulations, speed limits are applied to the traffic model, where the upper bound of velocity to 30 m/s and the lower bound of velocity to 20 m/s. Once any vehicle hits the speed limits, its velocity is set to either maximum speed or minimum speed correspondingly. The acceleration is set to zero.

C. State Space

Before getting into the definition of state, we introduce an observation space to help modeling the surrounding environment. The observation of a vehicle consists of seven variables $\{d_{fc},\ v_{fc},,\ d_{ft},,\ v_{ft},,\ d_{rt},,\ v_{rt},,\ lane\_id\}$, as shown in Fig. 3.

1) The relative position in x-direction of the front vehicle in the same lane, d_{fc} (subscript fc indicates front and center);
2) The relative velocity in x-direction of the front center vehicle, v_{fc};
3) The relative position in x-direction of the front vehicle in the potential target lane of lane-change, d_{ft} (subscript ft indicates front and target lane);
4) The relative velocity in x-direction of the front target lane vehicle, v_{ft};
5) The relative position in x-direction of the rear vehicle in the potential target lane of lane-change, d_{rt} (subscript rt indicates rear and target lane);
6) The relative velocity of the rear target lane vehicle, v_{rt};
7) The lane ID the vehicle is currently in, $lane\_id$.

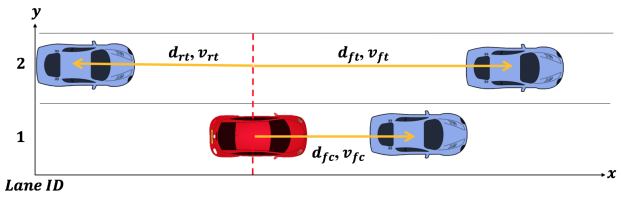

Fig. 3: Observed Variables

In real traffic flow, a driver can neither observe nor process all the above information from environment vehicles. A human can possibly observe whether a surrounding vehicle is in close or far distance, approaching or moving away. A illustration of human drivers' perception of close and far distance is shown in Fig. 4. Therefore, we assign the following discretized observation space for the drivers:

1)
$$d_{fc} = \begin{cases} 1, & \text{if } d_{fc} \leqslant \text{close distance;} \\ 2, & \text{if close distance} \leqslant d_{fc} \leqslant \text{far distance;} \\ 3, & \text{if } d_{fc} \geqslant \text{far distance,} \end{cases}$$
where the close distance $=$ 20 m, far distance $=$ 40 m;

2)
$$v_{fc} = \begin{cases} 1, & \text{if approaching, } v_{fc} < 0; \\ 2, & \text{if maintaining, } v_{fc} = 0; \\ 3, & \text{if moving away, } v_{fc} > 0; \end{cases}$$

3)
$$d_{ft} = \begin{cases} 1, & \text{if } d_{ft} \leqslant \text{close distance;} \\ 2, & \text{if close distance} \leqslant d_{ft} \leqslant \text{far distance;} \\ 3, & \text{if } d_{ft} \geqslant \text{far distance;} \end{cases}$$

4)
$$v_{ft} = \begin{cases} 1, & \text{if approaching, } v_{ft} < 0; \\ 2, & \text{if maintaining, } v_{ft} = 0; \\ 3, & \text{if moving away, } v_{ft} > 0; \end{cases}$$

5)
$$d_{rt} = \begin{cases} 1, & \text{if } d_{rt} \leqslant \text{close distance;} \\ 2, & \text{if close distance} \leqslant d_{rt} \leqslant \text{far distance;} \\ 3, & \text{if } d_{rt} \geqslant \text{far distance;} \end{cases}$$

6)
$$v_{rt} = \begin{cases} 1, & \text{if approaching, } v_{rt} < 0; \\ 2, & \text{if maintaining, } v_{rt} = 0; \\ 3, & \text{if moving away, } v_{rt} > 0; \end{cases}$$

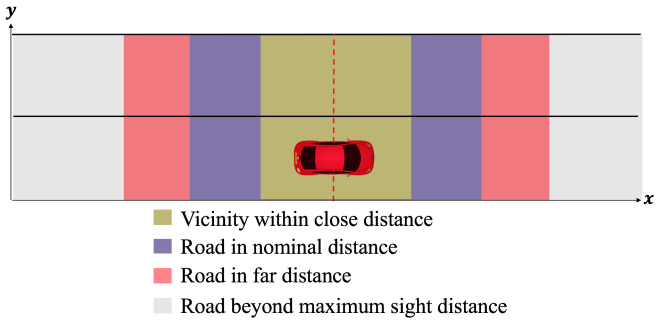

Fig. 4: Human drivers' perception of distance on road

7)

$$lane\_id = \begin{cases} 1, & \text{if in right lane;} \\ 2, & \text{if in left lane.} \end{cases}$$

If a car is beyond maximum sight distance, we assume the car is in "far distance" and "moving away". One may find there are $3^6 \times 2 = 1458$ different discrete states. Then we can map each discretized observation vector into integers ranged from 1 to 1458 using base-changing techniques. Therefore, a state of our model becomes an integer in the range from 1 to 1458.

D. Action Space

The action space includes the longitudinal acceleration, a_x, and the lateral velocity, v_y. We define six basic actions:

1) Hard acceleration, $a_x = 4$ m/s^2;
2) Mild acceleration, $a_x = 2.5$ m/s^2;
3) Maintain, $a_x = 0$ m/s^2;
4) Hard deceleration, $a_x = -4$ m/s^2;
5) Mild deceleration, $a_x = -2.5$ m/s^2;
6) Switch lane, assign v_y;

We assume a driver cannot change longitudinal velocity during lane changing maneuver. Therefore, the action is either assign a_x or assign v_y.

E. Reward Function

A reward function is defined to quantify the immediate reward at each time step. The reward function should give penalties to unsafe behaviors. The weight of penalty depends on the dangerous level. The reward function should also give rewards to behaviors that could give rise to the travel efficiency and performance. Explicitly, we defined five terms to evaluate the immediate reward r, based on current state and action,

$$r = w_1 c + w_2 v + w_3 h + w_4 a + w_5 l,$$

where 1) $c \in \{1, 0\}$ is whether the current state leads to a collision in the traffic; 2) $v = 0.2(v_x^t - v_{\text{nominal}})$, defines the difference between the current velocity of the ego vehicle v_x^t, and a expected nominal speed in the traffic v_{nominal}, with a scaling factor of 0.2, so that $v \in [-1, 1]$; 3) the discredited headway distance of $h \in \{-1, 0, 1\}$ that corresponds to close, medium and far distances to the front center vehicle; 4) The $a \in \{-5, -1, 0\}$ defines the effort of the maneuver, where we penalize action of "hard acceleration" and "hard deceleration" of -5, 0 for action "maintain", and -1 for all other actions;

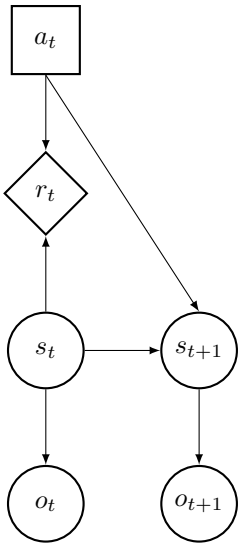

Fig. 5: Fully Observed MDP

5) $l \in \{-1, 0\}$ is whether the ego vehicle is travelling in the leftmost lane ($l = -1$ if in the leftmost lane).

The w_1, w_2, w_3, w_4, w_5, are the corresponding weights for each term. The weighting terms reflect the expectation of the trained policy, where the relative values corresponds to the trade-offs between travel efficiency, comfort and safety. In our training process, we define two sets of weights and obtain different policies. But in general,

$$w_1 >> w_2, w_3, w_4, w_5,$$

because a collision is the worst scenario in a traffic.

III. Decision Making Process

The goal is to learn the optimum policy for the ego vehicle, by assuming the control law of the environment vehicles are predefined, but unknown to the ego vehicle. We formulated the driving process of each vehicle in the traffic as a MDP with infinite horizon. In the context of MDP, at each time step t, the state of each vehicle s_t is obtained by observing the surrounding vehicles, and discretizing the observation o_t, assuming full observability. Each vehicle could extract action a_t at each time step according to its policy π and state s_t, i.e. $a_t = \pi(s_t)$. New states s_{t+1}'s are therefore reached by updating the dynamics of all vehicles in the traffic. The current state of each vehicle is only affected by its previous state, so that the process satisfies the Markovian property. The network representation of the process is shown in Fig. 5.

The system is highly stochastic from the perspective of the ego vehicle since the policies of the environment vehicles are unforeseen. Reinforcement learning is used to learn the optimal policy for the ego vehicle.

A. Policy-0

The environment vehicles are controlled by a predefined policy "Policy-0", π_0. In this traffic model, driver take safe (usually dummy) actions under different circumstances. We defined the Policy-0 as follows [5], $a^t_{i,\text{policy-0}} =$:

1) Very hard deceleration, , if $d_{fc} \leqslant$ safe distance;
2) Hard deceleration, if $d_{fc} \leqslant$ close distance and approaches to the front vehicle;
3) Mild acceleration, if close distance $\leqslant d_{fc} \leqslant$ far distance;
4) Maintain, if otherwise.

B. Q-learning

Q-learning is a standard reinforcement learning (RL) algorithm that iteratively update a Q-table. Each entry of the Q value table is a Q-function (state-value function), $Q(s, a)$.

With proper exploration method, the Q-table can converge to the optimal Q-function. An ad hoc exploration strategy, ϵ-greedy search, is used in this paper. A fixed ϵ provides a fixed exploration rate. In order to speed up the exploration in the initial stage of Q-learning algorithm, the ϵ annealing method was experimented, i.e. gradually decaying ϵ from 1 to a value close to 0 as the counting of iteration grows. The Q-learning using ϵ-greedy search algorithm is listed in Algorithm 1 [6].

Algorithm 1: Q-learning using ϵ-greedy search

Result: the optimal state action function $Q^*(s, a)$
$Q(s, a) \leftarrow 0$;
while *run time* $<$ *time limit OR episode* $<$ *episode limit* **do**
  $t \leftarrow 0$;
  $s_t \leftarrow$ traffic model random initialization;
  while $t \leqslant$ *horizon limit* **do**
    if $rand() \leqslant \epsilon$ **then**
      Randomly choose a_t from action space;
    else
      $a_t = \arg\max_a Q(s_t, a)$;
    end
    $r_t \leftarrow$ reward function(s_t);
    $s_{t+1} \leftarrow$ traffic model based on s_t and a_t;
    Bellman update on Q: $Q(s_t, a_t) \leftarrow$
     $Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$;
    $t \leftarrow t + 1$;
  end
end
return $Q^*(s, a) \leftarrow Q(s, a)$;

C. Deep Q-learning

Instead of using the Q-table, we could represent the Q-function with a neural network $N(s)$ that takes the state as input and output the Q-value for each possible action.

Since Q-values can be any real values, the neural network needs to perform a multivariate linear regression tasks. We have tried several neural network architectures that use ReLU activation function for all hidden layers, and linear activation function for output layer. The mean squared error loss function was used.

$$L = \frac{1}{2}[\underbrace{\hat{y}}_{\text{target}} - \underbrace{y}_{\text{prediction}}]^2 \tag{6}$$

The neural network architectures and training library that we have tried are listed below:

1. MATLAB Neural Network Toolbox:

- One hidden layer with 7, 30 and 100 neurons;
- Two hidden layers with 7 neurons in first layer and 8 neurons in the second layer;
- Three hidden layers with 8 neurons in first layer, 8 neurons in second layer and 8 neurons in third layer.

2. Python Keras Library:

- Two hidden layers with 24 neurons in first layer and 24 neurons in the second layer.

Neural Network can be hard to train due to the correlation of input data. Using similar and correlated inputs may lead to overfitting and convergence to local optimum. To improve the stability of neural network, target network and experience replay methods[7] were applied to break the correlation of input data.

Target Network: A separate network($\hat{N}$) is used for generating the targets $\hat{y}$. Every C episodes we replace the network $\hat{N}$ with the network N.

Experience Replay: We need to record (s, a, s', r) to re-train the neural network with the previous experiences. When training the neural network, random mini-batches from the previous experiences set are used.

The update rule of $N(s)$ is shown as follow:

1) Use $N(s)$ to get predicted Q-values for all actions.
2) Use $\hat{N}(s')$ to get $\max_{a'} \hat{Q}(s', a')$.
3) Set target for action to $r + \gamma \max_{a'} \hat{Q}(s', a')$.
4) Update the weights of neural network by using the loss function below.

$$L = \frac{1}{2}[\underbrace{r + \gamma \max_{a'} \hat{Q}(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2 \tag{7}$$

For deep Q-learning, the ϵ-greedy method was also applied. For the first E episodes, the ϵ was set to one to randomly explore the entire state space. After the first E episodes, the ϵ annealing method was applied.

IV. Training Process Evaluation

A. Training Process

a) Q-learning: To evaluate Q-learning training process we defined a score for each episode. The score is a scaled measurement of total distance that the ego vehicle travelled in each episode.

$$\text{score} = (\text{total travelled distance} - 4000)/1000. \tag{8}$$

We plotted score of each training episode. Fig. 6 shows an example of the score evolution history over 10,000 training episodes with a constant $\epsilon = 0.1$. Fig. 7 shows an example of the score evolution history over 10,000 training episodes with a exponentially decaying ϵ: $\epsilon(\text{episode}) = \exp(-5/10,000 * \text{episode})$. Red dots in figures indicate episodes in which vehicle collides. Blue dots indicate the safe episodes with no collision.

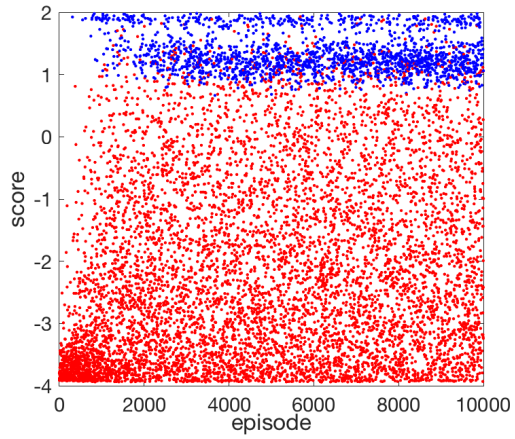

Fig. 6: Score history of constant ϵ (0.1) Q-learning

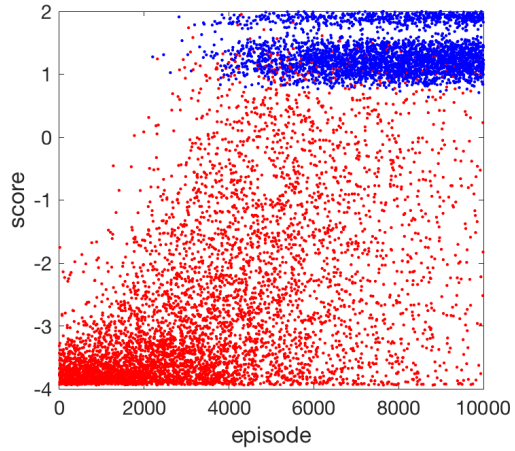

Fig. 7: Score history of ϵ-decay Q-learning

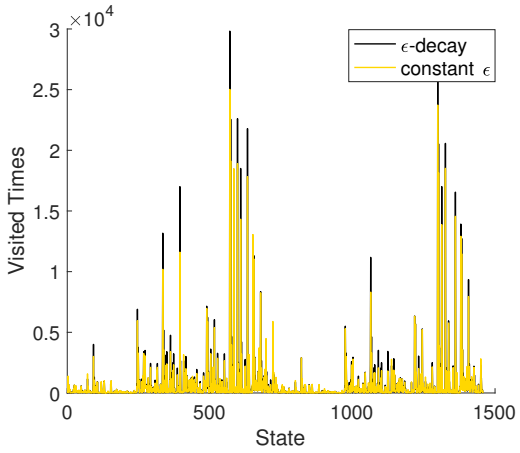

Fig. 8: State visited times count

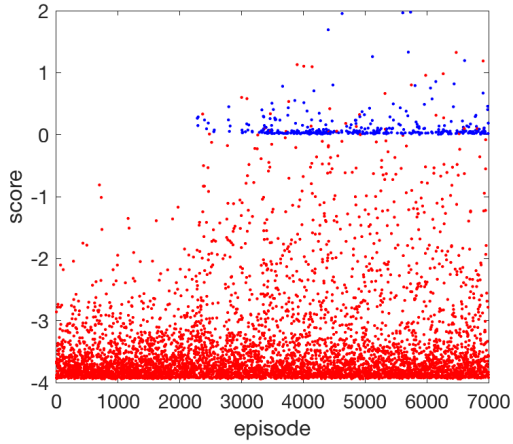

Fig. 9: Score history of deep Q-learning

The constant ϵ method provides a fixed exploration and exploitation ratio. We may find in Fig. 6 that the distribution of score converges fairly quickly to a fixed pattern. While the ϵ-decay method provides a dynamic exploration and exploitation ratio. At the beginning stage of training, exploration is dominant, i.e. $\epsilon \approx 1$. As the number of episode increases, ϵ decays to a smaller value, i.e. exploitation dominates. One may find in Fig. 7 that in the beginning stage, collision dominates since action is nearly random. As the number of episode increases, the score distribution pattern shows the clustering of the non-collision episodes, as well as the decreasing trend of collisions. This indicates the Q-table has converged.

Fig. 8 provides an overplot of state visited times counting of the two different exploration strategies. One may find that the different training methods demonstrate identical state visiting history.

The training time of both constant ϵ and ϵ-decay methods is five hours.

b) Deep Q-learning: All neural network architectures, hyperparameters, training library and exploration strategies that we have tried return the similar results if fixing a reward function. Fig. 9 shows the score history of deep Q-leaning with following settings: two hidden layers with 7 neurons in first layer and 8 neurons in the second layer. For the first 2000 episodes, the ϵ was set to 1. For 2000-3000 episodes, ϵ decreased from 1 to 0.1. After 3000 episodes, $\epsilon = 0.1$. The result shows that for a given reward function, the deep Q-learning performs similar learning trend as Q learning; however, it converges to and sticks at a local optimum. This can be explained by the failure of target network and experience replay methods in breaking the correlation of input data due to the fact that our state space is small.

B. Policy Extraction

The raw policy is extracted from the maximum Q value from each state using Eqn. (9),

$$\pi(s) = \arg\max_a Q(s, a). \tag{9}$$

The number of visited times is recorded for each state during training. A seldom visited state may be poorly trained (not reach local optimum value or not converge) through the learning process, where an unreasonable behavior may take place when the ego car reaches such states. Although these seldom visited states during training usually rarely meet in the simulation process as well, one still wish to refine those states

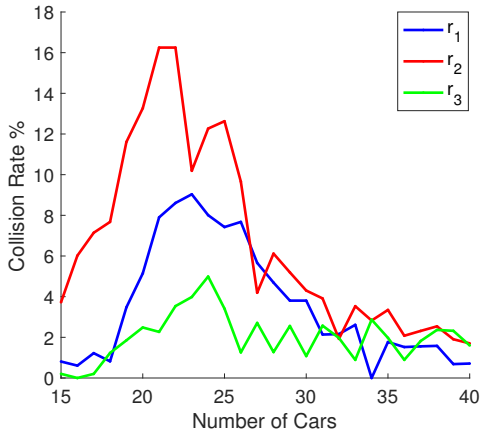

Fig. 10: Collision rate vs. number of cars

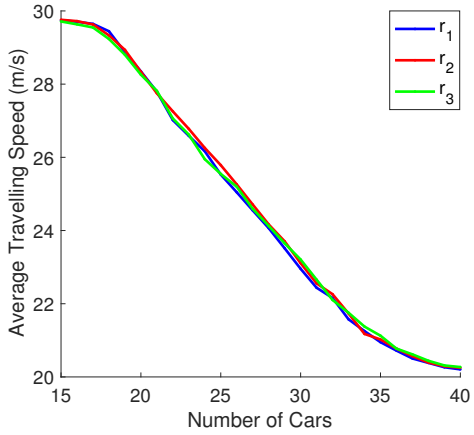

Fig. 11: Performance vs. number of cars

to a proved safe action so as to guarantee the integrity of the policy.

In our project, we replaced the actions of the rarely visited states (the visited time is less than 30 during the entire training process) with the same policy as "Policy-0".

V. Policy Evaluation

We evaluated the policy by applying the trained policy to one of the cars in the traffic, while other cars use "Policy-0" as defined in Section III-A. We simulated the traffic with random initialization, and evaluated the average performances among multiple simulations. We simulated three different policies where total number of vehicles in the traffic varies from 15 to 40. Policies are trained from the following reward functions:

$$r_1 = 1000c + 5v + h + a + l, \tag{10}$$
$$r_2 = 1000c + 10v + h + a + 0.1l, \tag{11}$$
$$r_3 = 1000c + 5v + h + a + 0.1l. \tag{12}$$

Since the uncertainty of the traffic model is high, each result is averaged over 500 samples. The results are indicated from Fig. 10 to Fig. 12.

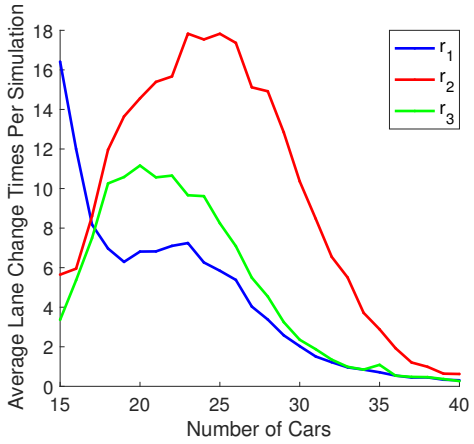

Fig. 12: Average lane change times vs. number of cars

The performances of the driving behavior are shown to be affected by the definition of the immediate reward. The r_2 gives more weights on velocity profile, which results in a slightly higher average travelling speed as shown in Fig. 11, but a greater collision rate in turn. The r_1 defines a larger penalty on making lane changes, so that the average lane change times per simulation is much lower than the other policies shown in Fig. 12. The adjustment of reward function is determined upon user preferences on the level of aggressiveness.

VI. Conclusion

In this paper, an autonomous driving policy model is trained, utilizing Q-learning and deep Q-learning approaches. The intelligent driver adopting the trained policy could perform reasonable behaviors in the interactive traffic with randomnesses. The trained policy is used to construct a closed-loop traffic simulator where we are able to evaluated the driving performance.

Future work may include the refinement of the deep Q-learning method, in terms of defining a finer discretization of the state and avoiding the overfitting problem.

VII. Links

GitHub repository:
https://github.com/Michelle-NYX/traffic-simulator-Q.git

Intelligent driver demonstration videos on YouTube:
https://youtu.be/OFRZzvPH30g

References

[1] G. S. J. K. A. Carvalho, S. Lefevre and F. Borrelli, "Automated driving: The role of forecasts and uncertainty-a control perspective," *European Journal of Control*, vol. 24, pp. 14–32, 2015.

[2] J. H. Yoo and R. Langari, "Stackelberg game based model of highway driving." Fort Lauderdale, Florida: ASME Dynamic Systems and Control Conference joint with JSME Motion and Vibration Conference, October 2012.

[3] D. V. E. T. R. B. S. Lefevre, Y. Gao and F. Borrelli, "Lane keeping assistance with learning-based driver model and model predictive control." the 12th international symposium on advanced vehicle control, 2014.

[4] A. C. S. Lefevre and F. Borrelli, "Autonomous car following: A learning-based approach." IEEE Intelligent Vehicles Symposium, 2015.

[5] D. W. Oyler, Y. Yildiz, A. R. Girard, N. I. Li, and I. V. Kolmanovsky, "A game theoretical model of traffic with multiple interacting drivers for use in autonomous vehicle development." Boston Marriott Copley Place, Boston, MA: American Control Conference (ACC), July 6-8 2016.

[6] M. J. Kochenderfer, *Desicion Making Under Uncertainty*. The MIT Press, 2015.

[7] 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.

CONTRIBUTIONS

Sheng Li: Implemented traffic simulator. Implemented Q-learning training process with epsilon greedy methods, as well as trained multiple policies. Performed policy evaluations. Made the simulation video.

Mengxuan Zhang: Problem formulation with introduction/ definition of observations, state and action spaces, reward functions. Helped with traffic dynamics modeling. Performed policy evaluations. Generate traffic simulator visualization plots.

Yuan Zhang: Wrote MATLAB version of deep Q-learning algorithm and implemented it. Modified traffic simulator for python and implemented python version of deep Q-learning algorithm.