Quadratic Q-network for Learning Continuous Control for Autonomous Vehicles

Pin Wang[1]*, **Hanhan Li**[2], **Ching-Yao Chan**[1]
[1] University of California, Berkeley
[2] Google AI
{pin_wang, cychan}@berkeley.edu
uniqueness@google.com

Abstract

Reinforcement Learning algorithms have recently been proposed to learn time-sequential control policies in the field of autonomous driving. Direct applications of Reinforcement Learning algorithms with discrete action space will yield unsatisfactory results at the operational level of driving where continuous control actions are actually required. In addition, the design of neural networks often fails to incorporate the domain knowledge of the targeting problem such as the classical control theories in our case. In this paper, we propose a hybrid model by combining Q-learning and classic PID (Proportion Integration Differentiation) controller for handling continuous vehicle control problems under dynamic driving environment. Particularly, instead of using a big neural network as Q-function approximation, we design a Quadratic Q-function over actions with multiple simple neural networks for finding optimal values within a continuous space. We also build an action network based on the domain knowledge of the control mechanism of a PID controller to guide the agent to explore optimal actions more efficiently. We test our proposed approach in simulation under two common but challenging driving situations, the lane change scenario and ramp merge scenario. Results show that the autonomous vehicle agent can successfully learn a smooth and efficient driving behavior in both situations.

1 Introduction

Reinforcement Learning (RL) has been applied in robotics for decades and has gained popularity due to the development in deep learning. In recent studies, it has been applied for learning 3D locomotion tasks (e.g. bipedal locomotion and quadrupedal locomotion [12]) as well as robot arm manipulation tasks (e.g. stacking blocks [6]). Google DeepMind also showed the power of RL for learning to play Atari 2600 games [9] and Go games [13]. In these applications, either the operation environment is simple, e.g. no interaction with other agents, or the action space is limited to discrete selection, e.g. left, right, forward, and backward.

In regard to the driving tasks for autonomous vehicles, the situation is totally different because vehicles are required to operate smoothly and efficiently in a dynamic and complicated driving environment. Tough challenges frequently arise in driving domains. For example, the vehicle agent needs to coordinate with surrounding vehicles so as not to disturb the traffic flow significantly when it executes a maneuver, e.g. merging into a joint traffic flow. More importantly, the control action should be continuous to guarantee smooth travelling.

*Corresponding author

Machine Learning for Autonomous Driving Workshop at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

There have been some efforts in applying RL to autonomous driving [18][10][11], however, in some of the applications the state space or action space are arbitrarily discretized (e.g. vehicle acceleration is split into some fixed values), to fit into the RL algorithms (e.g. Q-learning) without considering the specific features of the driving problem. This simplified discretization results in the loss of a complete representation of the continuous space. Policy gradient-based methods are alternatives for continuous action problems, but they often complicates the training process by involving a policy network and sometimes suffer from vanishing or exploding gradient problems.

In this study, we resort to model-free Q-learning and design the Q-function in a Quadratic form based on the idea of Normalized Advantage Function [5]. With this form, the optimal solution can be obtained in a closed form. Additionally, we incorporate the domain knowledge of the control mechanism in the design of an action network to help the agent with action exploration. We test the algorithm with two challenging driving cases, the lane change situation and ramp merge situation.

The reminder of the paper is organized as follows. Related work is described in Section 2. Methodology is given in Section 3, followed by application case in Section 4 and experiments in Section 5. Conclusions and discussions are given in the last section.

2 Related Work

In autonomous driving field, a vast majority of studies on operational level of driving are based on traditional methods. For example, in [8], a virtual trajectory reference was created by a polynomial function for each moving vehicle, and a bicycle model was used to estimate vehicle positions based on the pre-calculated trajectories. In [2], a number of way points obtained from Differential Global Positioning System and Real-time Kinematic devices were used generate a path for the autonomous vehicle to follow. Such approaches can work well in predefined situations or within the model limits, however, they have limited performance in unforeseeable driving conditions.

In recent years, we have seen a lot of applications of RL on the automated driving domain [18] [10][16][11]. For example, Yu et al. [18] explored the application of Deep Q-Learning methods to the control of a simulated car in JavaScript Racer. They discretized the action space into nine actions and found that the vehicle agent can learn turning operations when there were no cars on the raceway but cannot perform well in obstacle avoidance. Ngai et al. [10] put multiple goals in the RL framework (i.e. destination seeking and collision avoidance) to address the overtaking problem. They also converted continuous sensor values into discrete state-action pairs. In all of these applications, the action space was treated as discrete and few interactions with the surrounding environment were considered. Wang et al. [16] proposed a RL framework for learning on-ramp merge behavior, where a Long Short Term Memory (LSTM) was used to learn internal states and a Deep Q-Network was used for deciding the optimal control policy.

Sallab et al. [11] moved further to explore the impacts of a discrete and a continuous action space on the lane keeping case. They conducted experiments in a simulated environment, and concluded that the vehicle agent traveled more smoothly under continuous action design than discrete action design.

Q-learning is simple but effective, and is basically applicable to discrete action space. If a Q-function approximator can be designed to encode continuous action values to corresponding Q-values, it becomes an optimization problems in a continuous action space [17]. And it also avoids involving a complicated policy network as in most policy gradient based methods [14][3]. Based on these thoughts, we design a quadratic Q-network, similar to the idea of Normalized Advantage Functions (NAF) [5] in which the advantage term is parameterized as a quadratic function of nonlinear features of the state. We apply the method to the practical application case in autonomous driving, and combine it with domain knowledge of vehicle control mechanism to assist its action exploration.

3 Methodology

3.1 Quadratic Q-network

In our RL formulation, the state space S and the action space A are taken as continuous. The goal of the reinforcement learning is to find an optimal policy $\pi^* : S \rightarrow A$, so that the total return G accumulated over the course of the driving task is maximized. For a given policy π with parameters θ, a Q-value function is used to estimate the total reward from taking action a_t given state s_t at time t

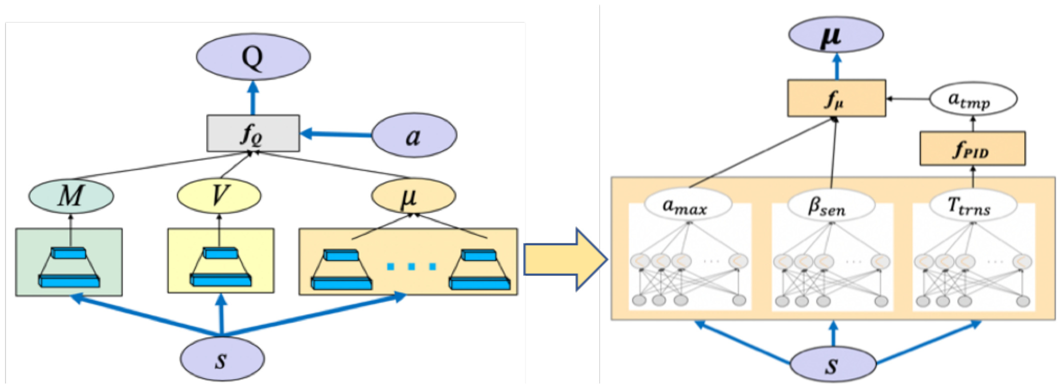

Figure 1: Architecture of Quadratic Q-network (left) and action network μ (right).

[14]. A value function is used to estimate the total reward from state s_t. The advantage function is to calculate how much better a_t is in state s_t as

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t). \tag{1}$$

In the case where the action space is discrete, we can obtain the optimal action a_t^* with the greedy policy directly by iterating over the action space, as

$$a_t^* = \text{argmax}_{a_t} Q^\pi(s_t, a_t). \tag{2}$$

However, when the action space is continuous, it is not easily ready to apply the basic Q-learning formula to find the optimal action. According to the nature of the quadratic equation, if the Q-function $Q^\pi(s_t, a_t)$ has a quadratic form, the optimal action can be obtained analytically and easily. With this idea, we design the Q-function in a quadratic format as

$$Q^\pi(s_t, a_t) = A^\pi(s_t, a_t) + V^\pi(s_t) = (\mu(s_t) - a_t)^T M(s_t)(\mu(s_t) - a_t) + V(s_t), \tag{3}$$

where $\mu(s_t)$ is a vector with the same dimension of the action, $M(s_t)$ is a negatively semi-definite matrix, and $V(s_t)$ is considered as a the value function with a scalar value as output. With this special form of Q-function, the optimal action a_t^* can still be obtained in a greedy way as in equation (2) but given by $\mu(s_t)$.

Figure 1 (left) depicts the architecture of the quadratic Q-network. $M(s_t)$ and $V(s_t)$ are built separately with a single multilayer perceptron (MLP) with two hidden layers. In contrast, $\mu(s_t)$ consists of three MLPs that are combined in a special way. We call $\mu(s_t)$ the action network and give its details in the next subsection. $M(s_t)$, $V(s_t)$, and $\mu(s_t)$ are combined in the function f_Q which is the equation (3).

Note that because any smooth Q-function should be Taylor-expanded in this quadratic format near the greedy action, there is not much loss in generality with this assumption if we stay close to the greedy policy that is being updated in the Q-learning process.

3.2 Action network

From equation (3) we can observe that $\mu(s_t)$ plays a critical role in learning the optimal action. If it is purely designed as a neural network with thousands of neurons, it may suffer a hard time learning actions meaningful to a driving policy.

Based on this insight, we design the form of μ similar to a PID controller where some tuning parameters are replaced with neural networks. In other words, we do not manually tune the coefficients for the proportional, integral, and derivative terms but use neural networks to automatically find the appropriate values based on the defined reward function in RL. This way it makes the controller adaptable to different driving situations, and moreover the output action is based on a long-term goal of the task rather than an action just calculated for a target at current step as in a PID controller.

The right graph in Figure 1 shows the design of action network $\mu(s_t)$ where three variables a_{max}, β_{sen} and T_{trs} are designed with neural networks, and Equation (4) and (5) show how these variables are combined. To be specific, from equation (4), we obtain a temporary action based on PID properties, where T_{trs} is the output from a neural network and interpreted as a transition time to mitigate errors between current and target states. The temporary value then goes through a hyperbolic tangent activation function in equation (5), where another two parameters, a_{max} and β_{sen}, are learned from neural networks. a_{max} represents a tunable maximum acceleration and β_{sen} indicates a sensitivity factor enforced on the temporary control action. Figure 1 (right) depicts the architecture of μ.

$$a_{tmp} = \frac{f(\Delta s)}{T_{trs}^2} + \frac{f'(\Delta s)}{T_{trs}} \tag{4}$$

$$a = a_{max} * \tanh(\beta_{sen} * a_{tmp}) \tag{5}$$

where Δs is the state difference between a desired state and the current state. The desired state can be defined conveniently, for example, it can be the target lane ID for the lateral control or the preferred car-following distance for the longitudinal control. The state difference can include values such as relative distance Δd, relative speed Δv, and/or relative yaw angle $\Delta\phi$.

3.3 Learning procedure

There are two iterative loops in learning the policy. One is a simulation loop where it provides the environment that the vehicle agent interacts with, and the other one is a update loop in which the neural network weights are updated.

In the simulation loop, we use $\mu(s_t)$ to obtain the greedy action for a given state s_t at step t. The greedy action is then perturbed with a Gaussian noise n_t to increase its exploration and executed in the simulation. After the execution, we get a new state s_{t+1} as well as a reward r_t from the environment, and store the transition tuple (s_t, a_t, s_{t+1},r_t) in a replay memory D.

In the update loop, samples of tuples are drawn randomly from D. To overcome the inherent instability issues in Q-learning, we use experience replay technique and a target Q-network as proposed in [1]. Weights in Q-network (θ) are updated by gradient descent at every time step, while weights in target Q-network (θ^-) are periodically overwritten by θ. Algorithm 1 gives the learning process.

Algorithm 1 Quadratic Q-learning

```
 1: Initialize Q-network weights $\theta$ in $M$, $V$, $\mu$
 2: Initialize target Q-network weights $\theta^- \leftarrow \theta$
 3: for episode $i = 1$ to $N$ do
 4:     Initialize action exploration noise $n_t \sim N$
 5:     Obtain initial state $s_0$
 6:     for $t = 1$ to $T$ do
 7:         Select an action $a_t = \mu_\theta(s_t) + n_t$
 8:         Execute the action $a_t$ and store the experience tuple ($s_t$, $a_t$, $s_{t+1}$,$r_t$) in $D$
 9:         if pretrain then
10:             Update Q-function weights in $M$ and $V$ while keep $\mu$ frozen with samples from $D$
11:         Update all the weights $\theta$ in Q-function (3) with batches drawn from $D$ for $m$ iterations
12:         Update all the weights $\theta^-$ in target Q-network periodically with $\theta$ from Q-network
13:     end for
14: end for
```

It is also worth mentioning that the overall training process includes two steps, pre-training step and training step. In pre-training, we only train the neural networks of M and V, and froze the parameters in μ. During training, we jointly update parameters in all neural network. This trick helps the agent learn faster.

3.4 Reward function

In our study, the immediate reward is designed as a linear combination of multiple feature functions with respect to driving safety, comfort and efficiency. Each state-action pair is evaluated with a negative value, thus teaching the agent to avoid resulting in situations with large penalties.

To be specific, safety is evaluated by relative distances from vehicles that matter most to the ego vehicle. It includes relative distance to vehicles on the longitudinal direction and the distance to the center-line of the target lane on the lateral direction.

$$R_s = -w_{s1}(\sum_{l=1}^{L} f_{s1}(\Delta d_{lg})_l - w_{s2} f_{s2}(\Delta d_{lt}) \tag{6}$$

where R_s is the safety reward term, f_{s1} and f_{s2} are the feature functions which can be power functions based on how much we want to rate this feature, w_{s1} and w_{s2} are the weights, Δd_{lg} and Δd_{lt} are the relative distance on longitudinal and lateral directions, and L is the number of adjacent vehicles.

Comfort is evaluated by the control variables, a_{lg} and a_{lt} (i.e. speed acceleration and yaw acceleration), and their derivatives, b_{lg} and b_{lt}.

$$R_c = -w_{c1}(f_{c1}(a_{lg}, a_{lt}) - w_{c2} f_{c2}(b_{lg}, b_{lt}) \tag{7}$$

where R_c is the comfort reward term, f_{c1} and f_{c2} are feature functions, and w_{c1} and w_{c2} are the weights.

Efficiency is evaluated by the maneuvering time, i.e. how long it takes to finish the task. For example, in a merging case, it is the time consumed from the initiation to the completion of the behavior. The efficiency at a single time step is calculated by the time step interval (Δt).

$$R_t = -w_t * f_t(\Delta t) \tag{8}$$

where R_t is the efficiency reward term, f_t is the feature function that can also be a power function of Δt, and w_t is the function weight.

Function weights are hyperparameters that are manually tuned through multiple training episodes. Their values and the expressions of the feature functions is given in the next section.

4 Applications on lateral and longitudinal control

We apply the proposed algorithm to two use cases, a lane-change situation and a ramp-merge situation. In the lane-change scenario, the lateral control is learned while the longitudinal control is an adapted Intelligent Driver Model (IDM) [15]. In the ramp-merge scenario, the longitudinal control is learned while the lateral control is to follow the center-line of the current lane. We defer the work of simultaneously learning control variables on the two directions to our future work.

There assumes to be a decision-making module and a gap selection module in the higher level that issue commands on when to make lane change or ramp merge. Our work focuses on learning the control variables under the received the commands.

4.1 Lateral control under lane-change case

The lane change behavior is affected by the ego vehicle's kinematics (e.g. vehicle speed, position, yaw angle, yaw rate etc.) as well as the surrounding vehicles' in the targeting gap. Road curvature also affects the success of a lane change, for example, a curved road segment introduces additional centrifugal force that should be considered in the lane change process. Therefore, we define the state space to include both vehicle dynamics and road curvature information.

As mentioned earlier, we resort to a well-developed car-following model, Intelligent Driver Model (IDM) [15], with some adaptation for the longitudinal control. IDM describes dynamics of the positions and velocities of single vehicles. Due to space limitation, we only briefly introduce the modified IDM that is adapted to alleviate overly conservative driving behaviors. The longitudinal acceleration a_{lg} is calculated by equation (9).

$$a_{lg} = a_m(1 - \max((\frac{v_\alpha}{v_0})^\delta, (\frac{s_0 + v_\alpha T}{s_\alpha} + \frac{v_\alpha \Delta v_\alpha}{2\sqrt{a_m b} s_\alpha})^2)) \tag{9}$$

where Δv_α is the velocity difference between the ego vehicle α and its preceding vehicle $\alpha - 1$, v_0 is the desired velocity of the ego vehicle in free traffic, s_0 is the minimum spacing to the leader, s_α is the current spacing, T is the minimum headway to the leader, a_m is the maximum acceleration, b is

Table 1: Reward function design for lane change and ramp merge

Functions	Lane change	Ramp merge	**Weights**	Lane change	Ramp merge
f_{s1}	None	$\sum_{l=1}^{2}(\Delta d_{lng})_l$	w_{s1}	None	0.01
f_{s2}	$\|\Delta d_{lt}\|$	None	w_{s2}	0.05	None
f_{c1}	$\|a_{lt}\|$	$\|a_{lg}\|$	w_{c1}	0.5	0.5
f_{c2}	$\|\omega\|$	None	w_{c2}	2.0	None
f_t	$\|\Delta t\|$	$\|\Delta t\|$	w_t	0.05	0.05

the comfortable braking deceleration, and δ is the exponential parameter. In our test case, we set s_0 to $1m$, T to $1s$, a_m to $2.0m/s^2$, b to $1.5m/s^2$, and δ to 4.

The action space for lateral control is treated as continuous to allow any reasonable real values being taken in the lane change process. Specifically, we define the lateral control action to be the yaw acceleration, $a_{lt} = \ddot{\theta}$ with the consideration that a finite yaw acceleration ensures the smoothness in steering, where θ is the yaw angle.

The reward function, composed of the three parts of safety, comfort and efficiency, is given in Table 1. In the safety part, only reward from the lateral direction is considered in which Δd_{lt} is the lateral deviation from current position to the center-line of the target lane. Safety in the longitudinal direction is taken care of by the gap selection and IDM model. The comfort part is evaluated by the lateral action a_{lt} and the yaw rate ω. The efficiency is evaluated by time-step intervals.

4.2 Longitudinal control under ramp-merge case

In the ramp-merging case, when the gap selection module finds a proper gap on the merging lane, the vehicle agent will try to merge into it by adjusting its longitudinal acceleration while keeping itself in the middle of the lane. The state space in such a situation includes the speed, position, heading angle of the ego vehicle, its leading vehicle and vehicles from the target gap. The action space is the longitudinal acceleration with continuous values in a limited range of $[-4.5m/s^2, 2.5m/s^2]$.

The reward function is given in Table 1. The safety term is decided by relative distances to the leading and lagging vehicles of the target gap on the merging lane. No lateral deviation is considered as discussed above. The longitudinal acceleration decides the comfort reward term. Efficiency is evaluated by time-step intervals, the same as in the lane-change case.

5 Simulation and results

5.1 Simulation environment

The lane-change behavior is simulated on a highway segment of 350m long and three-lane wide on each direction. The ramp-merge behavior is simulated in the highway-ramp merging zone where the ramp is merged into the rightmost lane of the highway. An illustration of the simulation scene is show in Figure 2.

The simulated traffic is customized to generate diverse driving conditions. The initial speed, departure time interval, and speed limit of each individual vehicle is set to random values as long as they are within reasonable ranges, e.g. [30 km/h, 50 km/h], [5s, 10s], and [80 km/h, 120 km/h], respectively. In the simulation, vehicles can interact with each other. One example is that lagging vehicles in a lane-change case can yield or overtake the ego vehicle, creating diverse and realistic driving situations for training the RL agent.

The RL vehicle agent in the lane-change case is randomly generated in the middle lane thus that it can make either left or right lane change based on the command received after traveling for 150m. In the ramp-merge case, the RL agent is generated on the ramp, about 150m away from the merging intersection. Vehicles on the highway travel on its current lane and have the IDM car-following behavior. Additionally, a small portion of aggressive driving behaviors are simulated by setting a relatively high acceleration range and small car-following distances, and conversely for defensive driving behaviors.

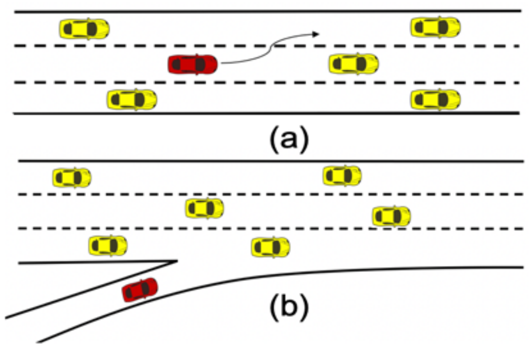

Figure 2: (a) Lane-change scenario (b) Highway ramp-merge scenario.

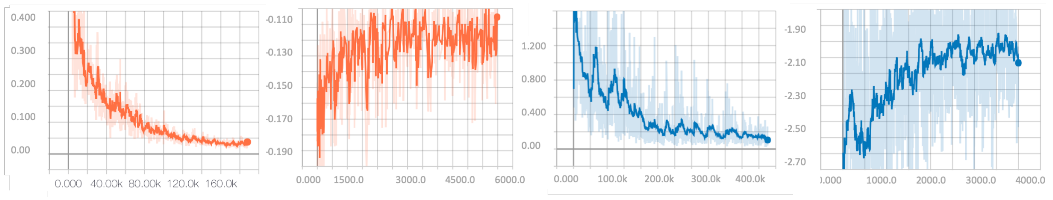

Figure 3: Training losses and accumulated rewards in lane-change case (in orange) and ramp-merge case (in blue) v.s. training steps, saved at different intervals. It shows that the loss decreases to a convergence and the accumulated reward increases to a convergence along with training steps.

5.2 Training results

The hyperparameters for training in the two application cases are similar expect for the learning rate, 0.0005 for lane change and 0.001 for ramp merge, and training episodes, 6000 for lane change and 4000 for ramp merge. Other hyperparameters are set as follows: replay memory=2000, batch size=64, discount factor=0.95, target-Q weights update frequency=1000, optimizer=Adam. Twelve intermediate checkpoints are saved in each application case for testing the learned models.

Training loss and accumulated rewards are plotted in Figure 3 for both the lane-change case and ramp-merge case. From Figure 3, we can observe that in both cases the training loss curve shows an obvious convergence and that the total rewards also demonstrate a consistently increasing trend, which satisfactorily indicates that the RL vehicle agent has learned the lane-change behavior and ramp-merge behavior.

Since each point in the total reward curves represents only one random driving case under that corresponding training step, it might not be enough to prove the learned driving behavior. Therefore, we conduct testing on the saved checkpoints to get an averaged driving performance. We run 100 episodes at each checkpoint in both the lane-change situation and ramp-merge situation, and then average their total rewards. The results are plot in Figure 4.

The testing curves in Figure 4 show consistent upward trend as the total reward curves in Figure 3, indicating that the RL agent has indeed progressively learned the driving behavior of lane change and ramp merge, and can take responsible actions with respect to safety, comfort and efficiency as defined in the reward function.

We also plot some driving dynamics to further compare the driving performance at the initial stage and the final stage, i.e., at the early saved checkpoint and last saved checkpoint. The right graph in Figure 5 demonstrates the lane change trajectories (blue for left lane change and red for right lane change) at the initial stage (upper right) and the final stage (lower right), respectively. It shows clearly that the trajectories at the final stage, in comparison to initial stage, are quite smooth and stable. The left graph in Figure 5 shows the acceleration curves in the ramp merge case as we learn the longitudinal control in this situation. We can see that the acceleration in a merging case gets smoother (lower left) in the final stage than that in the initial stage (upper left).

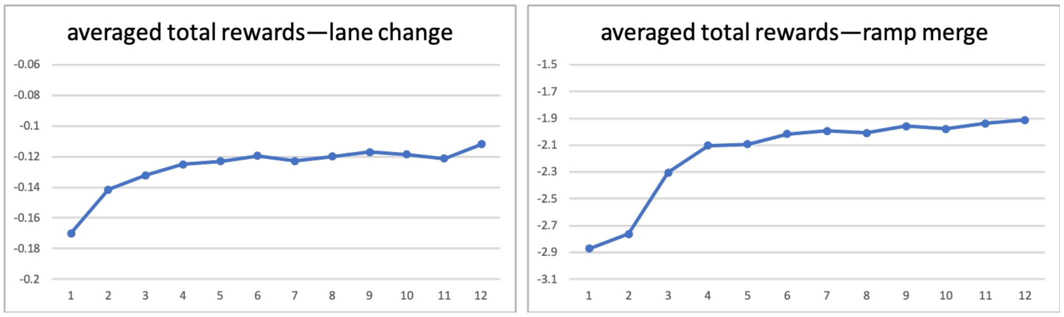

Figure 4: Testing performance. Averaged rewards over 100 test runs at each saved checkpoints v.s. saved checkpoints. Left: lane change case. Right: ramp merge case. It demonstrates that the averaged total rewards increase as training proceeds, consistent with the training results.

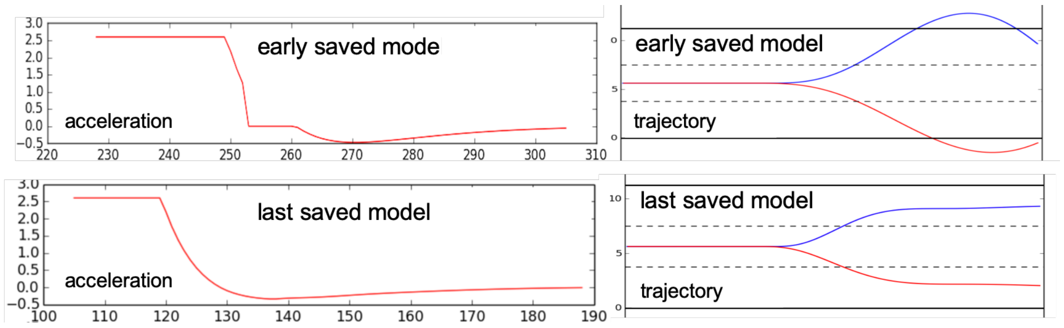

Figure 5: Acceleration comparison in ramp merge (left) and trajectory comparison in lane change (left). The graphs show that the acceleration and trajectory become more smooth and stable when training finishes.

6 Conclusion and discussion

In this work, we designed a Quadratic Q-network for handling continuous control problems in autonomous driving. With the quadratic format, the optimal action can be obtained easily and analytically. We also leverage domain knowledge of vehicle control mechanism for designing an action network, to provide the vehicle agent guidance in the action exploration.

The proposed method is applied to two challenging driving cases, the lane-change case and ramp-merge case. Training results show convergence in both the training losses and total rewards, indicating that the RL vehicle agent has learned to drive with higher rewards as defined in the reward function. Testing results show consistent convergence trend as that in the training, proving that the agent has indeed learned the behavior of lane changing and ramp merging. Comparison of the driving trajectories (in lane change situation) and vehicle accelerations (in ramp merge situation) at respectively the initial stage and final stage also reveals that the agent can drive safely, smoothly and efficiently.

This study demonstrates the potentials of applying the quadratic Q-learning framework to continuous control problems in autonomous driving. Our further step is to learn the longitudinal and lateral controls simultaneously based on different designs of reward functions. Also, we will try other directions for learning the policy, such as methods based on adversarial learning. Generative Adversarial Imitation Learning [7] and Adversarial Inverse Reinforcement Learning [4] show promising features in learning robotic control and it can recover both a policy and a reward function from demonstrations. Applying it to the dynamically changing environment in autonomous driving will be a challenging but interesting work.

References

[1] S. Adam, L. Busoniu, and R. Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):201–212, 2011.

[2] Y.-G. Choi, K.-I. Lim, and J.-H. Kim. Lane change and path planning of autonomous vehicles using gis. In *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 163–166. IEEE, 2015.

[3] T. Degris, P. M. Pilarski, and R. S. Sutton. Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference (ACC)*, pages 2177–2182. IEEE, 2012.

[4] J. Fu, K. Luo, and S. Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[5] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[6] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6244–6251. IEEE, 2018.

[7] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[8] M. Ho, P. Chan, and A. Rad. Lane change algorithm for autonomous vehicles via virtual curvature method. *Journal of Advanced Transportation*, 43(1):47–70, 2009.

[9] 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.

[10] D. C. Ngai and N. H. Yung. Automated vehicle overtaking based on a multiple-goal reinforcement learning framework. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 818–823. IEEE, 2007.

[11] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. End-to-end deep reinforcement learning for lane keeping assist. *arXiv preprint arXiv:1612.04340*, 2016.

[12] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[14] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[15] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

[16] P. Wang and C.-Y. Chan. Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2017.

[17] P. Wang, C.-Y. Chan, and A. de La Fortelle. A reinforcement learning based approach for automated lane change maneuvers. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1379–1384. IEEE, 2018.

[18] A. Yu, R. Palefsky-Smith, and R. Bedi. Deep reinforcement learning for simulated autonomous vehicle control. *Course Project Reports: Winter*, pages 1–7, 2016.