Automotive Innovation (2021) 4:328–337
https://doi.org/10.1007/s42154-021-00151-3

End-to-End Autonomous Driving Through Dueling Double Deep Q-Network

Baiyu Peng[1] · Qi Sun[1] · Shengbo Eben Li[1] · Dongsuk Kum[2] · Yuming Yin[1] · Junqing Wei[3] · Tianyu Gu[3]

Received: 24 June 2020 / Accepted: 26 April 2021 / Published online: 27 June 2021

Abstract

Recent years have seen the rapid development of autonomous driving systems, which are typically designed in a hierarchical architecture or an end-to-end architecture. The hierarchical architecture is always complicated and hard to design, while the end-to-end architecture is more promising due to its simple structure. This paper puts forward an end-to-end autonomous driving method through a deep reinforcement learning algorithm Dueling Double Deep Q-Network, making it possible for the vehicle to learn end-to-end driving by itself. This paper firstly proposes an architecture for the end-to-end lane-keeping task. Unlike the traditional image-only state space, the presented state space is composed of both camera images and vehicle motion information. Then corresponding dueling neural network structure is introduced, which reduces the variance and improves sampling efficiency. Thirdly, the proposed method is applied to The Open Racing Car Simulator (TORCS) to demonstrate its great performance, where it surpasses human drivers. Finally, the saliency map of the neural network is visualized, which indicates the trained network drives by observing the lane lines. A video for the presented work is available online, https://youtu.be/76ciJmIHMD8 or https://v.youku.com/v_show/id_XNDM4ODc0MTM4NA==.html.

Keywords End-to-end autonomous driving · Reinforcement learning · Deep Q-network · Neural network

Abbreviations

CNN	Convolutional neural network
RL	Reinforcement learning
MDP	Markov decision process
DQN	Deep Q-Network
DDQN	Double Deep Q-Network
DDDQN	Dueling Double Deep Q-network
TORCS	The Open Racing Car Simulator

1 Introduction

Autonomous driving vehicles are expected to have a huge positive impact on the automotive industry, e.g., to enhance road safety, ease road congestion, decrease fuel consumption and free the human drivers. Great efforts are undertaken in industry and academia on hardware and algorithmic research on the architecture of autonomous driving. The most popular paradigms are hierarchical scheme and end-to-end scheme.

The hierarchical scheme has been extensively studied during the last two decades. It decomposes the problem into several parts like environment perception, path planning, and motion control. Carnegie Mellon first presented BOSS autonomous driving car, which won the 2007 DARPA Urban Challenge. It used onboard sensors like GPS, lasers, radars, and cameras to percept the environment. Then a three-layer planning system combining mission, behavioral, and motion planning was adopted. The Boss car finished 96 km urban environment autonomous driving in about 4 hours [1]. Stanford also developed Junior in DARPA [2]. It was mainly composed of a perception module and a navigation module. The navigation module involved a finite state machine (FSM), which possessed 13 states, including lane-keeping and parking lot navigation. Further researchers introduced various improvements of the hierarchical scheme from different aspects. Furda et al. [3] adopted multiple criteria decision-making in the process of selecting the most appropriate driving maneuver, achieving safe autonomous driving in some typical urban traffic. Akai et al. [4] employed the

✉ Shengbo Eben Li
lishbo@tsinghua.edu.cn

[1] State Key Lab of Automotive Safety and Energy, School of Vehicle and Mobility, Tsinghua University, Beijing 100084, China

[2] Korea Advanced Institute of Science and Technology, 193, Munji-ro, Yuseong-gu, Daejeon, Korea

[3] DiDi Autonomous Driving Company, Beijing 100084, China

three-dimensional normal distribution transform (NDT) scan matching for accurate localization and a model predictive controller for vehicle motion control. The proposed autonomous driving system succeeded in mountainous public roads. Although these hierarchical methods have been investigated for years, they are inherently very complicated in structure and need carefully designing in every part, which can be a considerable project. This also leads to a prominent disadvantage of being prone to error propagation, as it was in the well-known case of the Tesla accident. A white trailer was misclassified as the sky in the perception module, resulting in dangerous planning and control.

Another autonomous driving scheme is the end-to-end method. End-to-end autonomous driving has a simple structure that directly uses raw sensor data as the inputs and outputs the low-level control command like steering angle and acceleration. This method is attractive due to its straightforward structure, which reduces the burden of designing complex modules. In addition, since all sensor data is directly used for driving, there will not be any perception information loss or error propagation as it is in a hierarchical scheme. End-to-end driving can be easily combined with imitation learning or reinforcement learning, which achieves state-of-the-art autonomous driving [5]. The end-to-end imitation learning is straightforward and effective, which aims to mimic driver's manipulation with neural networks [6]. The first try of this idea was from Pomerleau et al.[7] three decades ago. The proposed system took use of fully connected networks to map the camera and laser signal to steering angle. With the development of the convolutional neural network (CNN), Lecun et al. [8] managed to achieve end-to-end autonomous driving with CNN. This CNN was trained based on human driving data, which included videos from two cameras coupled with steering commands. The trained CNN was able to control a remote truck in a new environment while still avoiding collision. Then a turning point came. In 2016, NVDIA first realized end-to-end autonomous driving on real-world freeways [9]. Trained with driving data from humans, the developed system learned to drive on highways. It worked well even when visual guidance was unclear. However, all the experiments above have a fatal flaw in that they tend to terribly rely on a sea of labeled driving data. If these training data do not cover uncommon scenarios, the autonomous driving vehicles may probably fail. Facing urban traffic that is more complicated, the required data will rise at an exponential level that can hardly be collected. Besides, usually only safely driven data is provided in the training process, which means it cannot handle disturbance and unseen scenes caused by possible poor driving behavior.

To avoid the demand of a large amount of labeled data, reinforcement learning (RL) is regarded as a promising method for end-to-end autonomous driving. RL is a self-learning algorithm which learns by a trial-and-error fashion, i.e., it does not need explicit supervision from human [10, 11]. Consequently, researchers began to pay attention to the application of RL in autonomous driving [12, 13]. Yu et al. [14] applied Deep Q-network (DQN) to the webpage game JavaScript Racer. They took the raw pixel as input and nine discretized actions as the output. The trained controller succeeded in turning operation. Jaritz et al. [15] used the A3C reinforcement learning framework to learn vehicle control in a physically and graphically realistic rally game. They also proved that the trained networks drove well even on unseen tracks. Recently, some researchers have begin to consider training directly in the real world. Kendall et al. [16] developed an RL model that was able to learn a policy for lane-keeping in a handful of training episodes using a single monocular image as input. Their experiments showed RL agent could achieve decent performance only with less than thirty minutes of training.

Despite the success of the aforementioned work, they only feed the raw pixels into the decision model. Actually, there are sensors like engine speed sensors and wheel speed sensors in the real vehicle. Ignoring this accessible information in the driving process is a great waste. If these additional data can be combined with the pixels, the autonomous vehicle may potentially achieve much better driving performance. For instance, given the lateral speed that can hardly be deduced from images, the vehicle may be able to control the steering angle more precisely considering its motion state.

This paper adopts a modified version of the classical Deep Q-Networks (DQN), called Dueling Double DQN (DDDQN), to realize the end-to-end autonomous driving function. Instead of the traditional image-only input, a mixed state input which comprises both camera image and a vector of ego vehicle speed is proposed to provide additional information for vehicles. Furthermore, this paper also designs the corresponding dueling network architecture for the mixed state input, including a CNN to handle the raw pixels and fully connected layers for vehicle state. Our method is demonstrated on The Open Racing Car Simulator (TORCS), where it surpasses human drivers. Besides, this paper also visualizes the saliency map of the learned neural network and discovers the vehicle drives by observing the lane lines. Precisely, the contributions of this paper are as follows:

1. An end-to-end autonomous driving method through Dueling Double Deep Q-Network on TORCS. The state space, action space and reward function and implement RL algorithm is designed to realize end-to-end autonomous driving.
2. A mixed state input that comprises both camera image and a vector of ego vehicle speed. The corresponding dueling network architecture is designed, which includes both CNN and fully connected layers.

3. Visualization of neural network for end-to-end autonomous driving. The visualized saliency map of the learned neural network suggests that the vehicle drives by observing the lane lines.

The rest of this paper is organized as follows: Sect. 2 introduces the architecture of the end-to-end autonomous driving. Section 3 presents the deep reinforcement learning algorithm. Section 4 illustrates the design of state space, action space, reward functions and dueling network architecture. Section 5 discusses the experiment and training results. Section 6 concludes this paper.

2 End-to-End Autonomous Driving Architecture

Traditional hierarchical methods always decompose the problem into several parts like environment perception, path planning, and motion control, which are too complicated to design [1, 2, 17]. Instead, the structure of end-to-end autonomous driving is relatively simple and straightforward in that they directly map raw sensor data to vehicle control signals through deep neural network.

This research adopts the following end-to-end autonomous driving architecture which is illustrated in Fig. 1. At each time step, the environment, e.g., a simulator, sends the sensor data to the vehicle, which includes the frontal camera image, longitudinal speed, lateral speed, engine speed, and four wheel speeds. Both the images and the speed vector comprise a state representation of the current step. This state representation will be fed into the neural network, which is trained offline through a certain RL algorithm. Then the vehicle makes use of the policy network to select an optimal action, i.e., a steering angle. Finally, the vehicle performs this action in the simulator and it simulates the next step.

The adopted simulator in this paper is The Open Racing Car Simulator (TORCS). TORCS is a famous open-source car simulator widely used for autonomous driving research.

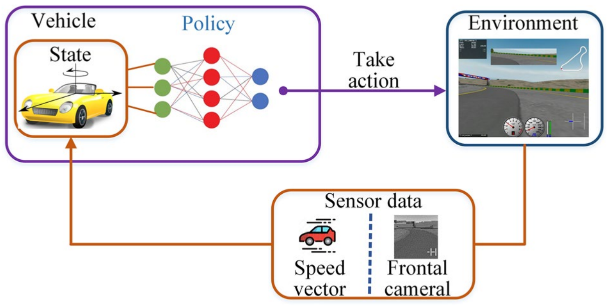

Fig. 1 End-to-end autonomous driving architecture

It provides users with an accessible vehicle dynamic model, different tracks and especially graphics with which users can train and test end-to-end autonomous driving neural network. TORCS also has a built-in physics engine with steering angle, velocity and acceleration as its control inputs and with camera image, vehicle position, velocity as its outputs. Through an API called gym-TORCS, researchers can easily input the control signals in TORCS and obtain the demanding sensor data.

The core component of end-to-end autonomous driving is the policy network, which serves as the brain of the ego vehicle. The following section will elaborate on the deep reinforcement learning algorithm used to train the optimal neural networks.

3 Deep Reinforcement Learning Algorithm

3.1 Preliminaries and Definitions

This paper adopts the Markov Decision Process (MDP) to formalize the reinforcement learning process. MDP consists of a set of state $\mathcal{S}$, a set of action $\mathcal{A}$, a reward function $r(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, a transition model $P(s'|s, a)$ and a discount factor γ. At each step of the decision-making process, the agent takes an action $a \in \mathcal{A}$, then receives a scalar reward signal r and reaches a new state s'. Here the reward r is a measurement of how good a decision is. It is a manually designed function that has a lower value when the vehicle enters a terrible state such as collision, and a higher value for a normal state such as keeping in the center of a lane. A policy $\pi(a|s) : \mathcal{S} \rightarrow \mathcal{A}$ is a mapping from state space to action space, which specifies what action will be take in a certain state. Reinforcement learning aims to find a policy π^* that achieves rewards as high as possible, i.e., maximizes the expected discounted total reward G_t in an episode.

The action-state values and state values are two primary quantities to organize and structure the search for good policies. For an agent behaving according to a stochastic policy π, the values of the state-action pair (s, a) and the state s are defined as follows

$$Q^{\pi}(s, a) \overset{def}{=} \mathbb{E}_{\pi}\left[\sum_{i=0}^{\infty} \gamma^{i} r_{t+i} | s_t = s, a_t = a\right] \tag{1}$$

$$V^{\pi}(s) \overset{def}{=} \mathbb{E}_{\pi}\left[\sum_{i=0}^{\infty} \gamma^{i} r_{t+i} | s_t = s\right] \tag{2}$$

The state-action values $Q^{\pi}(s, a)$ represent the expected return starting from s, taking the action a, and thereafter following policy π. Similarly, the state values $V^{\pi}(s)$ denote the expected return when starting in state s and directly following policy π now and thereafter. The optimal state-action values and state values are defined as

$$Q^*(s,a) \overset{def}{=} \max_{\pi} \mathbb{E}_{\pi} Q^{\pi}(s,a) \tag{3}$$

$$V^*(s) \overset{def}{=} \max_{\pi} \mathbb{E}_{\pi} V^{\pi}(s) \tag{4}$$

The optimal action value function obeys an inherent identity known as the Bellman optimality equation.

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r(s,a) + \gamma \max_{a'} Q^*\left(s',a'\right)\right] \tag{5}$$

The underlying idea of many RL algorithms is to iteratively solve the Bellman optimality equation, i.e., $Q_{i+1}(s,a) = \mathbb{E}_{s'}\left[r(s,a) + \gamma \max_{a'} Q_i\left(s',a'\right)\right]$. These methods are also named indirect RL [18]. Note that in practice, one normally lacks the state transition model, so the expectation needs to be computed by sampling experience, i.e., generate pair $\left(s,a,r,s'\right)$ by vehicle interacting with the environment. As i goes to infinity, it will finally converge to the optimal state-action values, given which one can readily derive the deterministic optimal policy

$$\pi^*(s) = \arg\max_{a} Q^*(s,a) \tag{6}$$

In a word, reinforcement learning estimates the optimal state-action values through the agent constantly interacting with the environment. As soon as the value converges, the optimal policy is obtained.

3.2 Deep Q-Network

Traditional tabular reinforcement learning stores the state-action values in a Q-table, which is impractical for high-dimensions tasks like end-to-end driving with image input. This problem is also described as the 'curse of dimensionality'. To handle that problem, a famous algorithm called Deep Q-network (DQN) was proposed [19]. Instead of Q-table, it approximates the state-action values with a deep neural network $Q(s,a;\theta)$, where θ is the parameter of the network.

The Q-network $Q(s,a;\theta)$ can be trained following the idea of solving the Bellman equation. It is reformed as an optimization problem, which allows us to implement gradient descent methods to minimize the following loss function:

$$J(\theta) = \frac{1}{2}\mathbb{E}_{s,a,r,s'}\left[y^{\text{DQN}} - Q(s,a;\theta)\right]^2 \tag{7}$$

where $y^{\text{DQN}} = r + \gamma \max_{a'} Q\left(s',a';\theta'\right)$ is the target of state-action value. DQN introduces two separate networks. The regular behavior network with parameter θ is used to make decisions while interacting. The target network with parameter θ' is used to form the target when training. The two networks working together are proved to make learning more stable [19].

Gradient descent is the most practical tool to optimize the loss function. Thus the loss function is differentiated with respect to the weights and arrive at the following semi-gradient, which provides the path to optimal state-action values.

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s,a,r,s'}\left[\left(y^{\text{DQN}} - Q(s,a;\theta)\right)\nabla_{\theta} Q(s,a;\theta)\right] \tag{8}$$

However, there is another concern about deep neural networks. During the training process of the target network with gradient, the training input must all be uncorrelated, which is not the case in RL. Since the experience is generated by a sequential decision-making task, it naturally correlates to each other. The experience replay mechanism is proposed to deal with this issue. The agent adds all its experiences to a replay buffer, which is then sampled randomly to perform updates on the network. The replay buffer will corrupt the relevance between the sampled experience and reduce variance [19].

Further researchers also develop a widely used trick called Double DQN (DDQN) [20]. It uses the current behavior network to calculate the argmax over next state values and the target network of that action, which will alleviate the overestimation problem induced by the max operator of RL, leading to better estimation. Double DQN trick is the same as DQN but only to replace the target y^{DQN} with

$$y^{\text{DDQN}} = r + \gamma Q(s', \arg\max_{a'} Q\left(s',a';\theta\right);\theta') \tag{9}$$

The corresponding semi-gradient of the neural network is

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s,a,r,s'}\left[\left(y^{\text{DDQN}} - Q(s,a;\theta)\right)\nabla_{\theta} Q(s,a;\theta)\right] \tag{10}$$

The key insight behind the whole algorithm is illustrated in Fig. 2. At each time step, the environment module generates state representation s. The behavior network output an action a according to ϵ-greedy policy π_{θ}^{ϵ}, i.e., most of the time choose the best action $a^* = \arg\max_{a} Q(s,a;\theta)$ but may also choose a random action with probability ϵ, to explore more in state space. Then the action will be performed and it reaches a new state s' with reward r. The experience $\left(s,a,r,s'\right)$ will all be immediately stored in the replay buffer. At the same time, the vehicle samples a small batch from the replay buffer. With the experience $\left(s,a,r,s'\right)$ in the batch, one can implement gradient descent of behavior network parameter θ according to Eq. (10). In addition, the target network will be updated with parameters of behavior network every τ steps. The above iteration process will be kept until the optimal policy is obtained.

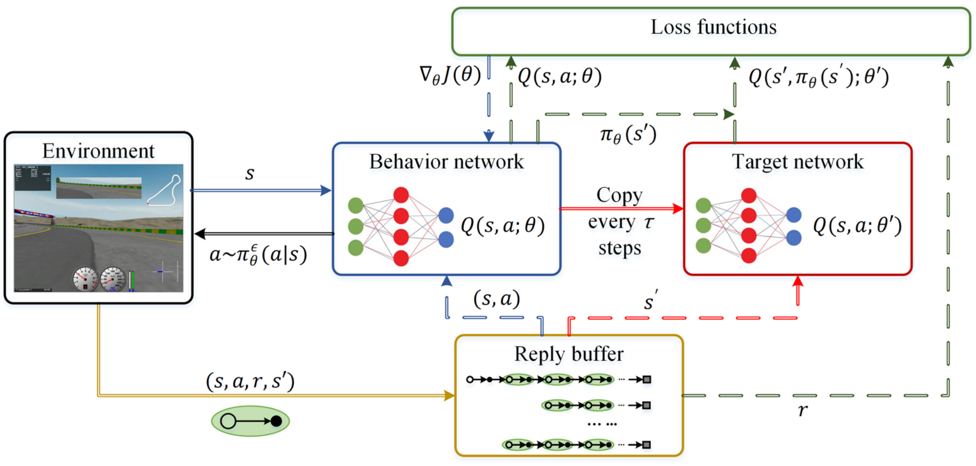

Fig. 2 Double DQN algorithm flow chart

When algorithm converges, neural network parameters are saved, which can be directly used as a controller for autonomous driving tasks like what is explained in the aforementioned end-to-end autonomous driving architecture in Fig. 1.

The pseudocode of Double DQN is given in Algorithm 1.

```
Algorithm 1 Double DQN
Input: $D^{replay}$:initialized replay buffer, $\theta$:weights for behavior action-value network, $\theta'$ :weights for target action-value network, $\tau$:frequency of updating target net
1:  for episode = 1:$M$ do
2:    Reset TORCS
3:    for steps $t$ = 1:$k$ do
4:      Sample action from policy $a \sim \pi_\theta^\epsilon$
5:      Play action $a$ and observe $(s', r)$
6:      Store $(s, a, r, s')$ in $D^{replay}$
7:      Calculate loss $\nabla_\theta J(\theta)$ via equation (10)
8:      Perform gradient decent step to update $\theta$
9:      if $t$ mod $\tau = 0$ then
10:       $\theta' \leftarrow \theta$
11:     end if
12:     $s \leftarrow s'$
13:   end for
14: end for
```

4 Lane-Keeping Through DDDQN

In this section, the RL algorithm Double DQN with Dueling Networks (DDDQN) is implemented on end-to-end autonomous driving, where the dueling networks are introduced to improve performance. The testing driving task is lane-keeping, i.e., the vehicle is expected to drive along the center of the lane without getting out of the lane.

This section first defines the state space of the task and designs the corresponding dueling network architecture. Then action space and reward function are explained.

4.1 Mixed State Space with Images and Speed Vector

In previous end-to-end reinforcement learning implementations, only the image pixels serve as the input to the neural network. However, in autonomous driving tasks, one actually has some off-the-shelf sensor data like engine speed or wheel speed. It is natural and sensible to consider feeding this data into the neural network apart from the images. With more information, it will make a possibly better decision. Note that this additional motion information is difficult to obtain just from images, so they are not redundant. With such an inspiration, this research chooses the mixed state space as the inputs for neural network

$$s_t \triangleq \left\{ s_t^{\text{spd}}, s_t^{\text{img}} \right\} \tag{11}$$

where s_t^{img} and s_t^{spd} represents processed frontal camera image and vehicle speed vector, respectively.

Specifically, the vehicle agent will observe and receive colored RGB images o_t^{img} in every step. Due to the limited computation and storage resource, the raw frames are preprocessed by converting their RGB representation to grayscale and down-sampling it to an 64×64 image s_t^{img}. In addition, the speed data from TORCS is extracted to define a speed vector s_t^{spd} which comprises longitudinal speed u, lateral speed v, engine speed v^{eng} and four wheel speeds $v_i^{\mathrm{whl}}, i = 1, 2, 3, 4$

$$s_t^{\mathrm{spd}} = \left[u, v, v^{\mathrm{eng}}, v_1^{\mathrm{whl}}, v_2^{\mathrm{whl}}, v_3^{\mathrm{whl}}, v_4^{\mathrm{whl}}\right]^{\mathrm{T}} \tag{12}$$

With the help of the extra motion information, a better control effect is expected to be realized. Furthermore, the next section will explain in detail how the image s_t^{img} and the speed vector s_t^{spd} are put into a custom-built neural network.

4.2 Dueling Network Architecture

Q-network outputs the state-action values for each action in a state, the architecture of which needs careful consideration. Recent research proves that structure directly estimating the action values of individual action is not very efficient, since the actions are usually relevant and may have similar values [21]. Instead, it is better to first estimate the state values and the relative advantages of every single action, which will then together derive the state-action values via some operations. Here advantage $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$ represents how a certain action a is better or worse than other actions in state s. This idea leads to a well-known architecture called Dueling Networks. It features two streams of computation, the value stream that outputs a scalar $V(s;\omega, \alpha)$, and the advantage stream that outputs an $|\mathcal{A}|$-dimensional vector $A(s, a;\omega, \beta)$, and they both share the same convolutional encoder. Here the parameters of the whole network θ is decomposed into three parts ω, η, β, where ω denotes the parameters of the convolutional layers, while η and β are the parameters of the two streams of fully connected layers. The two streams are merged by a special aggregator:

$$Q(s, a;\omega, \eta, \beta) = V(s;\omega, \eta) + \left(A(s, a;\omega, \beta) - \frac{1}{|\mathcal{A}|}\sum_{a'} A\left(s, a';\omega, \beta\right)\right) \tag{13}$$

The principal benefits of DDDQN lie in variance reduction and sampling efficiency improvement. The experiments in the next section will demonstrate how the dueling networks improve the performance.

Subsequently, the whole structure of the end-to-end Q-networks is introduced. As Fig. 3 shows, the 64×64 image will first be processed by three convolutional layers. The first convolutional layer has 32 8×8 filters with stride 4, the second 64 4×4 with stride 2 and the third one consists of 64 3×3 filters with stride 1. Then the output of the final convolutional layers will be unfolded into a 7744-vector and concatenated with the 7-vector speeds to form a 7751-vector. After that, the dueling network splits into two streams of full-connected layers. Both the value and advantage streams have three full-connected layers with the first layer having 128 units and the second layer having 32 units. The last hidden layer of value steam has one output $V(s)$ and the advantage counterpart $A(s, a)$ that has as many outputs as the number of feasible actions. Ultimately, the value and advantage streams converge to produce the final output using Eq. (13). In addition, there are rectifier nonlinearities between all adjacent layers.

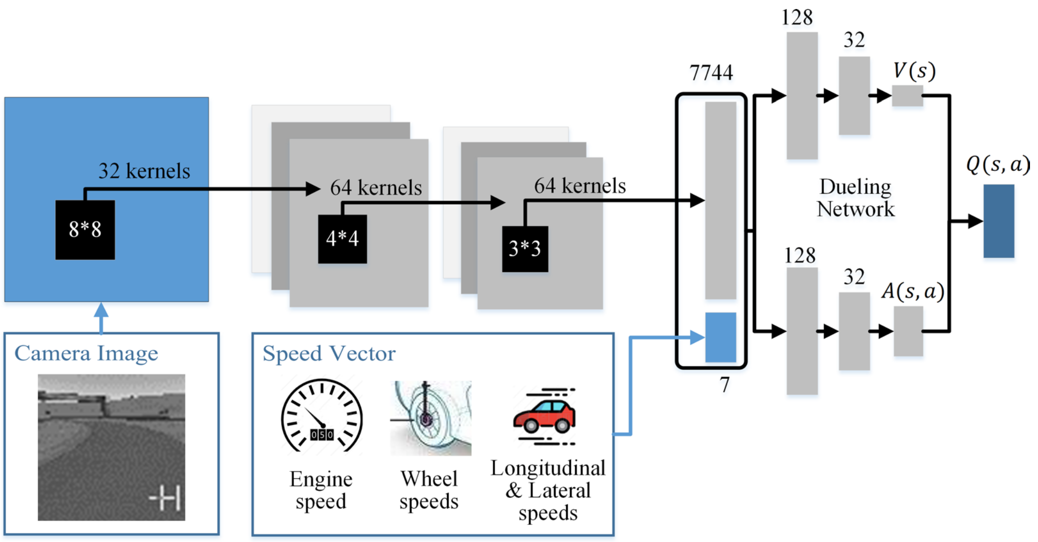

Fig. 3 Dueling network architecture for end-to-end autonomous driving

4.3 Discretized Action Space

TORCS accepts normalized steering angle command ranged from -1 to $+1$, where -1 and $+1$ means, respectively, full right and left that corresponds to an angle of 0.366519 rad. Considering that DQN can only be implemented on the problems with discrete actions, this paper discretizes the action space and sets 17 different normalized steering angle s: $\pm 0.25, \pm 0.20, \pm 0.15, \pm 0.10, \pm 0.05, \pm 0.02, \pm 0.01, \pm 0.005$, 0. Note that steering angle is divided more densely around 0 because this is helpful to drive more smoothly and steadily, especially on the straight road. Besides, since large steering angle command is not necessary for a lane-keeping task, the max steering angle is set to ± 0.25. During the training and testing, the longitudinal speed is controlled by a PID controller and the target speed is 80 km/h, while the target speed will be slowed down around the curves.

4.4 Reward Function

The reward function consists of three terms:

$$r = \lambda_1 \cos\varphi - \lambda_2 \left| \frac{P_y}{W_d} \right| - \lambda_3 I_{\text{fail}} \tag{14}$$

where $\lambda_1, \lambda_2, \lambda_3$ are nonnegative weight coefficient, φ is the angle between the road tangent and vehicle (see Fig. 4), W_d is the lane width, P_y is the lateral position error between the road center and the gravity center of the vehicle, which means $\left|\frac{P_y}{W_d}\right| \in [0, 1]$. I_{fail} is an indicative function

$$I_{\text{fail}} = \begin{cases} 1, & \text{if out of lane or stuck} \\ 0, & \text{others} \end{cases} \tag{15}$$

The first term in Eq. (14) is designed to encourage the vehicle to drive along the road. The second term plays as a punishment for deviating from the road center. The third term penalizes the vehicle heavily if it gets out of lane or stuck. This research chooses $\lambda_1 = 1, \lambda_2 = 1, \lambda_3 = 2$ in the following experiment.

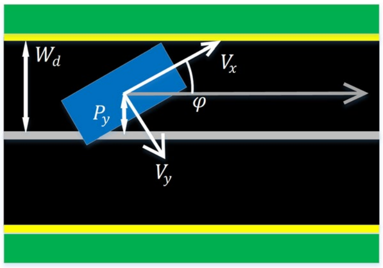

Fig. 4 Reward function representations

Fig. 5 Top view of the test track

5 Lane-Keeping Experiments

This paper trains and evaluates the end-to-end reinforcement learning autonomous driving method on TORCS. The Adaptive Moment Estimation (Adam) algorithm is adopted as the optimizers for networks because it is computationally efficient and converges fast [22]. The discount is set to $\gamma = 0.9$, and the learning rate to $\alpha = 0.0005$. The size of the experience replay memory is 10,000 tuples. The batch size for stochastic gradient descend is 32. The simple exploration policy used is ϵ-greedy policy with $\epsilon = 0.1$ all the time. The selected track for training and testing is Road Tracks CG Speedway No. 1 (see Fig. 5). The training platform is NVIDIA GTX 1650 and Intel i5-9400. The deep learning library is Pytorch. A single training run includes 400 episodes and it takes about 10 h. Note that the game will be reset if the vehicle is stuck or get out of lane in order to save time.

5.1 Evaluation Results

This paper chooses the average reward (the average reward per step in a whole episode) as the performance metric. The compared algorithms include DDDQN, DDQN, DQN and a human driver, who controls the vehicle manually with a keyboard after 30 min' practice. The learning curve of average reward during training is plotted by running five independent experiments for each RL algorithm. As shown in Fig. 6, all learning curves show that the average reward increases as the training goes. DDDQN has the best performance during the process, while DQN has the worst performance and large variance. DDQN is between the two methods. Apart from that, all three methods outperform a human driver only after a short training, revealing the potential of RL controllers.

After all algorithms converge, their lane-keeping performance is evaluated by using the learned network to drive 15 laps of the track. At the same time, a human driver also takes the same test of 15 laps. The video for the experiment is available online, https://youtu.be/76ciJmIHMD8 or https://v.youku.com/v_show/id_XNDM4ODc0MTM4NA==.html. The lateral position error P_y in a typical test lap is shown in Fig. 7. The DDDQN controller shows a great performance whose lateral position error varies in a very limited range

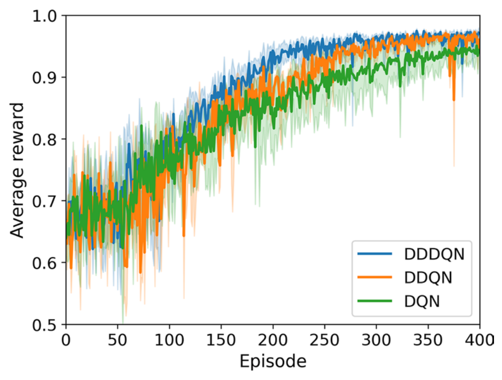

Fig. 6 Average reward during training

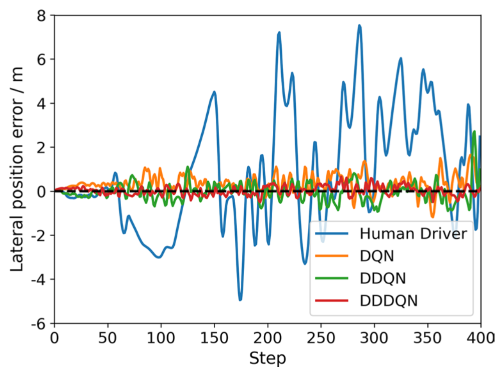

Fig. 7 Lateral position error in a typical lap with the learned network

around 0, even in the later steps when the vehicle is on a continuous bend. In contrast, the human driver has difficulty in controlling the vehicle, especially in later steps where there are many bends. It fluctuates dramatically. This figure reveals that the algorithm works quite well on the lane-keeping task.

Furthermore, to quantificationally assess the superior performance of DDDQN over others, the average reward and average lateral position error (the average lateral position error per step) in 15 laps are drawn in the boxplots Figs. 8 and 9. The proposed DDDQN dramatically outperforms the human driver. It wins many more rewards with less variance and the average lateral position error is even five times less than the human driver. DDDQN is also the best one compared with DDQN and DQN controllers, since it wins the highest average reward and least position error. This result confirms the introduction of a dueling network is helpful to achieve better control performance and the end-to-end RL controller is competent in this task. Besides, note that the tested policy is a noisy policy (ϵ-greedy policy) instead of a total greedy one. Thus, the vehicle may be driven to some rare place away from the road center, but the result shows the controller can drive it back since the average reward does not decline a lot. This result proves that the RL controller really learns to drive on the road, not just remembering a fixed driving routine.

5.2 Network Visualization and Saliency Maps

To gain deep insight into how the vehicle understands the camera image and makes decisions, this paper draws the saliency maps through the backward of the neural network, following the method proposed by Simonyan [23]. More specifically, the absolute value of the Jacobian of $\max_a Q(s,a)$ is computed with respect to the input frames to get a saliency matrix

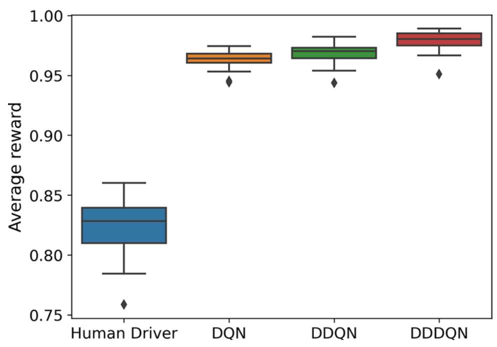

Fig. 8 Average reward of learned network

$$s^{\text{sal}} = \left| \nabla_{s^{\text{img}}} \max_a Q(s,a) \right| \tag{16}$$

This gradient matrix s^{sal} reveals which part of the image is salient and contributes most to the decision-making. Note that the matrix has the same dimensionality as the input frames, i.e., 64×64 in shape.

Although one can directly draw s^{sal} and the processed camera s^{img} relatively, like what Simonyan did in his paper, it is almost impossible to compare anything on such two low-resolution pictures. Therefore, it is necessary to process them and put them in one composite image.

This paper first captures the raw 640×480 frame o^{img} in TORCS, which is both colored and high-resolution. Second, the saliency matrix s^{sal} is resized into 640×480 and its values is normalized to $[0, 255]$. Then the colormap jet function is applied to convert it a colored map s^{map}. After that, with both o^{img} and s^{map} in the same shape, they can be combined to produce an identifiable saliency map $\text{Sal}(s)$:

$$\text{Sal}(s) = \lambda_s s^{\text{map}} + \left(1 - \lambda_s\right) o^{\text{img}} \tag{17}$$

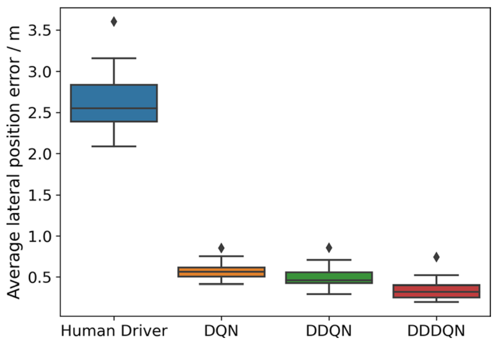

Fig. 9 Average lateral position error of learned network

where $\lambda_s = 0.1$ is the weight. Figure 10 depicts the saliency maps in different scenarios of left turn, right turn and straight line driving, from which the readers can easily find the network pays close attention to the lane lines as well as the road and horizon.

These results match the experience of human drivers. Intuitively, to locate and keep in the center of the lane, a driver must observe the position of lane lines all the time. Beside it is also necessary to focus on the road and horizon to make the decision of turning right or left.

6 Conclusions

Autonomous driving systems are on the rapid rise during the last decades. Especially, the end-to-end driving scheme has gained widely attention due to its simple structure compared with traditional hierarchical autonomous driving scheme. In this paper, the author applies a deep reinforcement learning algorithm DDDQN, and proves it is a powerful tool to realize end-to-end autonomous driving. This method is attractive because it does not rely on a great deal of labeled data or manually designed rules. The author first raises an architecture of end-to-end autonomous driving with both raw images and speed vector as its input, i.e., giving more information to the RL agent. Then, the action space, state space, reward function and dueling neural network architecture are designed for lane-keeping tasks. After that, the algorithm is trained on TORCS and the performance of the learned policy is evaluated. Experiments indicate the proposed method outperforms the human driver dramatically. To understand how the learned policy works, this paper also plots the saliency map and discovers that the vehicle drives depending on the observation of the lane lines and road. In conclusion, the presented RL architecture is a promising method for future end-to-end autonomous driving.

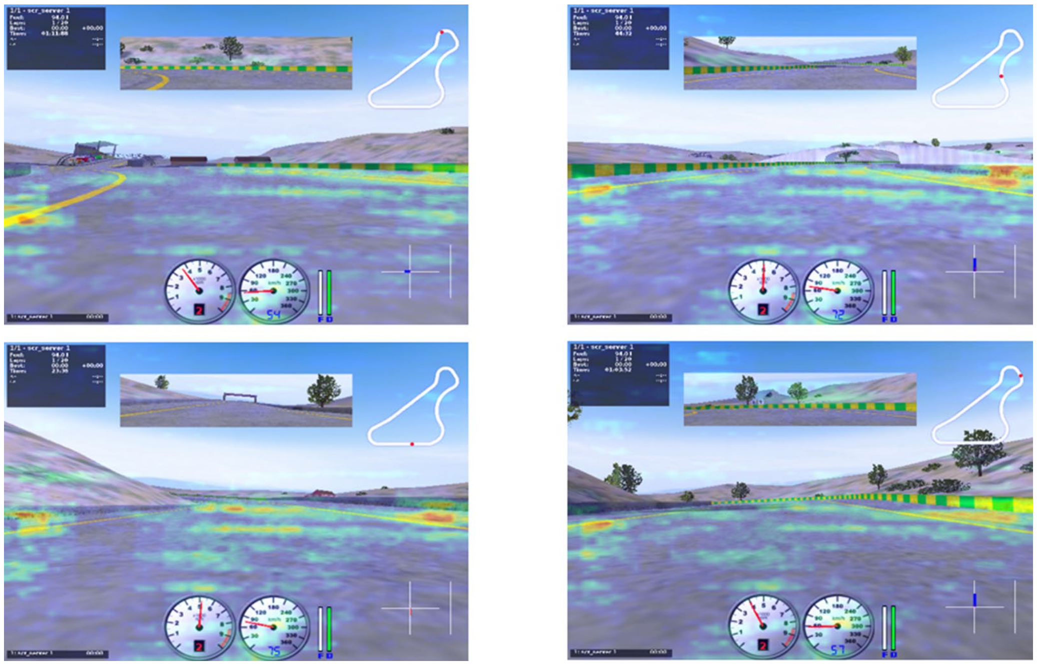

Fig. 10 Saliency maps in scenarios of left turn, right turn and straight line driving

In the future, the authors will extend the discretized action space to continuous action space and try to control the vehicle more smoothly. Besides, the potential of the method in different driving tasks and more complicated driving scenarios will be investigated.

Acknowledgements This work is supported by the National Key Research and Development Project of China under Grant 2018YFB1600600, and Beijing Natural Science Foundation with JQ18010. The authors should also thank the support from Tsinghua University-Didi Joint Research Center for Future Mobility.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Urmson, C., Anhalt, J., Bagnell, D., et al.: Autonomous driving in urban environments: boss and the urban challenge. J. Field Robot. **25**(8), 425–466 (2008). https://doi.org/10.1002/rob
2. Montemerlo, M., Becker, J., Bhat, S., et al.: Junior: the Stanford entry in the urban challenge. J. Field Robot. **25**(9), 569–597 (2008). https://doi.org/10.1002/rob
3. Furda, A., Vlacic, L.: Enabling safe autonomous driving in real-world city traffic using multiple criteria decision making. Intell Transp Syst Mag IEEE **3**(1), 4–17 (2011)
4. Akai, N., Saiki, L.Y.M, Yamaguchi, T., et al.: Autonomous driving based on accurate localization using multilayer LiDAR and dead reckoning. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems, pp. 1–6 (2017)
5. Li, S.E., Guan, Y., Hou, L., et al.: Key technique of deep neural network and its applications in autonomous driving. J Autom Saf Energy **10**(2), 119–145 (2019)
6. LeCun, Y., Bengio, Y., Hinton, G., et al.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539
7. Pomerleau, D.: Alvinn: an autonomous land vehicle in a neural network. In: Advances in Neural Information Processing Systems, NIPS Conference, Denver, Colorado, USA (1988)
8. LeCun, Y., Muller, U., Ben, J., et al.: Off-road obstacle avoidance through end-to-end learning. In: Advances in Neural Information Processing Systems. MIT Press (2005)
9. Bojarski, M., Del Testa, D., Dworakowski, D., et al.: End to end learning for self-driving cars. ArXiv abs/1604.07316 (2016)
10. Li, S.E.: Reinforcement learning and control. Tsinghua University-Lecture Notes (2019) http://www.idlab-tsinghua.com/thulab/labweb/publications.html
11. Duan, J., Li, S.E., Guan, Y., et al.: Hierarchical reinforcement learning for self-driving decision-making without reliance on labeled driving data. IET Intel. Transport Syst. **14**(5), 297–305 (2020)
12. Yin, Y., Li, S.E., Li, K., et al.: Self-learning drift control of automated vehicles beyond handling limit after rear-end collision. Transp. Saf. Environ. **2**(2), 97–105 (2020). https://doi.org/10.1093/tse/tdaa009
13. Li, S.E., Duan, J., Wang, W., et al.: Markov probabilistic decision making of self-driving cars in highway with random traffic flow: a simulation study. J. Intell. Connect. Veh. **1**(2), 77–84 (2018). https://doi.org/10.1108/JICV-01-2018-0003
14. Yu, A., Palefsky-Smith, R., Bedi, R.: Deep reinforcement learning for simulated autonomous vehicle control. Course Proj Reports (2016). https://doi.org/10.1016/0141-1136(95)00078-X
15. Jaritz, M., De Charette, R., Toromanoff, M., et al.: End-to-end race driving with deep reinforcement learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 2070–2075 (2018). https://doi.org/10.1109/ICRA.2018.8460934
16. Kendall, A., Hawke, J., Janz, D., et al.: Learning to drive in a day. In: 2019 IEEE International Conference on Robotics and Automation (ICRA), pp. 8248–8254 (2019)
17. Xin, L., Kong, Y., Li, S.E., et al.: Enable faster and smoother spatio-temporal trajectory planning for autonomous vehicles in constrained dynamic environment. Proc. Inst. Mech. Eng. Part D J. Autom. Eng. **235**(4), 1101–1112 (2020). https://doi.org/10.1177/0954407020906627
18. Guan, Y., Li, S.E., Duan, J., et al.: Direct and indirect reinforcement learning. ArXiv abs/1912.1 (2019)
19. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
20. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: 30th AAAI Conference on Artificial Intelligence, pp. 2094–2100 (2016)
21. Wang, Z., Schaul, T., Hessel, M., et al.: Dueling network architectures for deep reinforcement learning. 33rd International Conference on Machine Learning, 4:2939–2947 (2016)
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations (ICLR) (2015)
23. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: visualising image classification models and saliency maps. In: 2nd International Conference on Learning Representations, (2014)