Action Permissibility Prediction in Autonomous Driving through Deep Reinforcement Learning

Jiaoyang Wang¹, a, Qian Sun², *, Degui Yao², b, Feng Xiong¹, c

¹School of Mechatronic Engineering and Automation, Shanghai University, Shanghai, China
²State Grid Henan Electric Power Company Electric Power Science Research Institute, Henan 450000, China

*Corresponding author e-mail: xfeng@mail.shu.edu.cn, a269472124@qq.com, b1259424380@qq.com, c875828968@qq.com

Abstract. This paper proposes a new nature based on deep deterministic policy gradient for deep reinforcement learning in continuous state and action space, which can greatly accelerate the emergence of artificial intelligence training and solving problem: state-action permissibility. The proposed method is integrated into the latest DDPG algorithm to guide its training and is applied to solve the lane keeping (steering control) problem in autonomous driving or automatic driving. Finally, the TORCS which is the open racing car simulator simulation software builds various simulation environments, including different tracks to verify the effectiveness of the algorithm. The results show that the proposed method can significantly speed up the algorithm training speed of the lane keeping task.

1. Introduction
As Geoffrey Hinton said in the article Learning Internal Representations by Error Propagation, reinforcement learning is considered a powerful AI paradigm that can be used to teach machines by interacting with the environment [1] and learning from mistakes. Although Yoshua Bengio said in1990s practically, AI paradigm has not been successfully applied in automotive applications. In the real environment, the unmanned vehicle must be kept in the middle lane of the lane [2]. It is difficult to deal with the sudden situation according to the existing manual experience. Therefore, it is necessary to study a more intelligent algorithm to solve this problem said by Yann LeCun in 1995 [3].

Reinforcement learning has achieved some success in many areas, using DQN (deep Q network) algorithm combined with deep learning and reinforcement learning. This algorithm has proved that only the receiving pixel and game score can be used as input in the classic Atari 2600 game challenge [4]. Beyond all previous algorithms reached a level with professional players in a series of tests. However, DQN can only handle discrete and low-dimensional motion spaces, while the driving control of unmanned vehicles must output continuous steering wheel angle values and acceleration values, which is a continuous and high-dimensional motion space which is discovered by Jürgen Schmidhuber and his team IDSIA [5]. The DQN algorithm cannot be directly applied to continuous domains and cannot solve problems in the actual environment. In response to this situation, Google DeepMind developed a Deep DPG (DDPG) algorithm based on the Deterministic Policy Gradient (DPG) algorithm, which can control
the motion of unmanned vehicles while maintaining hyper parameters and continuous network structure [6].

2. Unmanned vehicle motion model parameter definition.

2.1. Sensor parameter input.

The steering, acceleration, braking and other actions of the unmanned vehicle must be output according to the DDPG algorithm, and the input environmental state parameters need to rely on the GPS, laser radar, camera, multi-wavelength millimeter wave radar to obtain the surrounding driving environment data. The device acts as a sensing module to input environmental state information to the decision module as an input parameter of the DDPG algorithm. The specific process is as follows:

```
Sensors collecting extract data feature and feature values.

Sensors collecting driving environmental data.

Data association is performed on the feature values and a common description is completed for the same target.

Different targets' feature values are converted into a dictionary format and input into the learning model as environmental parameters.
```

Figure 1. The flow of sensor input parameter.
2.2. Dynamic model of unmanned vehicle
The inertial position coordinates and the travel angle in the unmanned vehicle model are defined in the same manner as in the sports car model:

\[ x = v \cos(\delta_f) \]  \hspace{1cm} (1)
\[ y = v \sin(\delta_f) \]  \hspace{1cm} (2)
\[ \psi = \delta_f \]  \hspace{1cm} (3)
\[ v = a \]  \hspace{1cm} (4)
\[ x = v \cos(\psi + \beta) \]  \hspace{1cm} (5)
\[ y = v \sin(\psi + \beta) \]  \hspace{1cm} (6)
\[ \psi = \frac{v}{l_r} \sin(\beta) \]  \hspace{1cm} (7)
\[ \beta = \tan^{-1}\left(\frac{l_r}{l_f + l_r} \tan(\delta_f)\right) \]  \hspace{1cm} (8)

\( x \) and \( y \) are the coordinates of the centroid \((X,Y)\) in the inertial system. \( \psi \) is the direction of Inertia forward and \( v \) is the speed of the vehicle. \( a \) is the centroid acceleration direction of the same Position as the speed direction. The control inputs are the front and rear steering angles \( \delta_f \), \( \delta_r \), and \( a \).

Since the rear wheels cannot be turned in most vehicles, we assume \( \delta_r = 0 \). \( l_f \) and \( l_r \) represent the Distance from the center of mass of the vehicle to the front and rear axles. \( \beta \) is the angle of the current Center of mass relative to the longitudinal axis of the car.

3. Strategy design of the path of the unmanned vehicle remains

3.1. The design of the reward function.
In the improved strategy, we want to maximize the axial velocity, minimize the lateral velocity, and we punish the AI if it continues to deviate very far from the center of the road (collision or departure from the central axis of the road), the new proposed reward function is as follows:
From the reward function, we see that the car gets the maximum reward $V_x$ only when it is aligned with the track axis and $\delta_{\text{track}} = 0$; otherwise the reward will be less than $V_x$. Let $\delta_{\text{track},t}$ and $\delta_{\text{track},t+1}$ be the distance of the car from the lane center line (track axis) corresponding to state $s_t$ and state $s_{t+1}$ respectively.

$$r_t = \begin{cases} 
0, & \text{if } \delta_{\text{track},t+1} - \delta_{\text{track},t} \geq 0 \\
1, & \text{if } \delta_{\text{track},t+1} - \delta_{\text{track},t} \leq 0 \\
V_x (\cos \theta - \sin \theta - \delta_{\text{track}}) & \text{otherwise}
\end{cases} \quad (9)$$

3.2. The design of the explore function.

We use the Ornstein-Hollenbeck process to add noise to explore. The Ornstein-Hollenbeck process is as follows:

$$d_{x_t} = \varepsilon (\mu - x_t) dt + \sigma dW_t \quad (10)$$

It is a stochastic process with mean regression characteristics, where the $x$-return variable returns to the mean. $Y$ stands for balance or mean. $Z$ is the fluctuation of the process. Table 2 shows the recommended values used in the code:

| Action  | $\theta$ | $\mu$ | $\sigma$ |
|---------|----------|-------|----------|
| Steering | 0.6      | 0.0   | 0.30     |
| Acceleration | 1.0 | [0.3,0.6] | 0.10 |
| Brake   | 1.0      | -0.1  | 0.05     |

The most important parameter in the exploration process is acceleration $\mu$. You want to make the car have a certain initial speed, instead of getting into a local minimum (when the car has been braking on the brakes, no longer stepping on the throttle), you can experiment with the AI by changing the parameters and the behavior under the combination.

3.3. The improved DEEP-AP algorithm based on DDPG algorithm and its derivative algorithm.

There are always attributes of state-action permissibility (SAP) in the DDPG algorithm. This attribute causes an action to be allowed or disallowed in the new state when the agent reaches state $1$ after performing action $a_t$ under state $s_t$. If an action never leads to the best solution, then the action should not be executed in that state, so the action should not be attempted, causing the algorithm to fall into a local minimum infinite loop. Therefore, based on this algorithm in this paper, we incorporate the Action Permissibility (AP) predictor into the DDPG. This predictor model does not require any manually labeled training data (given allowed and not allowed). An example), the Action Permissibility (AP) function can be enabled by defining actions and the state-action permissibility (SAP) attribute can be used to automatically obtain this data during RL training. The integrated algorithm of our DDPG and AP predictor $E$ is called the DDPG-AP algorithm (as shown above in Table 2).

$E$ is predicted by the AP predictor in $t$ time, and then action selection $S_t$. 

(1) If $t < t_k$, the elemental ancestor of the environmental factor is required to be filled, and $N_t$ is the noise of the actor strategy.

(2) If $t \geq t_k$, the actor strategy will feed back to $E$ for AP prediction when the action selects $a_t$, and if the prediction action is unreasonable, it will sample from the action set with the sample size of the complete motion space using low variance uniformity. Find the allowed action for the current state. The value of $\alpha$ reflects the extent to which the actor participates in the AP predictor variable. The smaller the value of $\alpha$, the smaller the effect of the AP predictor on action selection (performed by the participant), and vice versa. In the experiments in this paper, we generally set $\alpha = 0.9$ when the verification accuracy $E$ is greater than 70%.

**Table 2. Algorithm of DDPG-AP.**

| Algorithm of DDPG-AP Action Selection |
|--------------------------------------|
| **Input:** Current state $s_t$, Actor $\mu(s|\theta^\mu)$ and AP predictor $E(s,a|\theta^E)$, the process of exploring noise $N_t$, step of time $t$, time step threshold $t_k$, probability $\alpha$ for consulting $E$ |
| **Output:** $a_t$ : Action selected for $s_t$ |
| 1: Select action $a_t = \mu(s|\theta^\mu)$ for $s_t$ |
| 2: if $t < t_k$ then |
| 3: $a_t = a_t + N_t$ |
| 4: end if |
| 5: if $t > t_k$ and $\text{Uniform}(0,1) < \alpha$ then |
| 6: $l(a_t) \leftarrow E(s_t, a_t|\theta^E)$ |
| 7: if $l(a_t)$ is ve (non-permissible) then |
| 8: using low-variance uniform sampling $A_{s_t}$ from $A$, build $D_{s_t}$ as $\{ (s_t, a) | a \in A_{s_t} \}$ |
| 9: $A_p(s_t) = \{ a \leftarrow E(s_t, a|s_t) + \text{ve}, (s_t, a) \in D_{s_t} \}$ |
| 10: if $A_p(s_t) \neq \phi$ then |
| 11: Sampling randomly $a_t$ from $A_p(s_t)$ |
| 12: end if |
| 13: end if |
| 14: end if |
| 15: Return $a_t$ |

**4. Simulation**
Simulation experiments were conducted using the open source standard automated driving simulator TORCS for learning and evaluation. The simulation experiment selects the Wheel-2, Spring, and E-road classic tracks in TORCS as the test track for the variant algorithm are shown in figure 2. The purpose of this experiment was to evaluate the lane keeping task, the process by which artificial intelligence learned to drive while positioning itself on the track/lane axle. In our experiments, we used five sensor readings to indicate that these state vectors are sufficient to learn good policies in various driving situations. As shown in Table 3:
Table 3. TORCS state and action variables along with their descriptions.

| State Variables | Name | Range(unit) | Description |
|------------------|------|-------------|-------------|
| angle            | [-π, +π](rad) | Angle between the car direction and the direction of the track center axis. |
| track            | (-∞, +∞) | Distance between the car and the track center axis. The value is normalized w.r.t to the track width: it is 0 when car is on the axis, -1 when the car is on the right edge of the track and +1 when it is on the left edge of the car. Values greater than 1 or smaller than -1 mean that the car is out of track center. |
| speedX           | (-∞, +∞)(km/h) | Speed of the car along the longitudinal axis of the car. |
| speedY           | (-∞, +∞)(km/h) | Speed of the car along the transverse axis of the car. |
| speedZ           | (-∞, +∞)(km/h) | Speed of the car along the Z-axis of the car. |

| Action           | Steering | [-1, 1] | Steering value: -1 and +1 means respectively full right and left, that corresponds to an angle of 0.366519 rad. |

We have also proposed some variants of the DDPG-AP algorithm to perform driving exploration strategies in different environments:

**DDPG-V1:** DDPG-V1 is an extension of the DDPG algorithm that applies the following two constraints to the motion space exploration: (1) If the car is on the left side of the track axis/lane centerline and the current action is in \( a_t > a_{t-1} \) (previous step) Action, then \( a_t \) will not be applied in the environment, it will uniformly sample the action from \((-1.0, a_{t-1})\). This constraint states that when the car is on the left side of the centerline, avoid turning to the left from the current position. (2) If the car is on the right side of the lane center line and \( a_t < a_{t-1} \), the action is sampled from \((a_{t-1}, 1.0)\). Therefore, further turn to the right should be avoided. Otherwise, the car will be executed here. In this case, the AP predictor variable will not be used for any guidance.

**DDPG-V2:** DDPG-V2 improves DDPG-V1. We only apply the above constraints when the situation became \( \delta_{\text{track},t} - \delta_{\text{track},t-1} > 0 \), only when the car leaves the track center due to its previous motion. If the car is near the center of the track, it is allowed. This method has a very strong constraint on the movement of the car. Again, this model does not use the AP predictor.

**DDPG-(AP+V2):** Combines V2 and AP predictors into the strongest constraint.

Figure 2. Test track for the variants of the DDPG-AP algorithm.

All Use each algorithm to drive a lap on each track and report the total rewards obtained by each algorithm, as shown in Figure 3:
Figure 3 shows the comparison of the average return of DDPG-AP and other variant algorithms in the training steps. In the training round, 15000 steps of training were carried out to show the average value of reward movement in the past 100 steps. In Figure 3, the average reward value of DDPG-AP increases very rapidly compared with other variant algorithms, and it is more stable than other algorithms (about 2500 steps tend to be stable). It also shows that DDPG-AP can quickly learn the strategy of keeping the car running stably on the track/lane. DDPG and DDPG-V1 are quite unstable. The performance of DDPG-V2 is better than that of DDPG and DDPG-V1, and the learning stability of combined algorithm DDPG - (AP + V2) is also better than that of DDPG-V2. But they are still not as stable as DDPG-AP.

5. Conclusion
In this paper, we propose a new property called state action permission to improve the training efficiency of AI. To take advantage of this feature, two new components are added to DDPG: AP function and AP predictor. They help DDPG algorithm choose the executable actions to accelerate training. Experimental results show that the method is effective. After further simulation test, it shows that adding AP predictor to DDPG algorithm can effectively improve the total reward value and training efficiency. In the future, the optimized algorithm framework can be extended to multi-dimensional continuous operation space and applied to other practical applications.

References
[1] SALLAB A E, ABDOU M, PERPT E, et al. Deep reinforcement learning framework for autonomous driving [J]. Electronic Imaging, 2017 (19): 70 - 76.
[2] MNINH V, KAVUKCUOGLU K, SILVER D, et al. Human-level control through deep reinforcement learning [J]. Nature, 2015, 518 (7540): 529 - 533.
[3] Wang Minglei, Chen Wuwei, Wang Jiaen. Obstacle avoidance path planning in intelligent vehicle lane keeping system [J]. Journal of Hefei University of Technology (NATURAL SCIENCE EDITION), 2014, 37 (02): 129 - 133.
[4] LILLICRAP T P, HUNT J J, PRITZEL A, et al. Continuous control with deep reinforcement learning [J]. Computer Science, 2015, 8 (6): A187.
[5] Sven A Beiker. “Legal aspects of autonomous driving”. In: Santa Clara L. Rev. 52 (2012), p. 1145.
[6] Yang Huihui, Ning Lijuan. Approximate unsteady solution of nonlinear drift Fokker Planck equation [J]. Acta physica Sinica, 2013, 62 (18): 38 - 45.