1 Introduction

1.1 Motivation

Car racing provides a unique and controlled environment for testing and advancing critical aspects of autonomous driving. While the objectives of racing and traditional road driving differ, the challenges inherent in racing align closely with key requirements for autonomous driving systems, such as real-time decision-making, handling dynamic environments, and optimizing complex behaviors under uncertainty. Tackling car racing tasks using Deep Reinforcement Learning (DRL) offers valuable insights and advancements that can be applied to broader autonomous driving applications.

Exploration is a cornerstone of reinforcement learning, particularly in car racing environments where:

Sparse and Delayed Rewards: Rewards, such as completing laps quickly or avoiding crashes, are often sparse and delayed, making it challenging to guide the agent effectively toward optimal policies. Complex State and Action Spaces: Racing tasks involve continuous control over acceleration, steering, and braking in a high-dimensional state space. Insufficient exploration can result in agents getting stuck in suboptimal behaviors, such as driving in circles or failing to navigate challenging corners.

1.2 Novelty

Designing an appropriate reward function for a car racing task can be non-trivial. Unlike traditional programming approaches, reinforcement learning shifts the focus from explicitly encoding specific behaviors to designing reward systems that enable agents to autonomously discover optimal policies. This highlights the need for improved exploration strategies to ensure that agents explore different trajectories, learn robust behaviors, and avoid premature convergence to suboptimal solutions.

In this project, we systematically evaluate and compare different exploration strategies, including noise network, dropout the policy layer, ICM and RE3, by leveraging A2C as a baseline. We gain insights into their impact on agent performance and highlight the broader applicability of DRL in advancing autonomous driving systems. Furthermore, we investigate the performance of traditional controllers MPC and PID in car racing and compare with our RL-based results. Further work is conducted on imitation learning.

2 Preliminary

2.1 A2C

A2C is a synchronous version of the Actor-Critic method. All workers or environments synchronize their gradients and parameters before performing updates. Stability in training due to synchronized updates across multiple environments. *Actor* is updated using the policy gradient:

$$\nabla_\theta J = \mathbb{E}[\nabla_\theta log\pi_\theta(a|s)A(s,a)].$$

Critic minimizes the temporal-difference (TD) error to learn the value function:

$$L(\phi) = \mathbb{E}[(r + \gamma V_\phi(s') - V_\phi(s))^2].$$

Compute the advantage $A(s,a)$ using either:

$$A(s,a) = r + \gamma V(s') - V(s) \quad (1 - step),$$
$$A(s,a) = \sum_{t=0}^{T} \gamma^t r_t - V(s) \quad (n - step).$$

Since we have multi process, Aggregate gradients from all workers and update the model.

2.2 Noise Network

NoisyNets are a class of neural networks in which weights and biases are perturbed by noise scaled by learnable parameters. These parameters are optimized through gradient descent. In the Advantage Actor-Critic (A2C) algorithm, the architecture typically includes a final fully connected layer with an input dimension matching the size of the embedding vector and an output dimension of one to approximate the value function. Additionally, another fully connected layer is used to generate actions, with the input dimension matching the embedding vector and the output dimension equal to the size of the action space.

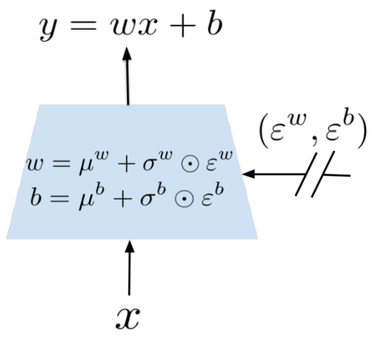
Building on the concept of noisy networks, the authors modify the final layer as follows:

$$y = wx + b$$

to

$$y = (\mu^w + \sigma^w \odot \epsilon^w)x + (\mu^b + \sigma^b \odot \epsilon^b),$$

where $\mu^w + \sigma^w \odot \epsilon^w$ and $\mu^b + \sigma^b \odot \epsilon^b$ replace w and b, respectively. The parameters μ^w,μ^b,σ^w,σ^b are learnable whereas ϵ^w and ϵ^b are noise random variables.



$y = wx + b$
$(\varepsilon^w, \varepsilon^b)$
$w = \mu^w + \sigma^w \odot \varepsilon^w$
$b = \mu^b + \sigma^b \odot \varepsilon^b$
x

Figure 1: Graphical representation of a noisy linear layer. The parameters μ_w , μ_b , σ_w and σ_b are the learnables of the network whereas ϵ_w and ϵ_b are noise variables which can be chosen in factorised or non-factorised fashion. The noisy layer functions similarly to the standard fully connected linear layer. The main difference is that in the noisy layer both the weights vector and the bias is perturbed by some parametric zero-mean noise, that is, the noisy weights and the noisy bias can be expressed as $w = \mu^w + \sigma^w \odot \epsilon^w$ and $b = \mu^b + \sigma^b \odot \epsilon^b$, respectively. The output of the noisy layer is then simply obtained as $y = wx + b$.

2.3 Random Encoders for Efficient Exploration

2.4 Intrinsic Curiosity Method

Making predictions directly in the raw sensory space is suboptimal, as it is not only computationally challenging to predict pixel values accurately, but also unclear whether pixel prediction aligns with the objectives of the underlying task. Instead, this method focuses on computing curiosity within a latent space, where features are more abstract and meaningful. To achieve this, the approach introduces three neural network components: a state encoder, a forward model, and an inverse model. These components collectively enable the agent to process and interpret latent representations, facilitating more effective learning and exploration. Assume we have the state s_t, s_{t+1} and a_t reprsent the observation at time t, the observation at time $t + 1$ and the action applied to agent at time t. The first sub-module encodes the raw state s_t in to a feature vector $\phi(s_t)$. The second sub-module or so-called inverse model takes the feacture vectors $\phi(s_{t+1})$ and $\phi(s_t)$ as inputs to predict the action a_t:

$$\hat{a}_t = g\left(s_t, s_{t+1}; \theta_I\right).$$

The loss function is $\min_{\theta_I} L_I(\hat{a}_t, a_t)$.

In addition to inverse dynamics model, this method also train another neural network (forward dynamics model) that takes as inputs a_t and $\phi(s_t)$ and predicts the feature encoding of the state at time step $t + 1$

$$\hat{\phi}(s_{t+1}) = f\left(\phi(s_t), a_t; \theta_F\right).$$

The loss function is $\frac{1}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|^2$.

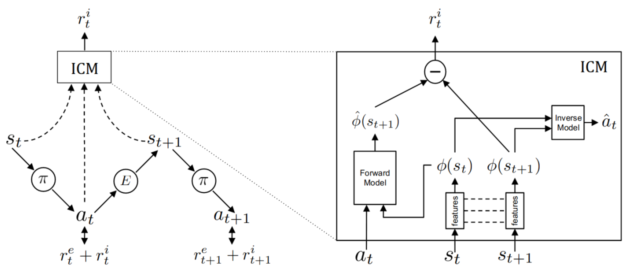


Figure 2: The structure of ICM

The intrinsic reward signal r_t^i is computed as

$$r_t^i = \frac{\eta}{2}\|\hat{\phi}(s_{t+1} - \phi(s_{t+1})\|^2,$$

where $\eta > 0$ is a scaling factor. Here is the back propagation of the ICM. The prediction error of the forward model is only used to train the forward model, not the encoder. The inverse prediction error of the inverse model is used to train both the inverse model and the encoder. The prediction error of the forward model is also the intrinsic reward (curiosity reward) to train our agent.

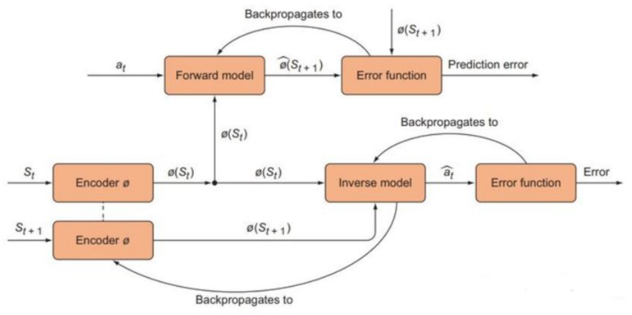


Figure 3: The back propagation of ICM.

2.5 Dropout for exploration

This method aims to enhance the exploration capability of reinforcement learning (RL) agents by incorporating dropout into the neural network architecture. Experiments were carried out in the MuJoCo simulation environment, including the HalfCheetah task, with Deep Deterministic Policy Gradient (DDPG) and Trust Region Policy Optimization (TRPO) selected as the RL models for evaluation.

3 Implementation details

3.1 Observation space and stating state

The car racing environment consists frames of game state, where each frame is a 96×96 RGB image of the car and race track, represented as $Box(0, 255, (96, 96, 3), uint8)$ in Gymnasium package.

3.2 Action space

We utilize the continuous action space: steering (-1 is full left and $+1$ is full right), gas and brake, representing as $Box([-1, 0, 0], 1.0, (3,), float32)$, as 3-dimensional array where action[0] is the steering direction, action[1] is gas pedal and action[2] is brake pedal.

3.3 Extrinsic Reward setting

The agent will receive reward -1 every frame and +1000/N for every track tile visited, where N is the total number of tiles visited in the track. If the car out of the edge of the environment then give -100 reward. The goal is to finish the race successfully in the least number of frames possible (fast).

4 Experiment

5 Conclusion

6 Others

6.1 MPC and PID

6.2 Survey of Imitation Learning

References

[1] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking,” in *2016 IEEE international conference on image processing (ICIP)*, 2016.