Implementation of Proximal Policy Optimization Algorithm for Autonomous Driving using AWS DeepRacer

Do-Yeon Kim[1], Joo-Ho Kim[1], Ye-Chan Song[1], Jae-Eun Park[1] and Young-Keun Kim*[1]

[1] School of Mechanical and Control Engineering, Handong Global University, Pohang. {21600053,21600144,21600369}@handong.edu

Abstract

This paper is focused on studying reinforcement learning for autonomous driving and implementing it on a small-scaled racing car. This paper designed and trained a PPO(Proximal Policy Optimization) algorithm for the autonomous driving in the simulation environment and deployed the trained model on the AWS DeepRacer to driving around the closed track. This paper explains how the reward function was designed for the optimal driving and how the hyperprameters were selected. The model was trained for four hours in the simulation environment provided by AWS. The designed reinforcement learning algorithm was validated by testing the trained model on the DeepRacer to circulate in the indoor track.

Keywords— *Reinforcement Learning, Deep Learning, Machine Learning*

I. Introduction

Developing self-driving system by using reinforcement learning has gained a high attention recently. Since it is difficult to train the agent in the real world environment, the reinforcement learning model is usually trained in a simulation environment using toolkits such as OpenAI gym and AWS DeepRacer simulator [1].

These toolkits allows the user create a custom environment of a simple track for a mini car. On these simulation environment, reinforcement algorithms can be developed and evaluated to modify the model for the optimal performance.

However, there exists some difficulties on deploying trained model on the actual world environment. Some essential input data such as precise location and heading direction may not be available in the actual world as they are in the simulation.

AWS DeepRacer is a recent system that has a high integration between the simulation environment and the real world for the autonomous driving of a mini car. It provides an online service to train and evaluate the reinforcement learning models in a simple simulated environment. The tranied model can be easily deployed on the DeepRacer vehicle of 1/18th scale RC car.

As a study on reinforcement learning for autonomous driving, this paper designed and trained PPO(Proximal Policy Optimization) algorithm [4] on the DeepRacer console and deployed on the vehicle for validation. This paper describes how the reward function and hyperparameters are designed and modified based on the learning rate and progress rate performance. Also, the experiment setup in the real-world track is expalined and the outcome is analyzed to validate the proposed algorithm.

II. Design Process of PPO Algorithm

The Proximal Policy Optimization (PPO) algorithm is a policy gradient method for reinforcement learning that optimally update based on the two networks of Actor and Critic. It is similar to well-known algorithms such as Actor-Critic, A2C, and A3C. However, PPO re-uses the a batch of experience data even after newly updating the policy. PPO leads to a less variance in training and helps the agent deviating to meaningless actions.

PPO aims to find the optimal parameter θ that maximizes the expected rewards by updating θ to minimize the cost function by using Trust Region method. It uses the ratio between the new policy(p_θ) and old policy($p_{\theta_{old}}$).

$$\theta \leftarrow \theta + \nabla_\theta \sum_{i=t-N+1}^{t} \frac{p_\theta(s_t,a_t)}{p_{\theta_{old}}(s_t,a_t)} \tag{1}$$

Instead of using a complex KL constraint, as in TRPO [2], PPO uses the policy ratio $r(\theta)$=(p_θ)/$(p_{\theta_{old}})$ to stay within a small interval ε around 1.

The objective function J_i of PPO that is maximized through the policy update is expressed as

$$J_i = \sum_{i=t-N+1}^{t} \frac{p_\theta(s_t,a_t)}{p_{\theta_{old}}(s_t,a_t)} A_i \tag{2}$$

where advantage A_i is the difference between the action value function $Q(a_i|s_i)$ and the state value function $V(s_i)$

for the action a_i at the state s_i. The positive advantage means the action taken by the agent is preferable. Since it is difficult to obtain the action value function directly, PPO estimates the advantage using generalized advantage estimation (GAE) [3].

$$A_i \triangleq Q(a_i|s_i) - V(s_i) \approx \sum_{k=i}^{t} (\gamma\lambda)^{k-i}\delta_k \tag{3}$$

where γ and λ are the GAE parameters and δ_k is the temporal-difference(TD) error defined by the reward R and state value function V.

$$\delta_k = R_{k+1} + \gamma V(s_{k+1}) - V(s_k) \tag{4}$$

This paper applied the clipped surrogate objective J_i^{clip} that saturates the advantage at a certain value and is known to have a better experimental performance than other surrogate objective functions.

$$J_i^{clip} \triangleq \min[ratio_i, A_i, clip(r_i 1-\varepsilon, 1+\varepsilon)A_i] \tag{5}$$

The update algorithm for PPO is summarized as shown in Figure 1.

Algorithm PPO

0. *Initialize* θ, w
Repeat 1~4
  1. *Collect N Sample (sample:* $\{s_i, a_i, s_{i+1}\}$)
  Repeat 2~3(*Epoch*)
    2. *Actor update:* $\theta \leftarrow \theta + \alpha\nabla_\theta \sum_{i=t-N+1}^{t} J_i^{clip}$
    3. *Critic update:* $w \leftarrow w - \beta\nabla_\theta \sum_{i=t-N+1}^{t} \left(A_i^{GAE}\right)^2$
  4. *Clear the batch*

Fig. 1. PPO Update Algorithm

III. SIMULATION

A. Reward Function

The simulation environment of AWS console provides the vehicle data of the 2D position (x_{car}, y_{car}), heading angle (H), and steering angle (θ_s). The reward function from these parameter values is designed as shown in Fig 2.

Reward Function

Initializing & Updating Paramters
  $H = car\ heading\ angle$
  $\theta_s = steering\ angle\ of\ car$
  $x_t, x_y = target\ point$
  $x_{car}, y_{car} = car\ point$
  $eps = error\ disired\ by\ the\ user$
  $dx = x_t - x_{car},\ \ dy = y_t - y_{car}$
  $\theta_t = polar(dx, dy)$
  $\theta_{best\ angle} = \theta_t - H$

reward
  $if(\theta_{best\ angle} - \theta_s < eps)$
    get *reward*

Fig. 2. Overview of the Reward Function

The key idea is giving a positive reward if the steering angle of the wheels (θ_s) is similar to the net heading angle (θ_{net}) towards the target point from the current position of the vehicle. The target points are set as the points on the center line of the track at a certain distance away. If the steering angle (θ_s) is exactly the same as the net heading angle (θ_{net}), then the vehicle would drive directly to the target point.

B. Hyperparameters

Hyperparameter	Value
Gradient descent batch size	64
Entropy	0.01
Discount factor	0.95
Loss type	Huber
Learning rate	0.0003
Number of experience episodes between each policy-updating iteration	20
Number of epochs	10

Fig. 3. Hyperparamerters

The hyperparameters for training are set up as shown in Figure 3.The batch size was 64, not a large size, and the

epoch count was 10 to speed up the learning. In addition, the discount factor was set to 0.95, giving a large weight to the present state. It used Huber loss that is a compromised loss function between MSE and MAE.

C. *Simulation Track*

Fig. 4. Simulation Program

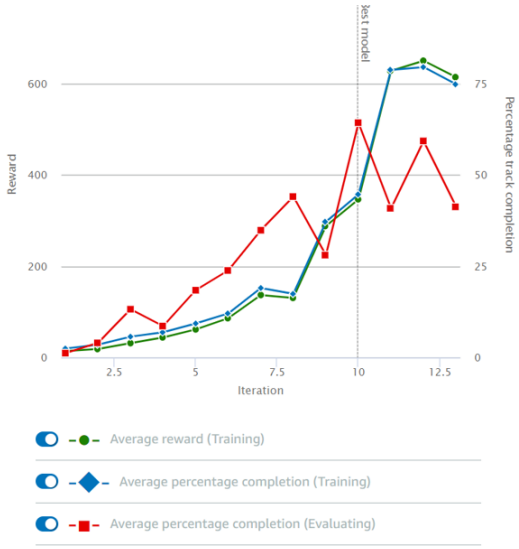

Fig. 5. Result Graph of Simulation

The training in the AWS console took four hours and the mean reward is shown to increases with the iterations, as shown in Figure 5. Furthermore, the average percentage completion of training has also seemed to be increasing continuously.

IV. Real-World Implementation

The trained model was deployed on the DeepRacer vehicle with the specification as shown in Table 1

Table 1. Specs of Deep Racer

Car	18th scale of real car
CPU	Intel atom Processor
Memory	4GB RAM
Storage	32GB Memory
Wi-Fi	802.11ac
Camera	4 MP Camera with MJPEG
Softwatre	Ubuntu OS 20.04.3

Fig. 6. Real Track

Fig. 7. DeepRacer driving within straight and curved lanes

The DeepRacer vehicle was tested on a small scale RC car track, as shown in Figure 6. Although it was a different environment from the simulation track, the vehicle managed to drive well without deviating off from the lanes. With the four hour trained model, the vehicle could be driven both the straight and curved lanes without any lane departure.

V. Conclusion

This paper studied PPO of reinforcement learning to implement an autonomous driving RC car using the AWS

DeepRacer vehicle. The reward function was designed to give a positive reward when the wheel steering angle is similar to the net heading angle of the vehicle towards the target point. The PPO model was trained in the simulation environment of AWS DeepRacer console and deployed on the vehicle. The driving test on the actual track showed the vehicle was able to drive within both the straight and curved lanes without any departure. The model could be improved with more training and modifying the designed simple reward function.

REFERENCES

[1] Bharathan Balaji, Sunil Mallya, Sahika Genc, Saurabh Gupta, Leo Dirac, Vineet Khare, Gourav Roy, Tao Sun, Yunzhe Tao, Brian Townsend, Eddie Calleja, Sunil Muralidhara, and Dhanasekar Karuppasamy. Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning. *CoRR*, abs/1911.01562, 2019.

[2] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[3] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.