Deep Reinforcement Learning for Virtual Self-Driving: An Algorithm Study

Kohei Nakano s1240233 Supervised by Dr. Xiang Li

Abstract

Reinforcement learning is considered to be a strong artificial intelligence paradigm that can be used to teach machines through interaction with the environment and learning from trial and error. Self-driving is big challenge because it requires very accurate and safe driving when driving on public road. In recent years, several important studies, for example, vehicle following, lane keeping assist, lane switching are realized. But full self-driving (as Level 5) is difficult to realize now because of some problems like capability of AI technology or training verification problems. In this study, we use Unity ML-Agents toolkit for self-driving learning because we considered Unity environment can reproducible closer to the real-world environment and validate the training by implementing deep reinforcement learning algorithms.

1. Introduction

This study focuses on algorithms of deep reinforcement learning (DRL) for self-driving. DRL is a method that fused Reinforcement Learning (RL) and Deep Learning (DL). RL is a method that maximize reward by trial and error, and DL is a method that extract feature by using Deep Neural Network (DNN). In this self-driving area, previous study has shown self-driving with Convolutional Neural Network (CNN), Double Deep Q-Network (Double DQN), and Deep Q-Network (DQN) [1], self-driving with DQN and Deep Deterministic Actor Critic (DDAC) [2], and Inverse Reinforcement Learning (IRL) and DQN [3]. They are famous for various purpose like AlphaGo [4] and video games. For the application side, from Tesla, Toyota, Honda, and many other car manufacture companies pay a lot of attention on self-driving studies, we hope this research may contribute to DRL factor. Virtual self-driving by ML-Agents [5] can be nearer to real world environment comparing with other 2D/3D car simulation environment. And Unity environment can make change to create complex situations with physics engine. Therefore, this research will try to evaluate the performance with more efficient algorithms like Proximal Policy Optimization (PPO) [6] and Soft Actor-Critic (SAC) [7]

2. Method

2.1 Overview

Self-driving is driving a car autonomously without human intervention. The reinforcement learning highly relies on trial and error, thus experiments in the real world are difficult considering vehicles preparation and environmental impact. In this study, we will use Unity environment and ML-Agents. ML-Agents is toolkit that was offered by Unity Technologies. It is for machine learning, especially RL in Unity environment. PPO is less computational complexity than Trust Region Policy Optimization (TRPO), it is the default learning method at OpenAI baselines. And PPO method is simpler to implement and tune than another state-of-the-art algorithm as well. Also, we use the programming environment of Python and TensorFlow. TensorFlow is an open source software library for machine learning. TensorFlow is used by various purpose like natural language processing, object detection, and pattern recognition. In this study, there is 2 steps for experiment. The 1st step, we will make agent learn how to drive in Unity environment with ML-Agents, Python, and TensorFlow. The method we will use in this experiment are PPO and SAC. In ML-Agents environment, there are Academy, Brain and agent to execute reinforcement learning. Academy is an object that management learning environment. Brain is an object that determines an agent's action according to the state observed by the agent. Agent is an object that act independently in the learning environment. In this environment, we focus on speed to go around the circuit, therefore the better lap time contribute to reward. The 2nd step, evaluate each experiment from the perspective of accuracy of driving and efficiency of driving. Accuracy of driving means how the car control their car lane and how the car drive without problems. Efficiency of driving means how fast the car laps the track at certain time.

2.2 Environments

Our environment in Unity contains circuit, walls, 5 cars. The circuit has inclination at the edge of the road. This environment is motivated from the Udacity Course [8]. And this environment consists of Unity Technologies' Standard Assets (for Unity 2017.3) [9] and NIANDREI's Lake Race Track [10] (see Figure 1 for a reference). There is no signal and no speed limit in this environment. The wall looks like obstacles or pedestrians, so it appears in a certain time and disappears after a period. The base of the car is Unity

Standard Assets car. Each 5 cars have 5 raycast sensors that can detect road, and 5 raycast sensors that can detect obstacle like other cars and walls. Raycast is one of the Unity's own program that returns true if the ray was hit and hit the specified collider, false otherwise.

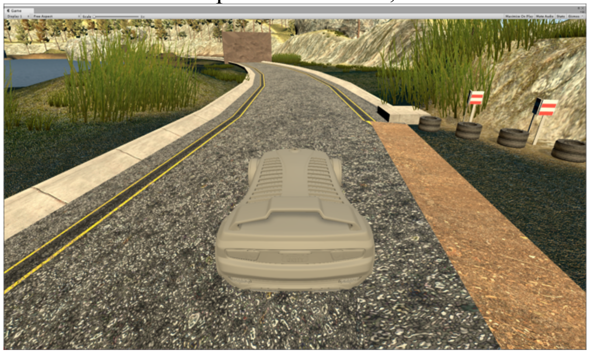

Figure 1: Screenshot of the experiment's environment in Unity

2.3 Rewards and Punishments

2.3.1 Out of course and other termination conditions

In self-driving situation, course out and car crush is the event that must not happen. Therefore, when the car reach to 3 unfavorable situations, give very big punishment. First, when the car goes out of the course. It is judged by touching the car collider to another collider of the road. Second, when the car stops or is too slow, it will give little punishment and count that the numbers. After counter reaches to certain limit (in this environment, 300 times), it gives very big punishment because improving the speed of learning because the car remains stationary in the early stage of learning. Third, when the cars collide each other, give very big punishment and additional punishment to the agent. Because it is less happening in this environment, therefore agent needs to learn with small number. When the agent goes around the circuit, it records the lap time, terminates the agent and reappears in the first position of each agents.

2.3.2 Center of road

In driving situation, keeping center position of road is very important. Therefore, I added 5 raycast sensors to the car to detect distance to the car to road edge. These sensors angles are $0°, 45°, 90°, 135°$ and $180°$ that starts from right side of car. These sensors help keeping the car center of road and avoiding car collision. If the car is not at center of the road, give little punishment to the agent. And the punishment happens every frame when the car is not on center position, therefore at final stage of learning, the car can drive at center position almost for all of frame.

2.3.3 Best lap time

This best lap time reward is given when the agent updates best lap time. Because in this environment, there is no legal speed therefore less lap time get more reward. In addition, there is 5 Check points in this environment and when agent through these checkpoints, the agent gets rewards.

2.3.4 Go forward

In the early stage of learning, the car drive back and forth. Therefore, to increase learning speed, give reward only when the car drive forward. And to update best lap time, increase reward based on distance traveled.

2.3.5 Counting breakdowns

In this study, focusing on lap time, breakdowns can become problematic. Therefore, to speed up learning, the program gives little punishments when breakdowns occur. But it is just a little punishment, agent can choose breakdown when needed like the car wants to curve.

2.3.6 Car and wall detection and breaking

There are other cars, walls and pedestrians in the real world, therefore we need to detect those things. In this environment, the car is with 5 raycast sensors to detect other cars and walls. These sensors angles are $70°, 80°, 90°, 100°$, and $110°$ that starts from right side of the car. In this environment, we regard the wall as obstacles and pedestrians, and the wall appear at every certain time and disappear after a period like pedestrian crossing the road. When the car detects other car, no reward by acceleration because when the car detects others, too much acceleration makes car collide. But slowing down just because the agent finds another car can cause traffic jam, therefore the agent can get little reward by slowing down, and car get closer to another car and keep chasing gets reward. And when meeting another car, it will stop counting by avoiding too long time stop. And when waiting for pedestrians to go cross the road, drivers often stop just before and wait, therefore, if the agent stop just before the wall and wait, agent gets reward.

2.4 DRL algorithm: PPO

In this study, PPO [6] is used by agents to learn. PPO is algorithm with similarity to Trust Region Policy Optimization (TRPO) [11] algorithm. Traditional algorithm includes the following gradient formula [6]:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[log\pi_\theta(a_t|s_t)\hat{A}_t\right] \tag{1}$$

L^{PG} is loss of policy gradient, θ is vector policy parameters, $\hat{A}_t$ is advantage estimator at time t, a_t is action at the time t, s_t is the state at the time t, and $\hat{\mathbb{E}}$ indicates the empirical average over a finite batch of samples, in an algorithm that alternates between sampling and optimization. In policy $\pi(a|s)$ of this formula, the value sometimes changes significantly at the time of updating, which may have an adverse effect on learning. But TRPO method includes the following objective formula [6]:

$$\max_{\theta} \widehat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{pld}}(a_t|s_t)} \hat{A}_t - \beta KL\left[\pi_{\theta_{old}}(\,\cdot\,|s_t), \pi_\theta(\cdot\,|s_t)\right] \right] \quad (2)$$

Coefficient β is following form the fact that a certain surrogate objective forms a lower bound on the performance of the policy π. KL is the Kullback-Leibler divergence that measure the magnitude of change between π_{old} before updating and π after updating. The provision of KL in the penalty term prevents sudden changes in policy. But TRPO is complex implementation, cannot use dropout method, and cannot adapt to Actor-Critic (AC) [12]. PPO was proposed to improve them. The point of PPO is, when putting $r_t(\theta)$ as follows, it becomes 1 when completely equal to π_{old} and π [6]:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \text{ so } r(\theta_{old}) = 1$$

$$L^{CPI}(\theta) = \widehat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] = \widehat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right] \quad (3)$$

L^{CPI} is loss that refers to conservation policy iteration. $r_t(\theta)$ denote the probability ratio. Namely, when focusing on whether $r_t(\theta)$ deviates from 1, deviation from the original distribution can be observed, and the following equation is obtained [6]:

$$L^{CLIP}(\theta) = \widehat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right]\right. \quad (4)$$

ϵ is a hyperparameter, and L^{CLIP} is a loss when $r_t(\theta)$ was clipped between $1-\epsilon$ and $1+\epsilon$. PPO, compared to TRPO method, is easier to implementation, and has better sample efficiency.

2.5 DRL Algorithm: SAC

In this study, SAC [7] is also used to learn. SAC is simply the ordinally AC objective function plus the entropy maximization term of the strategy. AC [12] is a model-free and off-policy algorithm for reinforcement learning. Actor is a following formula that calculate action [13]:

$$\nabla\eta(\theta_p) \sim \frac{1}{T}\sum_{t=1}^{T}\{log\pi(a_t|s_t, \theta_p)\}\delta \quad (5)$$

$\nabla\eta(\theta_p)$ indicates the corresponding gradient of the parameter θ_p for each action, T is total number of trials, δ is error, and $\pi(a_t, s_t)$ is policy when action is a_t that is action at the time t and state is s_t that is state at the time t. And critic is a following formula that calculate Temporal-Difference (TD) error [13]:

$$\delta = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \quad (6)$$

Here, r is reward, γ is attenuation rate, and V is value function. Therefore, in calculation of AC need these 2 steps. In standard reinforcement learning, the aim is to maximize the next expected sum of rewards formula [7]:

$$\sum_t \mathbb{E}_{(s_t|a_t)\sim\rho_\pi}[r(s_t, a_t)] \quad (7)$$

$\mathbb{E}$ is expected reward, ρ_π denote state and state-action marginals of the trajectory distribution induced by a policy π. On the other hand, in SAC, add entropy maximizing term to the above objective function [7]:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha H(\pi(\cdot\,|s_t))] \quad (8)$$

α is temperature parameter that is hyperparameter that determines how much the entropy term H is taken into account. By maximizing the above objective function, SAC realizes an algorithm that has both sample efficiency and stability.

3. Results

This experiment used PPO and SAC that is provided by Unity Technologies. Number of steps is $1.0e7$ and use normalize option. The comparison of the number of terminates for each step of PPO and SAC is in the Figure 2. As it shows, PPO has less terminating than SAC in middle of learning, and in the end of learning there is big difference of number of terminates. The terminate conditions of this environment are going around the circuit or stopping too much, going out of the course, and car collision by each other. And the figure shows number of terminates except for car goes around circuit. Therefore, less terminates means more times the car goes around the circuit in the certain steps. And comparison of the transition of the best lap time between PPO and SAC is in the Figure 3. Each graph gave the best result among the five agents in each method. Even at best lap time, the PPO is significantly different from SAC. In this experiment, because the emphasis is on the speed of the car, the less the lap time means the better the learning result.

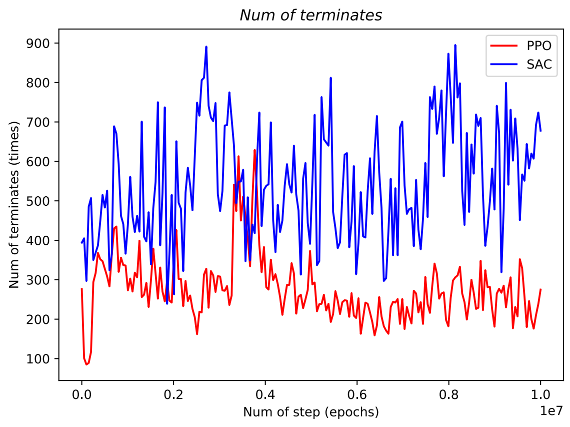

Figure 2: Comparison of the number of terminates for each step of PPO and SAC

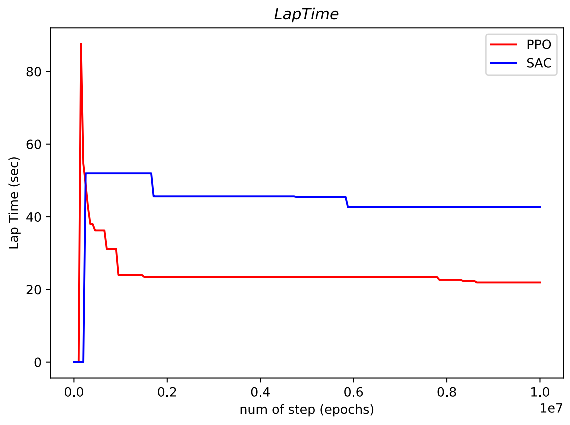

Figure 3: Comparison of transition of the best lap time of PPO and SAC

Figure 4, 5, and 6 is from TensorBoard, therefore it uses smoothing for clarity. Figure 4 shows transition of cumulative reward of PPO and SAC. It illustrates that, PPO gets more reward than SAC. This indicates that PPO is learning more effective than SAC. Figure 5 shows transition of the entropy of PPO and SAC. Looking at entropy, it can be seen that SAC has a higher entropy from the beginning of learning than PPO. It shows how random the model decision is. Low entropy means that the decision is stable. Therefore, agent that learns with PPO can choose stable action, a smaller number of terminates, and can goes around circuit faster. Figure 6 shows transition of the value loss of PPO and SAC. As shown in Figure 6, the value loss of SAC at the end of learning is little lower than the value loss of PPO. The meaning of value loss of the value function is loss when updating values. It correlates to how well the model is able to predict the value of each state. These values will increase as the reward increases, and then should decrease once reward becomes stable. As shown in Figure 4, the value of the cumulative reward has been stable with a low value. It means there is little increasing of cumulative reward. Considering the above results, in this environment and reward setting, PPO algorithm shows better result than SAC algorithm.

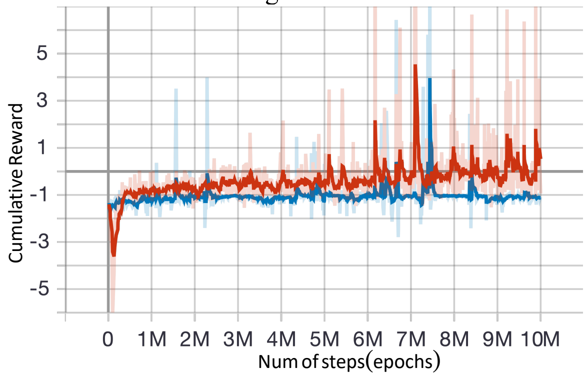

Figure 4: Comparison of transition of cumulative reward of PPO (red) and SAC (blue).

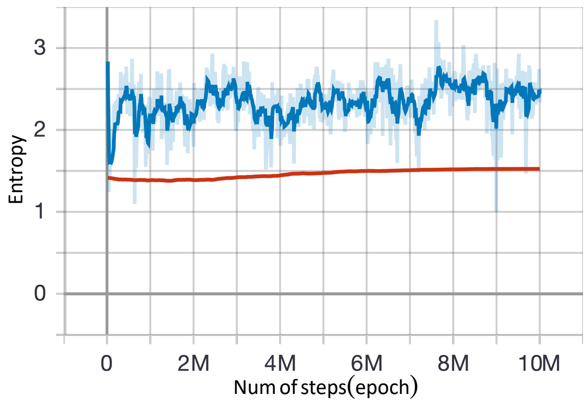

Figure 5: Comparison of transition of the entropy of PPO (red) and SAC (blue)

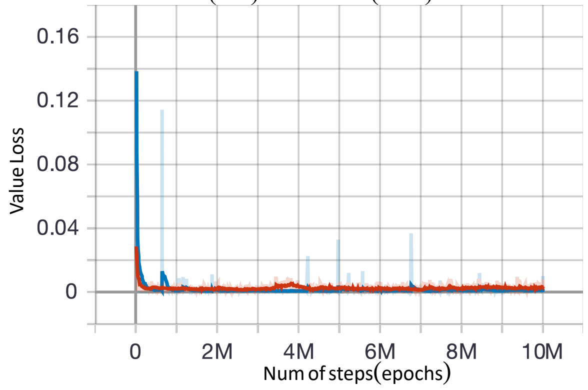

Figure 6: Comparison of transition of the value loss of PPO (red) and SAC (blue)

4. Discussion

This experiment shows the result that PPO has better learning efficiency than SAC in the tasks. Because PPO perform better in almost of situations, PPO are more likely to be more suitable for applying in the autonomous driving area than SAC. From the perspective of what we value, compared to other studies [1], [2], [3], the lap time and car speed are not heavily considered. This is different points from these studies. We can also see the result of stop count with go forward reward and simply turned off the go forward result that shown in Figure 7. Both experiments used PPO method and number of steps is 3.0e5. As shown in Figure 7, at the beginning of learning, go forward reward is a little more effective for reducing the number of stop count. But this does not say that reducing stop count is good for learning because the agent needs to drive and learn status except only when car stopped.

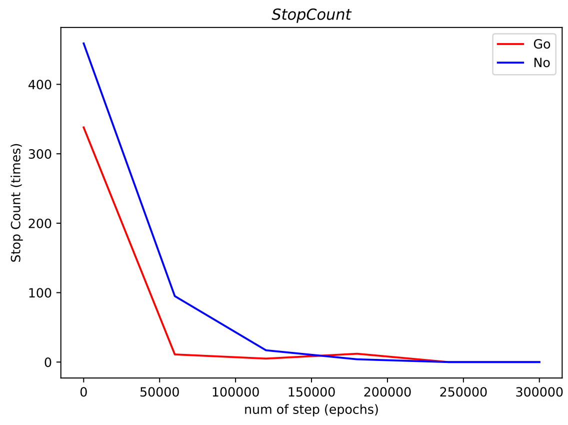

Figure 7: Comparison of Stop Count between that there is “go forward” reward (Go, red) and turned off the “go forward” result (No, blue) at the beginning of learning.

5. Future Work

In the environment part, I plan to add more complex environment for learning, like traffic signal, road sign, and more realistic pedestrians or cars. The road of this environment is simple therefore, we need more complicated road like multi-lane road or twisty road. In the method part, there might be more efficient algorithms or systematic approaches for self-driving. In recent years, there is powerful deep reinforcement learning algorithm like that is called Ape-X [14] or R2D2 [15]. Even without those algorithms, there are Asynchronous Advantage Actor-Critic (A3C) [16], Advantage Actor-Critic (A2C), DQN, and more algorithms. These algorithms are not very new algorithms, but I think I need to learn with these algorithms because of the efficiency and ability for knowing these algorithms are key-issues for self-driving based on ML-Agents. And eventually, we need to transfer from driving in the virtual world to driving in the real world. Therefore, we need framework for transfer learning result of Unity environment to the driving in the application of real world.

References

[1] A. E. Sallab, et al, “Deep reinforcement learning framework for autonomous driving.” *Electronic imaging*, vol 19, pp 70-76, 2019

[2] A. E. Sallab, et al. “End-to-end deep reinforcement learning for lane keeping assist.” *arXiv preprint,* 1612.04340, 2016

[3] S. Sharifzadeh, et al, “Learning to drive using inverse reinforcement learning and deep q-network.” *arXiv preprint,* 1612.03653, 2016

[4] D. Silver, et al, “Mastering the game of Go with deep neural networks and tree search”, *nature*, 529.7587:484, 2016

[5] A. Juliani, et al, “Unity: A general platform for intelligent agents.”, *arXiv preprint*, 1809.02627, 2018

[6] S. John, et al, “Proximal policy optimization algorithms.” *aeXiv preprint,* 1707.06347, 2017

[7] T. Haarnoja, et al, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.” *arXiv preprint* 1801.01290, 2018

[8] Udacity.com, “Become a Self-Driving Car Engineer | Udacity” 2016, [Online]. Available: http://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd013 [Accessed: 6-Nov-2019]

[9] Unity Asset Store, “Standard Assets (for Unity 2017.3)” 2018, [Online], Available: https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-for-unity-2017-3-32351 [Accessed: 24-Nov-2019]

[10] Unity Asset Store, “Lake Race Track” 2018, [Online], Available: https://assetstore.unity.com/packages/3d/environments/roadways/lake-race-track-55908 [Accessed: 15-Nov-2019]

[11] J. Schulman, et al. “Trust region policy optimization.” *International conference on machine learning*, pp 1889-1897, 2015

[12] V. R. Konda, and J. N. Tsitsiklis, “Actor-critic algorithm.” *Advances in neural information processing system.* pp 1008-1014, 2000

[13] T. Sogabe, Introduction of Reinforcement Learning Algorithm, Basics and Applications Starting with the “Average” (強化学習アルゴリズム入門「平均」から始まる基礎と応用) 1st ed, Tokyo, JP, Ohmsha, Ltd, 2019, pp.163-164

[14] D. Horgan, et al. “Distributed prioritized experience replay.” *arXiv preprint*, 1803.00933, 2018
[15] S. Kapturewski, et al. “Recurrent experience replay in distributed reinforcement learning.” *The International Conference on Learning Representations*, 2019.
[16] V. Mnih, et al, “Asynchronous methods for deep reinforcement learning.” *International conference on machine learning*, pp 1928-1937, 2016