DeepRacer Model Training for autonomous vehicles on AWS EC2

Junyao Li
Information Systems
Northeastern University
Boston, USA
li.junya@northeastern.edu

Mohamed Abusharkh
Digital Media Software Engineering
Ferris State University
Grand Rapids, USA
MohamedAbusharkh@ferris.edu

Yong Xu
Businesse Data Analytics
Ferris State University
Grand Rapids, USA
xuy1@ferris.edu

2022 International Telecommunications Conference (ITC-Egypt) | 978-1-6654-8808-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/ITC-Egypt55520.2022.9855675

***Abstract*—Autonomous vehicle (AV) is the future of public transportation to reduce the road congestion and accidents. However, fully self-driving car is still a challenge for carmaker, and they must ensure the maximum driving security to avoid the ethical issue. To develop a mature model to AV, reinforcement learning becomes a solution which can explore many possibilities and choose the best possible action choice facing different road conditions. AWS DeepRacer is a comprehensive platform for researchers to start reinforcement learning for AV. With using DeepRacer Console, all the parameters including action spaces, reward functions and hyper parameters can be edited online and trained remotely. However, the price for training a converged model on DeepRacer Console is quite expensive for beginners. This paper present a DeepRacer simulation process built on EC2 instance which demand no hardware and cost significantly less than regular DeepRacer Console. Based on the default models from AWS, the authors experimentally adjusted hyper parameters and added reward functions to achieve higher speed and smoother driving actions. Even though the computing resources still limited the agent performance and stopped the model from convergence, there are 2-3m/s speed increases when adding low speed penalty and progress penalty on reward function.**

Keywords—autonomous vehicles; reinforcement learning; AWS DeepRacer; AWS EC2 instance; reward function

I. Introduction

A. Autonomous Car and Benefits

Autonomous robots will be one of the universal solutions in the fields that human encounter difficulties. From alleviating the problem of environmental pollution to increasing transportation convenience to delivering foods and packages, autonomous system is going to play a significant role in human life and industry. Advancement in the field of Reinforcement Learning, along with the cloud-computing resources have approaching the dream of fully self-driving vehicles. With increase in land traffic, widespread adoption of autonomous driving systems will improve resource efficiency [1].

At present, autonomous driving on the market still requires human intervention and attention to the driving situation. The ultimate expectation for an autonomous system is all the tasks can be handled by machine itself in all driving environments. The fully autonomous system can take care of all possible situations including accidents. There are many benefits of the fully autonomous system. For instance, if fully self-driving car is on the road, most people will not need to drive the car themselves which means disabled people and elderly group will possess the same movability as a normal human. Moreover, in terms of increase of land traffic, to protect pedestrian security and reducing the road accidents, autonomous system will play an important role in the future [2]. People will be as safe as taking a subway to use autonomous vehicle for the purpose of going out.

B. Reinforcement Learning

Reinforcement Learning is an aspect of machine learning (ML) that agents make decisions by evaluating states through trial-and-error to explore the optimal actions [3]. The core of reinforcement learning is to choose the most appropriate behavior for the environment to be rewarded and avoid punishments during the interactions. For example, the most common similar behavior is learning to play video games. Players can learn different gameplay strategies by completing tasks or clearing levels. Mission failure will allow players to review and modify their action's strategy. And mission success will allow players to take the same strategy in similar situations.

II. AWS DeepRacer

A. AWS DeepRacer Platform

AWS DeepRacer is a comprehensive learning platform for users of all levels to experiment with reinforcement learning and compete the model with others. Reinforcement learning can be used to build autonomous driving applications. It includes the following components: AWS DeepRacer Console, which trains models and evaluates reinforcement learning models in a virtual environment by simulating an autonomous driving environment. AWS DeepRacer Vehicle is a 1/18 scale physical vehicle that is capable of running on a trained AWS DeepRacer model [4].

There is two ways to evaluate the model. One is evaluating on DeepRacer console on the virtual environment. The other one is that downloading the model on the physical car to run on the physical track. In this paper, we will mainly experiment in the first way. We focused on virtual evaluations through adjusting reward functions, action spaces and hyper-parameters to achieve optimizing model in terms of passing rate and cost time.

B. *DeepRacer Architecture*

The Architecture of AWS DeepRacer is comprised of SageMaker, RoboMaker along with the other AWS cloud services. SageMaker is an AWS platform providing machine learning environment for people to train Machine Learning models and RoboMaker is a cloud service that provides developing, testing and deploying robotic solutions. DeepRacer incorporates these platforms to train reinforcement learning models and to create virtual agent and environment using SageMaker and RoboMaker respectively [6]. The trained models will be stored into the cloud storage platform S3 with training and testing logs and other related details.

AWS RoboMaker creates a virtual environment that includes an agent and defined track for racing within the AWS DeepRacer architecture. The agent operates with the policy models that has been trained in SageMaker until reaching the expected total reward.

During the iteration of training episodes, the course is divided into segments of a fixed number of steps for each episode. Experience buffer is cached in Redis in each segment which is an ordered list of the tuples of (state, action, reward, new state) for each step. SageMaker randomly pulls training data from the experience buffer in batches and feeds it to the neural network to update the weights [6]. The revised model is then stored in S3 for SageMaker to use for generating more experiences. This iteration runs until the model is stopped or reach the expected total reward. In the early episode of the training, the experienced buffer is initialized with random actions. Figures 1 illustrate this architecture.

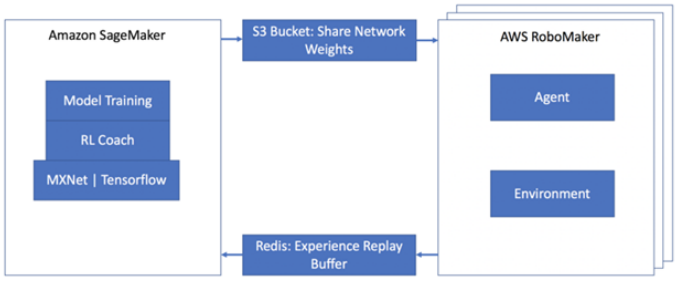

Figure 1: AWS DeepRacer Service Architecture [6]

C. *Hyperparameter*

Hyperparameters are the variables affecting the training model and they can be adjusted as the experiment progresses. There is not a formulation to define an optimal hyperparameters which means the continuous and systematic experimentations are required for obtaining experience to derive a best set of them. There are some terminologies to be introduced first. Learning Rate is the size of the updates to make to the model during the training iterations. For instance, smaller values will slow down the training but brings a more stable process and maybe get stacked in local optimum values. In contrast, larger values will result in bigger update and train faster but no lead to convergence. Entropy is the degree of uncertainty which control the exploration vs exploitation balance [4]. Discount Factors determine how many future steps to be accounted for the current expected rewards. For instance, the agent may consider the next 10 steps when taking an action with a discount factor of 0.9 and consider the next 1000 steps with a discount factor of 0.999. [4]

D. *Action Space*

Action space is a set of all combination of available speeds and steering angles. When the agent interacts with the state, it will choose one of the actions based on the model selection and probability. In the case of DeepRacer car, there are two action spaces include continuous action space and discrete action space [4]. Based on the input of the image captured from sensors on the front of the car, the neural network will choose a proper combination of speed and steering angle. For the discrete action space, it will demand the definition of each action in the action space include the number of actions, speed of each action and steering angles of each action, which result in lots of intensive operation of action setting. On the contrary, the continuous action space can let user to define the range of options the agent picks where its action from. For instance, the AWS DeepRacer car chooses a speed from 0.5 m/s to 4m/s and turn left, right, or go straight by choosing a steering angle from -20 to 20 degrees. [4]

E. *Reward Function*

Reward function is the core part that play an important role in guiding the process of optimization. When the reward function is defined, the model will set up the goal during training to maximize the future reward. In the case of the DeepRacer, the reward function is a python file that takes in a dictionary object of parameters containing present state information and returns a numerical estimation of rewards [4]. Users can edit the python file to return a fixed reward, or a reward calculated by the code with intrinsic parameters. There are some example codes demonstrated on the AWS documentation which can be a good start point. For example, there is a following-center-line rule that farther the car is away from the center line, the less reward is returned.

III. Training on AWS EC2 Instance

There are multiple ways to train the model include training on DeepRacer Console, training on local machine or training on EC2 instance. DeepRacer model is an intensive training iteration which means it will consume a huge amount of computing resources. It will result in a high demand on an expensive hardware or a long-time training on DeepRacer Console. As a result, to run the applications on AWS infrastructure is a better option for training DeepRacer [5] because the servers will be cost-effective virtual servers on Amazon Elastic Compute Cloud.

A. *General Model*

This model uses the default parameters provided by AWS. For reward function, the main reward is from following the center line. If the agent is closer to the center line, higher reward will be assigned to the agent. For hyperparameter and action space, it provides continuous type and a speed range from 0.3 to 1.5 with steering angle from -30 degree to 30 degree. After we ran the model in a few hours on EC2, we can get almost 100 %

passing rate and when you check with the evaluation video, it will show that the agent are keeping a low speed to follow the center line which means it is studying to not running out of the track. The 3-trial evaluations show that the model spends about 39 second to pass each track and we will try to improve its. performance by adjusting the hyperparameter and adding more rules in the reward function.

The training configuration is given in the table, below.

Hyperparameter	Value
Gradient descent batch size	64
Entropy	0.1
Discount factor	0.995
Loss type	huber
Learning rate	0.0003
Number of experience episodes between each policy-updating iteration	20
Number of epochs	10

Figure 2: Hyperparameter for the General Model

The reward picture is given, below.

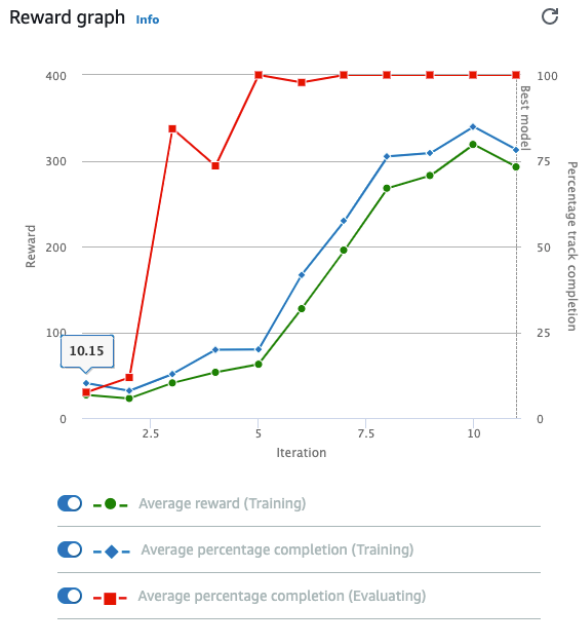

Figure 3: The total reward for General Model

The completion time and trail results are given in the table, below.

The action space configuration is given, below.

Sensor(s)

Camera

Action space type

Continuous

Action space

Speed: [0.5 : 3.5] m/s

Steering angle: [-30 : 30] °

Figure 4: Action space for the General Model

The track of the test run is in the picture given, below.

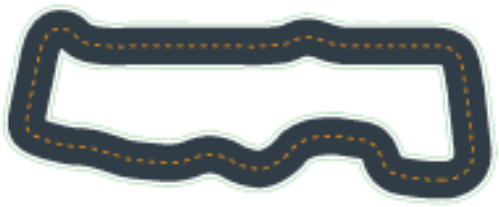

Figure 5: The track path [4]

TABLE I. THE RESULTS FOR THE GENERAL MODEL

Trial	Time	Trial results	Status
1	00:38.195	100%	Lap complete
2	00:39.872	100%	Lap complete
3	00:39.472	100%	Lap complete

B. Hyperparameter Tuning

To increase the speed and the help the car running better to pass the track, we consider improving the discount factor to make the agent that can predict more steps for the future. the value 0.9999 will make the model predicting 1000 steps of agent actions. According to the result, it helps our model learn faster to grow from chaos to simply pass the track but did not improve the agent performance. However, we did not discover any strong infection or relationships between agent performance and hyperparameters.

The hyperparameter and values are given in the table, below.

Hyperparameter	Value
Gradient descent batch size	64
Entropy	0.01
Discount factor	0.9999
Loss type	huber
Learning rate	3e-05
Number of experience episodes between each policy-updating iteration	20
Number of epochs	10

Figure 6: The hyperparameter after tuning

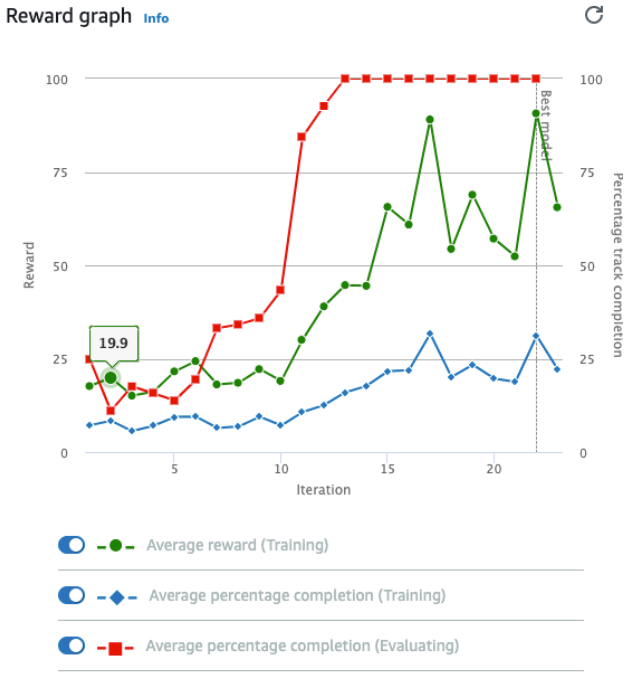

Figure 7: The total reward after tuning hyperparameter

C. *Add the progress penalty and speed penalty in reward function*

To improve the running speed, the effective way is setting reward functions for running faster or passing the track with less time. In this model, we set up a low-speed penalty and a low progress penalty. Both rules will encourage the model to run faster and reduce the reward when the speed is not as expected. We train this model for almost 24 hours with the same track. Obviously, the evaluation showed that the vehicle took less 3 or 4 seconds than our original model (by default), which shows a good direction on improving the model performance.

The reward function is given, below.

```
if (steps % 10) == 0 and params['speed'] < avg_speed:
    reward *= 0.5
else:
    reward *= 2.0                                    (1)
if progress >= (steps / TOTAL_NUM_STEPS) *100:
    reward *= 2.0
else:
    reward *= 0.1                                    (2)
```

The reward picture is given, below.

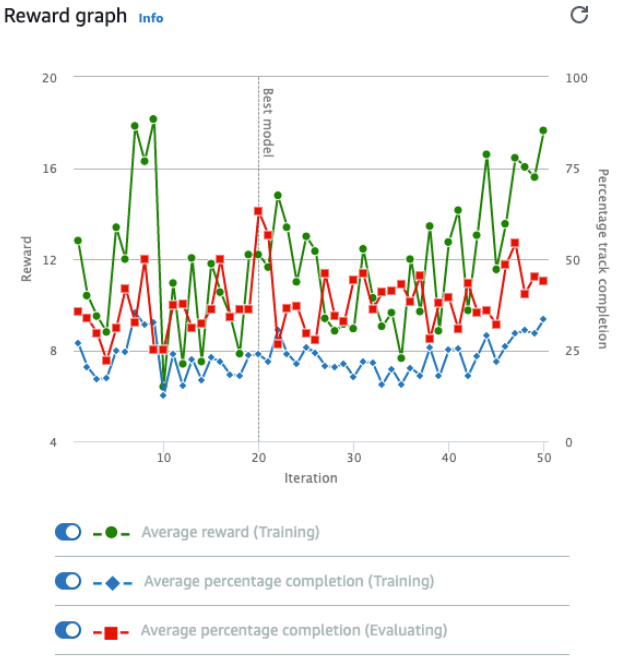

Figure 8: The total reward for editing reward function

The completion time and trail results are given in the table, below.

TABLE II. THE RESULTS FOR EDITING REWARD FUNCTION

Trial	Time	Trial results	Status
1	00:36.324	100%	Lap complete
2	00:36.400	100%	Lap complete
3	00:35.466	100%	Lap complete

IV. CONCLUSIONS

The goal of the reinforcement learning is meaningful. If we can set up the right variable for training, machine will try and come up with an optimal results based on what it explored in training. In terms of implementing reinforcement learning model on autonomous vehicle, there is still a long distance between experiments and reality. One key thing is computing resource which means more GPU and more money. When we pursuit a better model or a higher performance vehicle, the complexity of model will rise rapidly. If we can have sufficient resource, we can continue to provide different variable scaling experiment results for indicating how variable affects reinforcement modeling which will provide some insights for researchers of DeepRacer or reinforcement learning.

REFERENCES

[1] Bagloee, S. A., Tavana, M., Asadi, M., & Oliver, T. (2016). Autonomous vehicles: challenges, opportunities, and future implications for transportation policies. Journal of modern transportation, 24(4), 284-303.

[2] Duarte, F., & Ratti, C. (2018). The impact of autonomous vehicles on cities: A review. Journal of Urban Technology, 25(4), 3-18.

[3] Akalin, N., & Loutfi, A. (2021). Reinforcement learning approaches in social robotics. Sensors, 21(4), 1292.

[4] Deepracer documentation, Accessed: May 10, 2022. [Online]. Available: https://docs.aws.amazon.com/deepracer/latest/developerguide/.

[5] AWS official blog for reducing cost on deepracer by renting EC2, Accessed: May 10, 2022. [Online]. Available: https://aws.amazon.com/cn/blogs/china/use-amazon-ec2-to-further-reduce-the-cost-of-deepracer-training/

[6] Deepracer developerguide, Accessed: May 10, 2022. [Online]. Available: https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-how-it-works-service-architecture.html