Reinforcement Learning based Driving Strategies for Autonomous Driving Systems

Project-II (CS47006) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Harshit Soora

(17CS10014)

Under the supervision of

Prof. Pallab Dasgupta

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Spring Semester, 2020-21

May 6, 2021

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: May 6, 2021 (Harshit Soora)

Place: Kharagpur (17CS10014)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Reinforcement Learning based Driving Strategies for Autonomous Driving Systems**" submitted by **Harshit Soora** (Roll No. 17CS10014) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2020-21.

Date: May 6, 2021

Place: Kharagpur

Prof. Pallab Dasgupta
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Abstract

Name of the student: **Harshit Soora** Roll No: **17CS10014**

Degree for which submitted: **Bachelor of Technology**

Department: **Department of Computer Science and Engineering**

Thesis title: **Reinforcement Learning based Driving Strategies for Autonomous Driving Systems**

Thesis supervisor: **Prof. Pallab Dasgupta**

Month and year of thesis submission: **May 6, 2021**

Autonomous driving is a technologically growing field of study. It is specially challenging to derive driving policies for Autonomous Driving Systems (ADS) due to their complex state and action space. In this work we break down the driving manoeuvre into a set of manoeuvres namely straight drive, left/right turn and left/right lane change. We learn different RL policies corresponding to each manoeuvre and verify our results on driving simulator CARLA.

Acknowledgements

I would like to express my sincere gratitude to my colleague Briti Gangopadhay for her vital support and encouragement regarding the tasks. It have been a privilege to work with her, constant supervision and willingness to solve every hurdle along the way using her expertise.

I am in highly indebted of Prof. Pallab Dasgupta for such flexible deadlines and work hours. It have been great honour to work on such state of the art problem statement where I can proudly say we are the second group to work upon.

I sincerely thank both of them.

Contents

List of Figures

Chapter 1

Introduction

Technological growth has persuaded the automotive industry to take a paradigm shift from assisted driving to fully autonomous driving. However, autonomous driving systems are extremely safety critical which requires them to adhere to high safety standards. One of the major roadblocks towards commercial deployment of autonomous vehicles is development of driving strategies that are formally provable to be safe under all conditions. The goal of this project is to create a hierarchy of program controlled Reinforcement Learning agents with low level RL agents such as Straight Drive, Lane Swap, Right/Left Turn for Autonomous Driving Systems. This overall architecture ensures faster training of sub policies and deterministic triggering of them by a higher level program. Individual maneuvers are implemented as a Reinforcement Learning Task where we define our State, Action, Transaction Function, and Reward Model.

The aim of thesis is to familiarise the readers with different Reinforcement Learning Algorithms. We describe Markov Decision Process , Reward Hypothesis, and explain other background definitions required for understanding the work. Then we describe the Algorithm that is used for this Project, its definition and comparisons. In the final part, we will look into the code and respective graphical plots of Actor-Critic and Reward achieved by our Car which is implemented in simulation engine CARLA. We start by describing some relevant concepts related to Reinforcement Learning.

1.1 Markov Decision Process

MDPs are a classical formalization of sequential decision making, where actions influence not just immediate rewards but also subsequent situation or states and through those future rewards. Markov Property requires that the Present State contains all the information necessary to predict the future, thus we don't need to store earlier outcomes.

1.2 Episodic vs. Continuous Tasks

The agent-enviornment interaction breaks up into episodes.. Each episode begin will begin with a initial start start and send with a reset state to start state. Where in Continuing Task there are no terminal states and interaction goes on continually.

Unified Notation here we club both the type of tasks, where considering episode termination to be into a special absorbing state that transaction only to itself and generates only reward of zero. This is basic pruning of infinite states.

1.3 Bellman Equations

These allows us to relate the current state to the value of future states without waiting to observer all the future rewards. Without Bellman, we have to consider infinite number of possible future. Certain assumptions, the action choices depends only on the current state while the next state and reward depend only on current state and Action

1.3.1 Value Function

These are of two types where in one we are given state, and its value shows the estimate of how good it is for the agent to be in the given state. Others, where we are given a state-action pair, and its value shows the estimate of how good it is for

the agent to perform a given action in this given state.

$$v_\pi(s) = \mathbb{E}_\pi \left[\mathcal{G}|\mathcal{S}_t = s\right]$$

$$v_\pi(s) = \mathbb{E}_\pi \left[\Sigma_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1} | \mathcal{S}_t = s\right] \; for \; all \; s \, \epsilon \, \mathcal{S}$$

1.3.2 Policies and State-Action Value Function

Value Function is defined with respect to particular ways of acting or taking actions one after another based on a certain pattern or architecture called Policies.

It is mapping from state to probabilities of selecting each of possible actions. Reinforcement Learning methods specify how the agent's policy is changed as a result of its experience.

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\mathcal{G}|\mathcal{S}_t = s, \mathcal{A}_t = a\right]$$

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\Sigma_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1} | \mathcal{S}_t = s, \mathcal{A}_t = a\right] \; for \; all \; s \, \epsilon \, \mathcal{S}, \; a \, \epsilon \, \mathcal{A}$$

1.4 Optimal Policy

Given an optimal value function it is actually easy to find an associated optimal policy where an optimal policy is one that maximises the reward obtained in each state.

There always exists one optimal policy and there can be more than one. If there are multiple maximising actions we could define a stochastic optimal policy that chooses between each of them with some probability.

1.5 Stochastic Policies

There are two types of policies, Deterministic Policy where we deteministicly choose the next state based on certain parameter or values or reward or output of a function. But this type of policy have a high amount of getting stuck in some state.

Stochastic Policies will randomly pic on of the successor state with certain probability which then lead to finitely termination of the episode while Deterministic Policy may go into infinite loop (two state pointing to each other).

1.6 Monte Carla Method & Bootstrapping

The idea is we gather a larger number of returns under certain policy and take their average. Each return depends on may random state transitions due to the dynamics of the MDP, which is a lot of randomness to deal with. But the idea of DP combined with it solved more of sub problems resulting in optimal solution.

This idea of using value estimate of successor states to improve our current value estimate is known as Bootstrapping, here we get estimate up to a certain state-level in the episode and set the cumulative reward function of this episode accordingly.

The policy evaluation step changes the value functions less and thus the evaluation step typically terminates quickly. The size of State Space grows exponentially as the number of state variable increases, but this is not an issue of DP but as an inherent complexity of the problem of MDP.

1.7 Reward Hypothesis

There are two type of Learning, one is Supervised Training and other is Reinforcement Learning. The underlying idea among these two is Supervised learning only get taste of the training data and learns the classification. But Reinforcement learning gathers its data itself and learn the feature on it. For this we need a certain Reward model that classifies whether the data is useful for our analysis or not.

Our tasks are, agents should optimise and identify where reward signals come from and design an algorithm to maximise it. Goal and purpose can be thought of as the maximum of the expected value of the cumulative sum of rewards received.

1.7.1 Negative vs Positive Reward

An agent tries to get less negative rewards as it can, thus when we put negative rewards in a state, out agent will try to come out of this state as soon as possible.

An agent tries to get as many positive rewards in its sub state to increase overall episodic reward, thus when we put positive reward in a state, out agent will stay here as long as possible.

1.7.2 Continuous vs Discrete Reward

There isn't any logic that the reward function should be continuous, we did many experiments to check it. This can be explained as simple, there isn't any back propagation or differentiation done with reward function, it only depicts how good an action taken in a certain state is and to encourage or discourage an action in a particular state.

Our function is an increase in start as we need our car to start moving then we slowly decrease it as the speed of the car is required to bring down a threshold such that we can apply the brake to stop it at a Red Light. This is a deterministic program and handled outside our Reinforcement Learning model.

Chapter 2

Reinforcement Learning

In this section we talk about the reinforcement learning practices that are commonly used in Automated Driving and Reward learning.

Let us first go into some quick definitions.

The decision maker is **Agent**, everything outside of it is **Environment**. **State** are defined as the environment parameter of the condition which our Agent is currently in, **Actions** are the possible sets of transactions we can do from current to reach the any other state, **Transaction Function** is defined to enlist the Action required to move from one state to other, **Reward** is cost of this transaction.

2.1 Deep Q Networks

(1) This uses single neural network function approximation to learn the large state and action spaces. Action taken at particular state s is arg-max of Q-values q (state-action value function), w is our initialised weights of the Neural Network. We initialise out target network to these weights and, slowly try to learn and update these over time.

The idea of having Q values for every action we could possibly take given a state and overtime we need to update those Q values such a way that running through a chain of action swill produce a good result.

The following algorithm is used here

Algorithm 1: DQN Experience Replay Memory

Result: A optimal policy π

Initialize Replay memory $\mathcal{R}$ with fixed length ;

Initialize parameter w ;

Initialize parameter v ;

while *episode in range(1, M)* **do**

  Extract the initial state s_0 from environment;

  while *t in range(1, T)* **do**

    Select a random action with probability ϵ else

    Select an action $a_t = \arg\max(s, a, w^t)$;

    Extract next state s_{t+1} and reward r_t from environment ;

    Store tuple (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$;

    Sample random batch from $\mathcal{R}$;

    $y \leftarrow r_t + \gamma max_a \hat{q}(s_{t+1}, a_0, w^t)$;

    $\hat{y} \leftarrow (s_t, a, w^t)$;

    $w^{t+1} \leftarrow w^t$ - $\alpha \Delta \frac{1}{2}(y-)^2$;

    Every c time step, set $v \leftarrow w^t$;

  end

end

Challenges

- The deterministic policy many times lead to infinite episode and getting stuck.
- We can only choose between discrete action space, not continuous actions space.
- Sometimes we are unable to back propagate through a model, this is why the model is not differentiable.

2.2 Deep Deterministic Policy Gradient

Algorithm 2: DDPG Algorithm

Result: A optimal policy π

Initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ ;

Initialize target network $\hat{Q}$ $\hat{\mu}$;

Set weights $\theta^Q \leftarrow \theta^{\hat{Q}}$ and $\theta^\mu \leftarrow \theta^{\hat{\mu}}$;

Initialize Replay memory $\mathcal{R}$ with fixed length ;

while *episode in range(1, M)* **do**

Initialize random process $\mathcal{N}$ for action exploration noise;

Extract the initial state s_0 from environment;

while *t in range(1, T)* **do**

Select a action $a_t = \mu(s|\theta^{mu}) + \mathcal{N}_t$;

Execute action a_t ;

Extract next state s_{t+1} and reward r_t from environment ;

Store transaction (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$;

Sample random mini batch of size N from $\mathcal{R}$;

Take $y_i = \mathrm{r}_i + \gamma\hat{Q}(s_{i+1}, \hat{\mu}(s_{i+1}|\theta^{\hat{\mu}})|\theta^{\hat{Q}})$;

Update critic by minimizing the loss:

$$L = \frac{1}{N}\Sigma_i(y_i - Q(s_i|a_i\theta^Q))^2 \ ;$$

Update the actor policy using the sampled policy gradient:

$$\Delta_{\theta^\mu} J \simeq \frac{1}{N}\Sigma_i \Delta Q(s_i|a_i\theta^Q)|_{s=s_i, a=\mu(s_i)} \Delta_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

Update the target network:

$$\theta^{\hat{\mu}} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\hat{\mu}}$$

$$\theta^{\hat{Q}} \leftarrow \tau\theta^Q + (1-\tau)\theta^{\hat{Q}}$$

end

end

(2) Asynchronous methods can focus updates on more relevant states less often. If the states space is very large, asynchronous methods may still be able to achieve good performance whereas even just one synchronous sweep of the state space may be intractable.

It combines the idea from DPG(Deterministic Policy Gradient) and DQN(Deep Q Network). It uses experience replay and slow learning target networks from DQN and it is based on DPG which operates over continuous Action Space.

2.2.1 Policy Gradients

The idea comes as follow, say we are training or learning something and it don't have anything to do with the background. Now if the background get changes then our Neural Network gives outrageous results far from what were earlier.

Some times complex value functions don't work and policy gradient does the work. We will focuses on learning a policy that optimizes the value without learning the values separately. This ignores all other knowledge(cons).

2.2.2 Actor

The Actor is our parameter policy, our action are chosen by Actor mainly we focus n continuous action space . Here, we can easily extent to high dimension or continuous action space and can learn stochastic policies.

2.2.3 Critic

It predicts if the action is good(positive value) or bad(negative value) given a state and an action pair. It is our value function, which in some sense criticizes the behaviour of the actor in such a way the Actor can more efficiently learn.

We learn both simultaneously or to first make sure that our value function is very accurate before we start using it to improve our policy.

2.2.4 Two Target Networks

Conceptually, this is like saying, "I have an idea of how to play this well, I'm going to try it out for a bit until I find something better.", which overrides the earlier

idea of single target network that is conceptually equivalent to saying, "I'm going to re-learn how to play this entire game after every move". These analogies clearly show that two target model is less computationally expensive, here we have different target network for Actor and Critic, where we predict over Actor and slowly train the Critic. This keeps the disturbance low in the target network where we predict and train at the same time leading to lower convergence.

2.3 Asynchronous Advantage Actor-Critic : A3C

(3) Deep Q Learning uses a single agent and single environment. Unlike it, this algorithm uses multiple agents with each agent having its own network parameters and a copy of the environment.

These agents interact with their respective environments asynchronously, learning with each interaction. Each agent is controlled by a global network. As each agent gains more knowledge it contributes to the total knowledge of the global network.

Simpler techniques are based in either Value-Iteration methods or Policy-Gradient methods. A3C algorithm combines best part if both the methods ie.

- The algorithm predicts both the value function as well as the optimal policy function (probabilistic distribution of the action space).
- The learning agent uses the value of the value function(Critic) to update the optimal policy function (Actor).
- The idea is special in such a way that you don't have to rely on GPU for speed. It will run tasks parallel and in multiple cores of a CPU, and will share some screenshots later.

2.4 DDPG vs DQN vs A3C

When using Neural Networks for Reinforcement Learning, most optimization algorithms assume that the samples are independently and identically distributed.

When the samples are generated from exploring sequentially in an environment this assumption no longer holds.

In DQN, we use replay buffer to address these issues, when the replay buffer is full the older samples were discarded. At each timestamp the actor and critic are updated by sampling a mini batch uniformly from the buffer.

Major challenge of learning in continuous action space is exploration of state and action pairs. We will treat the problem of exploration independently from the learning problem, Noise samples from a noise process are being added.

Work around of replay in A3C is done as we no longer use experience replay but instead use manual agents all exploring the state space simultaneously. Each agent should run in a separate process so that the training occurs in parallel.

Chapter 3

Problem Definition and Experimental Work

Learn to Drive in a Day(4) is the only open source, Deep Deterministic Policy Gradient implementation, which incorporated LIDAR data points as State Space. This implementation is a all in one package where you need to spawn Ego Vehicle and it will return you with the distance traversed via the Auto Drive and simulation time.

From the Literature Analysis, this was the very first paper which developed a working model for DDPG algorithm that makes us the second model to work upon same. Our model took the problem statement differently, we created a hierarchical topology which select individual maneuver such as Straight Drive, Right Turn, or Lane Change, interactively based upon external inputs(if any) else mere justification for the final stop point.

We incorporated the Radar Data points as our state space which created a new problem statement which we will talk about in later section. In a nutshell, our work amazingly introduces safety as we switch among maneuvers via a state automata machine where we can explain the selection of maneuver based upon state space which is really hard to explain in pure Reinforcement Learning module. Let us now look into individual maneuvers in details.

Before we also talk about their reward models for each maneuvers, let us first define a general criteria upon which we created them-

- **Goal Reward** does not really encourage the agent to get to the goal with any sense of urgency.
- **Action-Penalty** runs into serious problem if there is a small probability of getting stuck and never reaching the goal tasks.

3.1 Problem Formulation

The whole problem could be divided into hierarchical model where each maneuver represents Reinforcement Learning problem. A state $\mathcal{S}$ is a tuple consisting of $(\delta_{lon}, \delta_{lat}, p_{ego}, p_{dist}, v_{ego}, v_{veh})$ where $\delta - lon$ and δ_{lat} represents the longitudinal and lateral distance between the ego car and destination, and it is positive when the ego car is behind the target location. p_{ego}, p_{dist}, denote positions of the ego car and target location, and v_{ego} denote the odometer value. The state and actions are continuous.

The action space contains a set of possible maneuvers. Action is a tuple of (throttle, angle, maneuver), For throttle value in action it take between 0.0 to 1.0 as we don't consider breaks and angle value lies 0 in Straight drive maneuver, 0.0 to 1.0 in Right turn maneuver and -1.0 to 1.0 in Lane Change Maneuver. A policy is learnt through model-free learning technique such as Deep Deterministic Policy Gradient Learning since the underlying Markov Decision Process is not known.

3.2 Carla Simulation Environment

All our experimental work was done on simulation environment by Carla. It support development, training, and validation of autonomous driving systems. It is all flexible to use providing all state of the art sensors that could be opt with the ego vehicle to generate training data. In our specific use case, we considered Radar sensor, collision sensor, odometer sensor, lane detector and static prob(barrier).

The Carla Version that we are using is 0.9.8(5), along with it the version of python that is compatible with it is 3.7 or higher. Other dependencies like Tensorflow(2.4.1),

Tensorboard(2.4.1), PyTorch(1.8.1), Keras(2.4.3), Pygame(2.0.1) and Numpy(1.19.5) are used. The code is available at our github repositories (6) (7). You can even have a look at videos guiding though the installation process here (8).

3.3 Straight Drive

In this maneuver, as suggested by the name, agent learns the throttle values. It was fairly a simple maneuver that could be trained with almost 1000 episodes. Even we didn't require lots of exploration to perform this task. The sweet spot for the action range in this maneuver was between 0.3-0.6 throttle value.

One of the major downside or future work needed in this maneuver is to learn about breaking. We didn't put our agent to learn to apply breaks whenever needed as this my lead to side channel problems like agent never reaches goal or even severe it may not start. But this could be done by randomly popping static prop(barrier) along the path of agent and let it learn thought the radar data points. This would again come at expense of training a lot more different scenarios.

3.3.1 Reward Model

```
SECONDS_PER_EPISODE = 80
reward = 0
done = False
if len(self.collision_hist) != 0:
    done = True
    reward = reward - 400
elif kmh<2:
    reward += -1
elif kmh<10:
    reward += float(kmh/10)
elif kmh<25:
    reward += 1 + float(kmh/25)
elif kmh<50:
    reward += 2 - float(kmh/25)
```

```
    else:
        reward += -1

    # Build in function of Carla
    if self.vehicle.is_at_traffic_light() and kmh<20:
        done = True
        reward = reward+200
    elif self.vehicle.is_at_traffic_light() and kmh>20:
        done = True
        reward = reward-100

    if self.episode_start + SECONDS_PER_EPISODE < time.time():
        done = True
        reward = reward-200

    return reward, done
```

Continuous reward does not make sense in Reinforcement Learning as there is no issue of differentiation for which the Reward model need to be continuous. But for the sake of exploration the reward model should cover most of the state space thus we need to give continuous rewards to fuel the exploration till the final goal else we end up optimising till a local optima.

3.3.2 Graph

Actor Loss defines how freely or rewarding an action is in particular state. Critic wherever criticise this behaviour by comparing it with "What action the target model would have taken in this episode", thus making it a Mean Square Error between training model and target model.

Average Reward is the episodic reward average for an aggregate of last 50 episodes. We can even experience that our agent have been going great in the final stages as most of exploration is over and epsilon gradient(represents the exploration probability at each time step) have been dampen by now. Therefore, the behaviour of low rewards in start and better rewards in the end are justified.

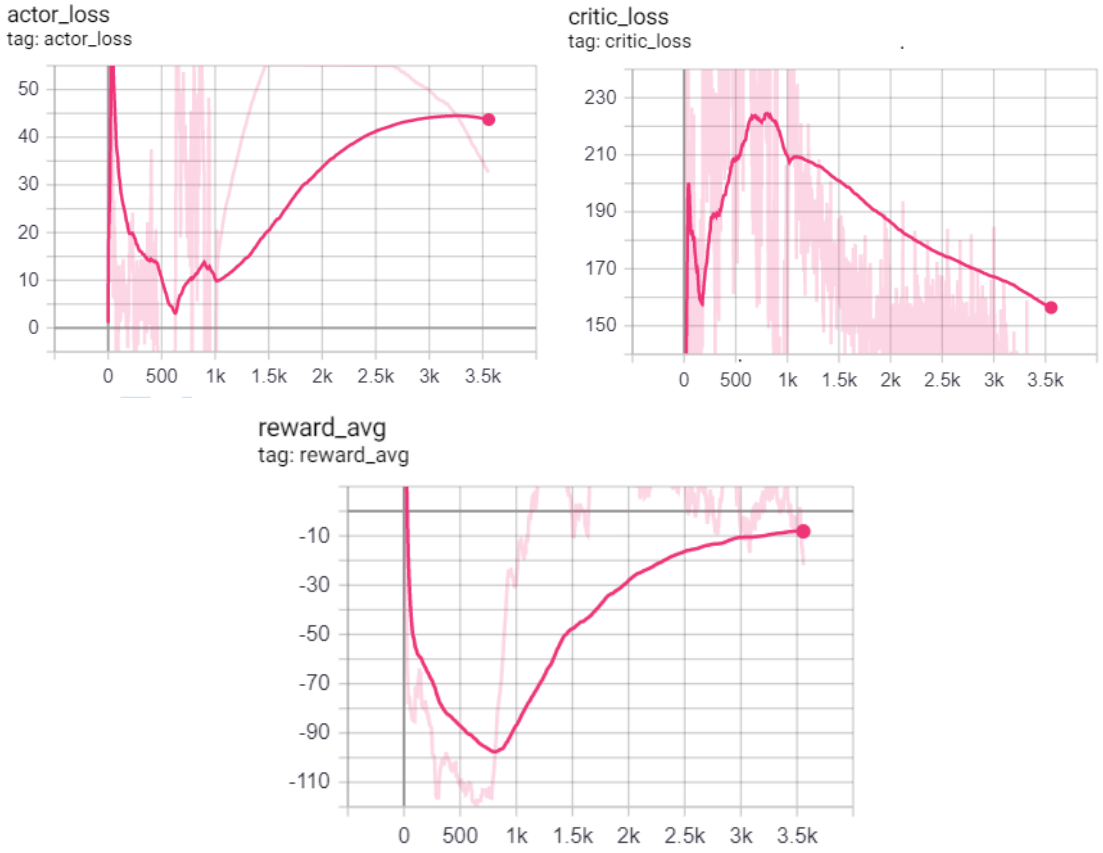

FIGURE 3.1: Straight Drive
X-axis denotes steps taken and Y-axis are individual values based upon graph tag

3.4 Right Turn

This maneuver was created to help agent to learns the angle values. It was a bit complex maneuver so training required around 5000 episodes. We required a fair amount of exploration to perform this task. The sweet spot for the action range in this maneuver was between 0.1-0.4 angle value.

One of the major downside or future work needed in this maneuver is to learn about breaking and negative angle value. We didn't put our agent to learn to apply breaks and take a small amount of negative angle whenever needed. This would again come at expense of training a lot more different scenarios and could arise many side track problem most of which are discussed in the 4.4 Additional Experimental Observations.

3.4.1 Reward Model

```
SECONDS_PER_EPISODE = 30
reward = 0
done = True
# In case of collision terminate the episode
if len(self.collision_hist) != 0:
    reward = reward-200
if self.episode_start + SECONDS_PER_EPISODE < time.time():
    reward = reward-100

# If target is reached and the angle is achieved
if norm_target<=1.5 and diff_angle<=0.5:
    reward = reward+200
else:
    # More close to the target better the reward
    # diff_angle 90----0     norm 15----0
    reward += 0.2*(90-diff_angle)
    done = False
    if diff_angle>45:
            val = 0.6*(-norm_1) + 0.4*(-norm_target)
    else:
            val = (-norm_target)
    reward += val*0.8

return reward, done
```

This was a hard task to formulate as taking odometer values only for state space wasn't going to work. So what we did instead is define a norm distance and target angle that the agent was supposed to achieve with respect to target location. There was a issue of more negative rewards in this maneuver, thus we added more positive rewards based upon how close to target and how align is the agent with target angle.

3.5 Graph

Actor Loss definitely wasn't the best but this was due to no such standard of what angle value should one take in middle of a turn, its mere exploration basis thus adding lower gradient decent factor and more exploration could help to improve it. Critic was criticising the actions correctly but above explanation applies to it too.

Average Reward is the episodic reward average for an aggregate of last 50 episodes. Its behaviour is justified as it started with exploration in the start and end upon achieving goal rewards many time. We can see (if smoothing factor wasn't there in its graph) it crossed the 0 reward level many a times.

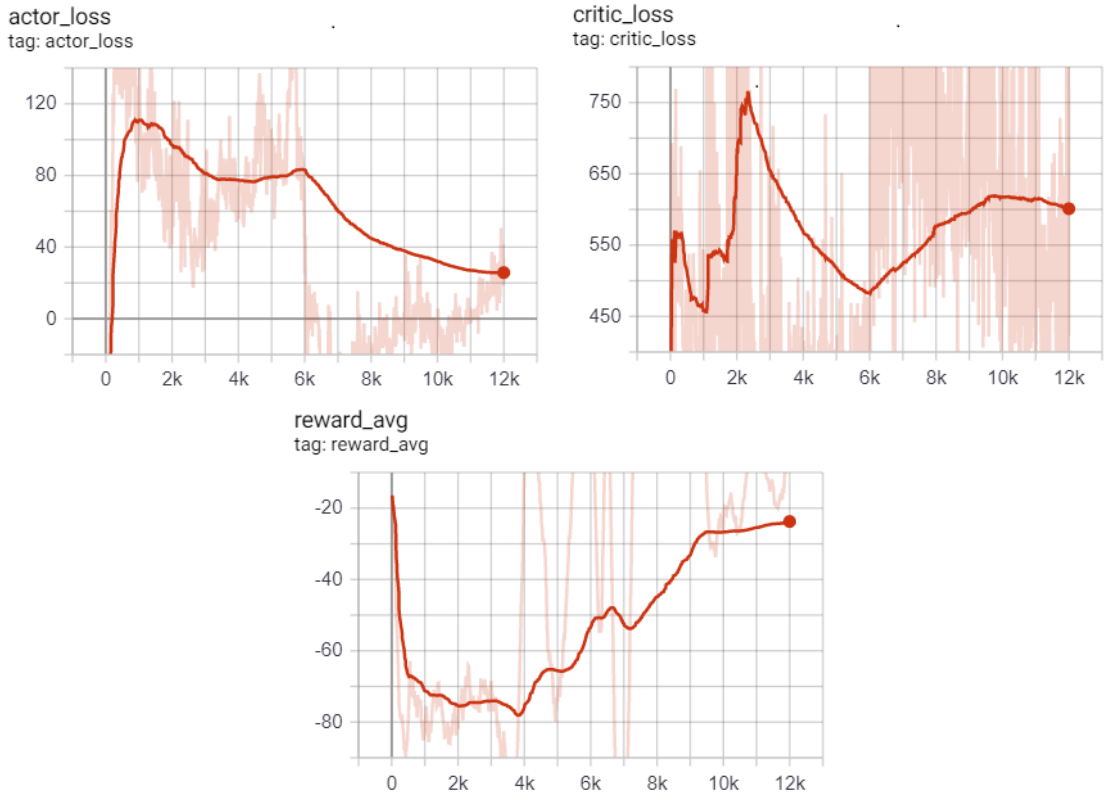

FIGURE 3.2: Right Turn
X-axis denotes steps taken and Y-axis are individual values based upon graph tag

3.6 Lane Change

In this maneuver agent learns the angle values in both positive and negative range. It was the hardest of all and took almost 5000 episodes to train. We required lots of exploration to perform this task and even we decrease the damping factor or epsilon decay to 0.999. We also increase the lower limit of exploration epsilon to 0.2, which means agent always tries to explore new actions with a probability of 20%.

One of the major downside or future work needed in this maneuver is the need of more training episodes. We also lowered the gradient decent α for Actor Model to 0.003 agent. The earlier problem to learn to apply breaks still stays.

3.6.1 Reward Model

```
reward = 0
done = True
if len(self.collision_hist) != 0:
    reward = reward-1000
if norm <= 2:
    reward = reward+500
max_Norm = math.sqrt((110-92)**2 + (134-129.5)**2)    #18.55
reward += (max_Norm - norm)*2

if (self.lane_id_ego!=self.lane_id_target):
    done = False
    if (action[1]<0):
            reward = reward + 30 # More Positive reward
    if (action[1]>0):
            reward = reward - 10 # Less Negative reward
else:
    # Large positive reward if ego car is in target lane
    reward = reward+100
    if (diff_angle > 5):
        done = False
            if(action[1]<0):
                    reward = reward - 30
```

```
            if(action[1]>0):
                reward = reward + 50
        # If the ego car is within range of target angle
        elif (diff_angle<=5):
            reward = reward+500
    return reward, done
```

The state space was similar to Right turn. There was some sweet spotting for the action values as we put direct negative rewards when we don't take action that took us directly towards the target lane. To generalize a bit, the norm value here could be automated and calculated as per need more around that could be read here 4.5.

3.6.2 Graph

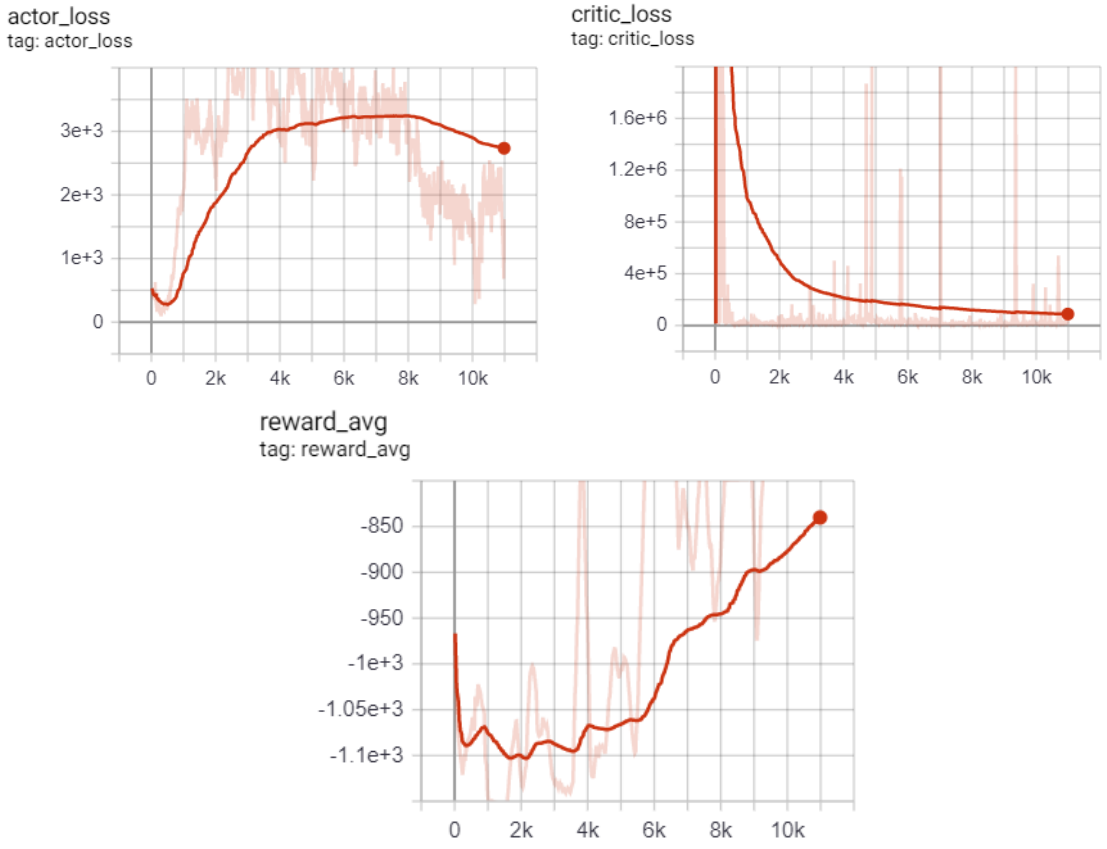

FIGURE 3.3: Lane Change
X-axis denotes steps taken and Y-axis are individual values based upon graph tag

Critic loss was having a really hard time to converge in this maneuver this was all due to the exploration epsilon but larger training episode surely helped. Actor loss did amazingly the freedom to take all action in range -1 to 1 thus we don't need to crop actions.

Forgot to mention the activation function in use,

```
import torch.nn.functional as F

self.l = -1 # lower limit of action
self.r = 1 # upper limit of action
output = F.tanh(self.actor(action))
output = (self.l + self.r)/2 + output*(self.r - self.l)/2
```

Since tanh do gives values between -1 to 1 thus scaling of the final output is not required in this case.

Chapter 4

Additional Experimental Observations

Working upon the problem statement, we developed some side problems most of which are covered in this section. Still some of which are unsolved and hence require more work to arrive at some conclusion thus we put them in the Future Work section.

4.1 Straight Model: Slow Start & Late Stopping

We face this problem in stating where our Car doesn't take any action at all in start state. It keeps on giving throttle value 0.0(action) thus keeping it in same state. Once a negative reward is given here, the car tries to move out of here as soon possible.

Our car tried to keep on getting as many positive sub goal rewards, by staying in particular state(basically speed value) as long as possible. This event made use to increase the final goal reward with respect to intermediate goals.

Also we set a timeout value as car try to keep the speed with best positive reward and diminish the effect of negative reward of not stopping at red light.

4.2 Straight Model : Length Traversed

We tried to plot how much our ego car is willing to traverse in straight drive. This wasn't like the Learn to Drive in a Day(4)'s length which included all three maneuvers at once. Here we find the longest possible route in Carla Simulated environment and added a reward value if the ego car stops at the red light signal.

It is clearly seen in the figure 4, our ego vehicle tried to cross 130 m mark initially which happen as the agent was under exploration and it cross the red light (In case it was green else episode would have ended) but later it tries to stop around the region of 130 m to maximise its reward value, which is where the red light was from the start position.

This complete analysis clearly shows out module is willingly to traverse any length and comparing it to Learn to Drive in a Day(4) doesn't make any sense at all. Both us and them uses different definition of length traversed.

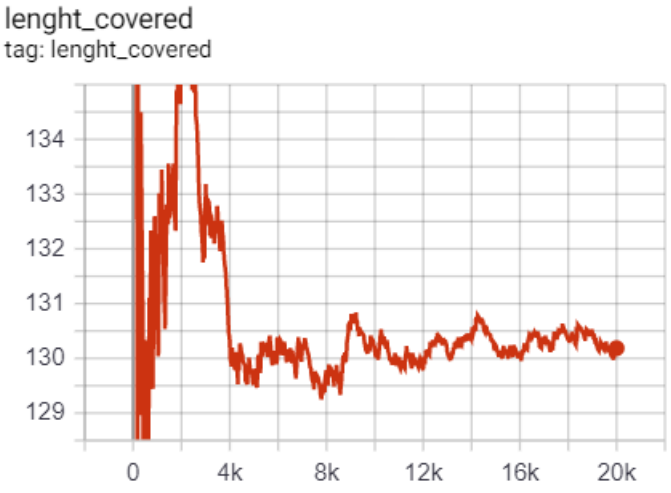

FIGURE 4.1: Straight Drive Length

4.3 Turn Model : Sharp vs Long Arc

Our model went into much trouble when it comes to turning. Since we constantly feed it with a throttle value and there was no measures for applying any breaks. Thus two problems arise

- **Sharp Turn** here an agent is required to take the turn as soon as possible for that the turn value so be high like 0.5-1.0 . But here the problem of roundabout across the starting point occurs since the ego car doesn't have the freedom to take negative turn value (that will lead us to to many other problem)
- **Long Turn** here an agent is required to take a long turn may be due to obstacle on its immediate turn for that turn value should be low like 0.0-0.5 . But here the ego vehicle runs into the opposite lane since throttle is constant so even if it takes high turn value in between it will surely land into the opposite lane or even severe it may collide with divider.

Both of these problem can be solved via Check-pointing where we add middle points along some arc from starting point to the target point. For simplicity of our modules we took this arc as Circle, more about check-pointing will be covered in next section.

4.4 Turn Model : 2 Check-point vs 1 Check-point

Problem in our turning model are

- Ego Vehicle don't have the freedom to take reverse turn like taking a slight left turn to complete the right turning maneuver.
- Ego Vehicle can't slow its throttle value, we set it to constant 0.5
- Ego Vehicle can't even apply break, as that may even lead to severe problem like vehicle may not start.

All these problem do have individual solutions as throttle could be give a range from 0.2-0.6 and reverse turn can be implemented with addition negative rewards in case it get activated when the ego vehicle haven't started thus leading it in other direction. All these will eventually increase the complexity of the module and that even may not converge or achieve some local minima.

These is one simple solution to these all which is check-pointing. In this process we add some local checkpoint along the route of the ego vehicle (this may be automated by taking some arc function passing through both the source and destination). In our case we simplified the arc by taking a simple circle passing by the start point

and target point.

To choose the check points, we cut two points at an angle 30° and other at 60°. Then we added reward based upon norm distance from these two point. But this lead to some other problem where it forces the ego vehicle to take a long arc thus going into other lane or colliding with divider.

Finally, the best plot were achieved with a single check point taken at an angle of 45° along the circular arc.

4.5 Lane Change : Norm Distance

This was a really special task, since we need some more state space parameter in Lane Change maneuver. We introduce Norm distance, but problem here arises we don't have any target checkpoint from which we can take the norm distance. So we created this equation,

$$TargetLocation.X = CurrentLocation.X + \delta$$

$$TargetLocation.Y = CurrentLocation.Y + \delta * \tan\theta$$

where δ is the width of lane, θ is the urgency of lane change which can be between $\frac{\pi}{4}$ to $\frac{\pi}{6}$, taking a sharp lane change could happen with $\frac{\pi}{4}$ and if the ego vehicle is intending to take a long lane change than it may opt $\frac{\pi}{6}$.

FIGURE 4.2: Carla Simulator - Lane Change maneuver

Chapter 5

Future Work

This section will talk mostly about the problem statement that are still open after long experimentation and training. Let us do a walk though of each in details.

5.1 Optimal Radar Points : State Space

This is the most foremost problem that we faced "How many radar points should we incorporate in our training model". Let us first analyse both sides of the coin.

- **Taking less radar point** say 100, result in earlier convergence of the training module but may not result in desired behaviour.
- **Taking more radar point** say 1000, result in desired behaviour but training such a large state space requires many episodic tasks.

This problem statement can be seen in this way "How much complex State space is required such that it results in earlier convergence and required behaviour or high accuracy in testing phase. This is kind of dilemma where you have to choose something in between.

Our specific maneuvers ran on radar data within range of 500-600 points. Straight drive and Turning maneuvers require more state space as it need more refinement of the boundary edges in Turning maneuvers and long vision in Straight drive maneuver, so here we set the radar data points to 600. On the other hand, lane swap

doesn't require either long vision or have to care for boundary edges thus we set its radar data points to 500.

5.2 Check-pointing Problem in Turning Model

Let us first get into the major problem in Turning maneuver. Since we considered norm distance from target location as our state space thus it make the ego car to take a sharp turn. The agent if running along the axis between the target location and the current location then can achieve the maximum reward.

This result in high value of action in range 0.8-1 for turning, since the throttle input is constant therefore the agent not willingly take a roundabout. To solve this problem we introduce Checkpoint along the arc passing via the start location and target location.

Now consider scenario of taking infinite checkpoints along the arc. This will help the ego vehicle to run precisely along that arc. In our experiments, we ran two cases-mentioned earlier as single checkpoint vs double checkpoint.

But the statement still stays as how many check point should be there for the best convergence and behavioral requirement, is it three? may be, someone need to look into that.

5.3 Optimal Reward Model : Inverse RL

Inverse Reinforcement Learning (9) where a trainer demonstrate an example of the desired behaviour and the learner figures out what rewards the trainer must have been out what rewards the trainer must have been maximizing that makes this behaviour optimal.

All the reward model that we presented in different maneuvers are created after long experimentation and trial & error. This bring us the problem of learning reward model from experimental data.

Bibliography

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.

[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep rein. learning," 2016.

[4] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," 2018.

[5] C. Team, "Carla release 0.9.8 [reference link]," https://carla.org/2020/03/09/release-0.9.8/.

[6] H. Soora, "Safe strategies for autonomous driving systems (code distribution) [available on github]," https://github.com/harshit-soora/Carla-DDPG.

[7] B. Gangopadhyay, "Hierarchical-program-triggered-rl (code distribution)," https://github.com/britig/Hierarchical-Program-Triggered-RL.

[8] Sentdex, "Programming autonomous self-driving cars with carla and python [available on youtube]," https://www.youtube.com/watch?v=J1F32aVSYaU&t=1252s&ab_channel=sentdex.

[9] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," pp. 663–670, 2000.