Project: End Car Game using TD3 Deep Reinforcement Learning Algorithm

Submitter:

Monimoy Deb Purkayastha

Email: monimoyd@gmail.com

Aim:

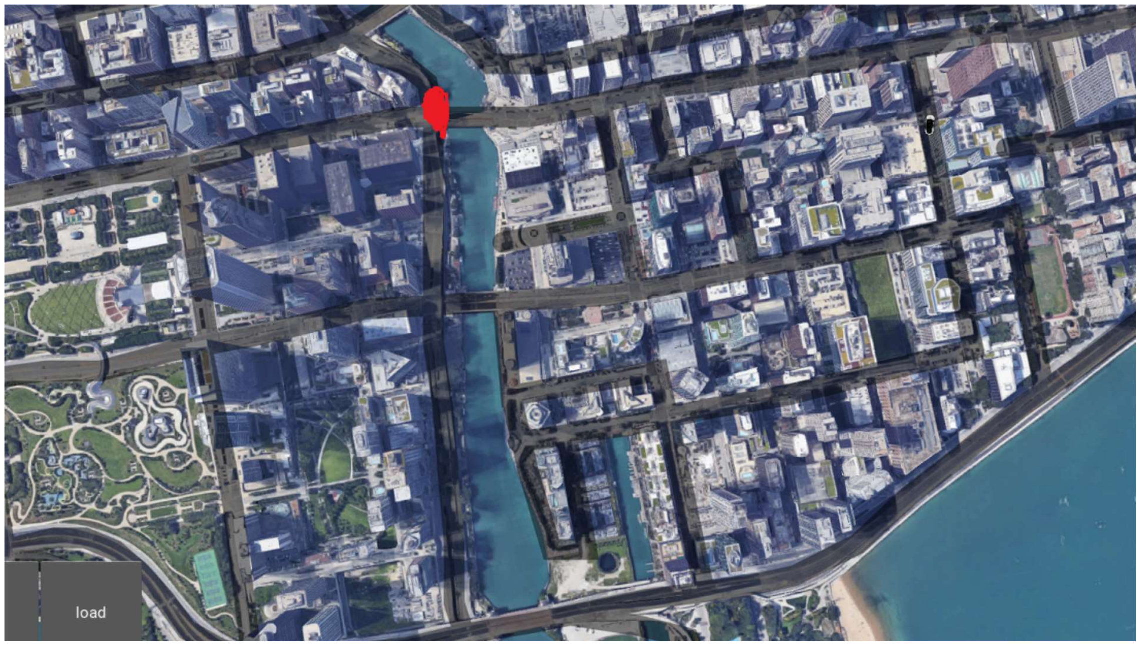
For fulfilment of Extensive Vision AI (EVA) TSAI certification

Acknowledgements:

i. **Rohan Shravan of TSAI** for guiding and mentoring the project
ii. **All the Students of TSAI** for constant support
iii. **The base code for TD3 is taken from** awesome Deep Reinforcement Learning 2.0 Course on Udemy
iv. www.bandicam.com using which I made the video

I. Project Overview

Problem Statement



For this project a map of a city with the roads is provided as an image. Also a car is provided. My task is:

1. To keep the car on the road maximum number of times
2. To reach goal in minimum number of time steps.

Youtube Video Link (The car reaches destination goal from three different source locations):

https://youtu.be/PuYGt-HnJ0s

Github link for the project:

https://github.com/monimoyd/EndGameAssignment

I have used Kivy Environment for this project

https://kivy.org/doc/stable/installation/installation-windows.html with maps provided by TSAI.

In this project I have created a framework for Deep Reinforcement Learning.

Highlights of various activities I have done in the project

- Created a simulated gym environment based on Kivy environment
- Used Twin Delayed Deep Deterministic (TD3) algorithm for training
- Used **LSTM** and Convolutional Neural Network(**CNN**) based DNN model
- For state space, mask is superimposed with a **triangular rotated car** along with a **numbered score** based on what the car did in the step
- Random sampling of actions initially to fill up the replay buffer initially
- Mixture of random actions and policy actions are done based on exploration factor **epsilon** value which is reduced every episode
- After every two training, evaluation is done
- **Metrics (Total Rewards, On Road/Off Road count, Goal Hit Count, Boundary Hit Count)** are calculated in each episode of training as well as evaluation
- **Both Actor and Critic Models after each episode is stored**
- Based on **Analytics on collected Metrics** for evaluation and training episodes I have chosen appropriate model for testing
- Tuned parameters like **learning rate, weight decay** to overcome agent circling (i.e. **Ghumar)** effect

How to install and Run on Windows:

i. First create a new conda environment

```
conda create -n rl python=3.7 anaconda
conda activate rl
```

ii. If you have only CPU:

```
conda install -c pytorch pytorch-cpu
```

If you have GPU make sure that CUDA, CUDNN and drivers are already installed

```
conda install pytorch torchvision cudatoolkit=10.2 -c pytorch
```

iii. Install kivy usin g command below:

```
conda install -c conda-forge kivy
```

iv. Clone the repository

```
git clone https://github.com/monimoyd/EndGameAssignment.git
```

v. Testing

For Testing use the command below:

```
python TD3_test.py
```

[Note: The default code for Full Evaluation (i.e. Test) is in demo mode, if you do not want in demo mode you can change by setting full_eval_demo_mode to False in map.py]

vi. Training

For training use the command below

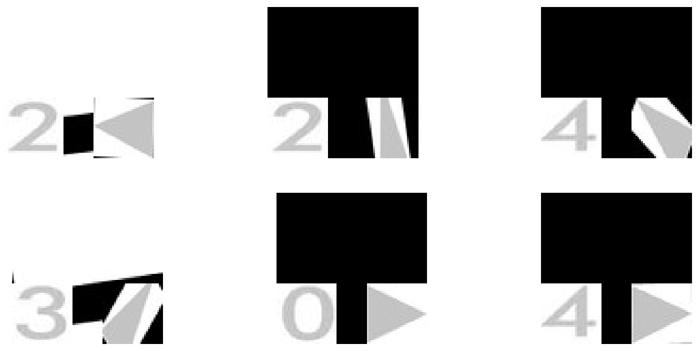
```
python TD3_train.py
```

II. Environment

The agent i.e. car takes from the current state an action based on the policy and environment tells the next state and gives a reward.

a. Observation State Space:

The following a few picture samples show the examples of one part of observation state space



Observation state space consists of a image

- Taking mask value of sand by taking 80x80 numpy array by taking 40 pixel around the car.

- A isosceles triangular shaped car (i.e. only two sides are equal) rotated in the direction the car is moving is superimposed on the image. Isosceles triangle help in asymmetry
- A numbered score representing what the car did in this step is also superimposed. Numbered scores are represented as per below table:

Number	What the car did
0	Car hit the boundary
1	Car has done 360 degree rotation clockwise/counter clockwise
2	Car is off the Road
3	Car is on the Road but distance to destination is increased from the last position
4	Car is on the Road but distance to destination is reduce from the last position
5	Car has reached the Destination goal

This image is processed by CNN and LSTM

In addition the following additional attributes are used in state space:

Attribute	Description
Car Angle	The angle the car is rotated divided by 360. If angle is more than 360 or less than -360, modulus operation is done
Orientation	Angle of orientation of current position of car to the destination goal divided by 360
On Road	Whether car is on road. It has value 1 if car is on the Road, 0 if car is off the Road
Difference of Distance	Difference of distance of the car to the Destination goal from the current position and the last position normalized by dividing by 4

b. Action Space:

Action space of consists of two actions i. Rotation ii. Velocity along x axis

Action	Description
Rotation	Rotation angle along x axis. It can have continuous values -3 to 3
Velocity	Displacement along x axis. It can have value between 0.4 to 2.4

Note: In the Policy network the policy is implemented to give value continuous value between -5 to 5 for both rotation and velocity. Before applying to the car, the value is normalized to the range specified in the table for each of the action

c. **Reward:**

The Rewards are given by environment at each step the agent

Condition	Reward	Comment
Car is off the road	-2.0	
Car is on the Road but distance to Goal is reduced from last position	+5.0	
Car is on the Road but distance to Goal is increased	+2.0	
Car Hit the Boundary	-50	During training Car is moved to a random location after it hits the boundary
Car Hit the Goal	+100	During Testing(i.e. Full Evaluation), once the goal is hit a message is displayed that says your car has reached destination. During training if goal is hit then another goal is set and training continues till maximum timesteps is reached
Car has done 360 rotation (clockwise/anti-clockwise	-50	The car angle is changed by taking modulus of 360 if clockwise or modulus of -360 if anti-clockwise

Each Training Episode has fixed number of 2500 steps. Once episode is over done variable is set to True.

III. Solution approach

In this project I have used Twin Delayed Deep Deterministic (TD3) algorithm (https://arxiv.org/pdf/1706.02275.pdf) for training. TD3 is an off policy algorithm which can be applied for continuous action spaces. T3D concurrently learns a Q-function and a policy.

It uses Actor Critic approach where Actor function specifies action given the current state of the environments. Critic value function specifies a signal (TD Error) to criticize the actions made by the actor.

TD3 uses Actor and Critic principle. TD3 uses two Critic Networks and One Actor network

TD3 uses experience replay where experience tuples (S,A,R,S`) are added to replay buffer and are randomly sampled from the replay buffer so that samples are not correlated.

TD3 algorithm also uses separate target neural network for both Actor and Critic for each of the agent.

There are six neural networks used in T3D

i. Local network for Actor
ii. Target network for Actor
iii. Two networks for Critic
iv. Two Target network for Critic

This algorithm uses time delay for updating Actor after a certain number of iterations. Also, Target Actor and Critic networks are updated periodically after certain number of iterations using Polyak averaging.

Name Twin in the algorithm is used because there are two Critics used.

There are two objectives of the algorithm:

i. Minimize the Critic loss which is sum of mean squared error between Q value of target Critic and the two Critics. Here Gradient descend is used for updating parameters of Critic Network

ii. Maximize the performance of Actor. Here Gradient ascent is used for updating parameters of the Actor network

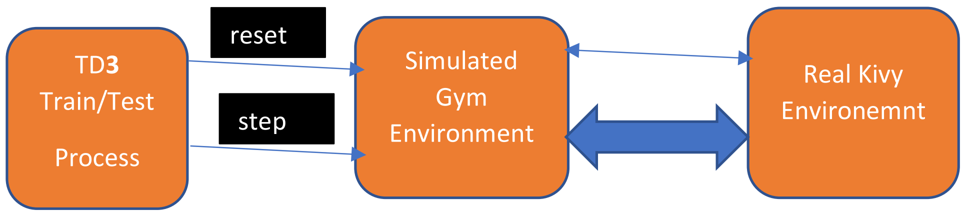
For Details of TD3 algorithm and flows please visit:

https://github.com/monimoyd/P2_S9

IV. Methodology and Solution approach

i. Simulated Gym Environment to encapsulate the Kivy Environment

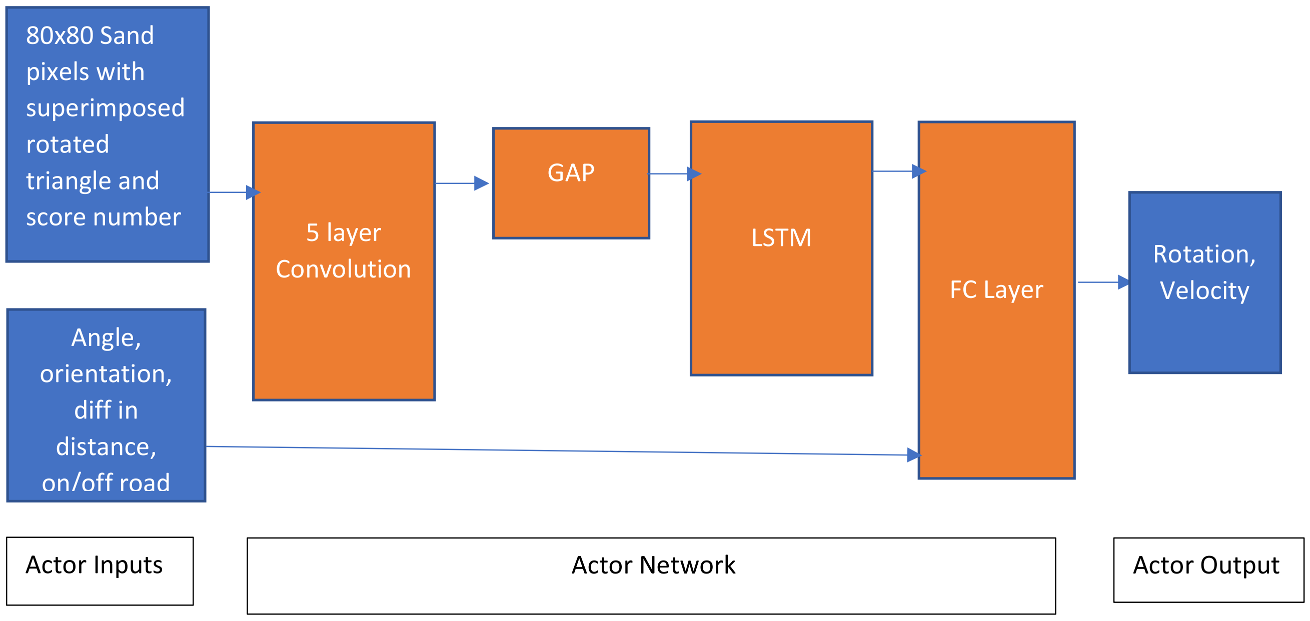
Kivy Environment does not provide methods like reset, step which is very easier to work for any RL project. To solve this I created a simulated Gym Environment which interacts with Kivy based on Multiprocess Queue and Event mechanism provided by Python. The real Kivy environment works on a separate process while TD3 training works on a separate process.



In this TD3 train process first starts and it will start the Kivy Environment. There is simulated gym Environment to which TD3 Train process can call methods like env.reset() to reset the environment and env.step(action) to take a action ands gets next state.

Internally Simulated Gym Environment interacts with Real Kivy Environment using Event and Message Queues.

ii. Actor Network


80x80 Sand pixels with superimposed rotated triangle and score number
Angle, orientation, diff in distance, on/off road
5 layer Convolution
GAP
LSTM
FC Layer
Rotation, Velocity
Actor Inputs
Actor Network
Actor Output

Actor Input:
The Actor Network takes Input as two element tuple

i. 80x80 numpy array representing sand with superimposed isosceles triangle rotated the same direction as car and a number score (1-5)
ii. Second element is a Numpy Array having 4 parameters, these are
    a. Angle of Car
    b. Negative orientation of car to the goal
    c. Difference in distance between current car position to the goal and previous car position and the goal divided by 4
    d. A flag on_road, whose value 1 means car is on road and -1 means car is off

Convolution Layer:

There are 5 convolution layers used to transform the road pixel input. Except last layer, each layer 16 3x3 filters with stride 2 and ReLU Activation is used. Last layer has 16 3x3 filter with stride 1

GAP Layer:

Global average pooling layer is added after 5 convolution layer which transform into 16x1x1

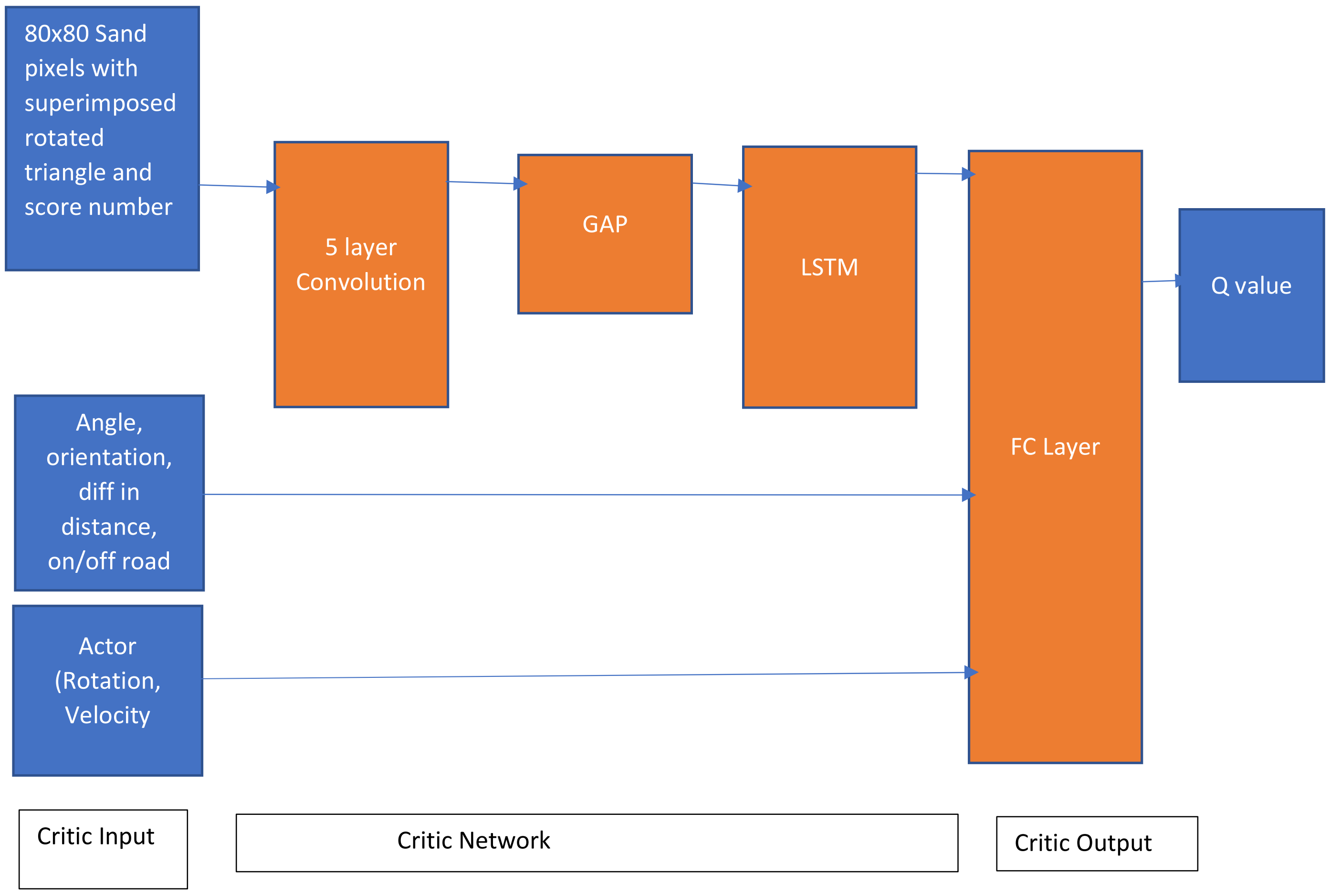
LSTM Layer:

LSTM layer takes the 1 d array and encode into hidden layer of 32 vector tensor

FC Layer:
There are three full connected layers.

- First layer layer takes hidden layer output form LSTM concatenate with 4 extra parameters additional state information (angle, orientation to goal, difference in distance to goal, On road flag value) and convert into 64 1D tensors and applied ReLU activation
- Second layer concatenates first layer output and the and output is 128 1d tensor and applied ReLU activation
- Third layer output form second layer transform to 1 tensor on which tanh is applied and multiplied by max_action to get the actor output

iii. **Critic Network**


80x80 Sand pixels with superimposed rotated triangle and score number
5 layer Convolution
GAP
LSTM
FC Layer
Q value
Angle, orientation, diff in distance, on/off road
Actor (Rotation, Velocity
Critic Input
Critic Network
Critic Output

Critic Input:
The Critic Network takes Input as two element tuple

i. 80x80 numpy array representing sand with superimposed isosceles triangle rotated the same direction as car and a number score (1-5)
ii. Second element is a Numpy Array having 4 parameters, these are
   a. Angle of Car
   b. Orientation of car to the goal
   c. Difference in distance between current car position to the goal and previous car position and the goal divided by 4
   d. A flag on_road, whose value 1 means car is on road and 0 means car is off the Road

i. Actor output (Rotation, Velocity)

Convolution Layer:

There are 5 convolution layers used to transform the road pixel input. Except last layer, each layer 16 3x3 filters with stride 2 and ReLU Activation is used. Last layer has 16 3x3 filter with stride 1

GAP Layer:

Global average pooling layer is added after 5 convolution layer which transform into 16x1x1

LSTM Layer:

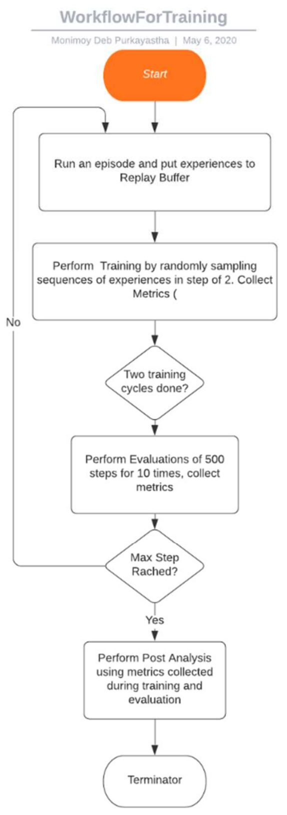
LSTM layer takes the 1 d array and encode into hidden layer of 32 vector tensor

FC Layer:

- First layer layer takes hidden layer output form LSTM concatenate with 4 additional state information (angle, orientation to goal, difference in distance to goal, On road flag value) and actions (Rotation, Velocity) convert into 64 1D tensors and applied ReLU activation
- Second layer concatenates first layer output and the and output is 128 1d tensor and applied ReLU activation
- Third layer output form second layer transform to 1 tensor which is the Q value of Critic

iii. Workflows Used

The following flowchart shows the workflow used.


WorkflowForTraining
Monimoy Deb Purkayastha | May 6, 2020
Start
Run an episode and put experiences to Replay Buffer
Perform Training by randomly sampling sequences of experiences in step of 2. Collect Metrics (
No
Two training cycles done?
Perform Evaluations of 500 steps for 10 times, collect metrics
Max Step Rached?
Yes
Perform Post Analysis using metrics collected during training and evaluation
Terminator

i. Running an episode of 2500 steps
ii. Perform Training and save model after each episode tagged with the episode number so that it can be retrieved for post analysis. Collect metrics
iii. After every two training cycles, Perform Evaluation for 500 steps for 10 times and collect metrics
iv. Run analytics on the metrics and choose the best model

iv. Training

a. Hyper parameters Used during training

I have used Adam optimizer for both Actor and Critic Networks with the following hyper parameters:

Batch Size: 128
Number of time steps : 500000
Steps per episode: 2500
Discount Rate (gamma) : 0.99
Soft update of target parameters (Tau) : 0.005
Initial warmup episodes without learning: 10000 timesteps
Number of learning steps for each environment step : 128
Exploration Noise : 0.1
Policy Noise : 0.2
Actor Learning Rate: 1e-5
Actor Weight Decay: 0.01
Critic Learning Rate: 2e-5

b. Techniques used to Improve Training

To improve the training I have used the following techniques

i. **Warmup**: Initial 10000 timestamps the replay buffer is filled up random policy by choosing action randomly from the action space. This will help better exploration
ii. **LSTM**: As the steps taken by car is a sequence problem LSTM used in the network to improve performance.
iii. **Choosing Sequences of Experiences from Replay Buffer** : As I have used **LSTM**, instead of just taking sample randomly from replay buffer, I have taken sequences of experiences . As I used fixed number of timesteps for each episode, records for consecutive episodes are sequentially stored in replay buffer and it is easy to get records for a particular episode using the following:

- Randomly choose one episode from the list of completed episodes.
- Randomly choose one of timestep from the chosen episode. Ensure that timestep chosen is between 0 and 2500-256
- Instead of taking consecutive experiences, I have chosen experience a gap of two

For example if currently 100 episodes are run and the randomly chosen episode is 10 and starting sequence is 100 then the record ids that are chosen from Replay Buffer for training are:

[100*2500 + 100, 100*2500 + 102, 100*2500 + 104, …100*2500 + 356]

iv. **Exploration factor epsilon**: For episode explorations, I used a epsilon value which is initialized to 0.9 and after each episode epsilon value is reduced by 0.05 until it reaches 0.2. A random number is generated between 0 and 1 if it is less than epsilon value then next action is taken from random policy else action is taken from the DNN based policy network

v. **Gaussian noise** with mean value of 0 and standard deviation (sigma) of 0.1 has been added to explore states.

vi. **Saving Models separately for each episode**: After each episode the model that is used during the episode is saved separately for both actor and Critic. As I have run 200 episodes, so there are 200 instances of both Actor and Critic models are saved. Based on analytics results of metrics collected during the training episode and evaluations done after training 2 episodes, helps to decides which model is doing the best and that is used for deployment.

v. Evaluation:

There are two modes of evaluation:

i. Eval : This is used after two training cycles to evaluate the Policy network by using only Policy learnt so far. Each evaluation consists of 500 timesteps and each is run 10 times
ii. Full Evaluation (Testing): - This mode is used for demo and deployment and only Policy network is used. By default it is set to demo mode but if you do not want in demo mode you can change the variable full_eval_demo_mode to False in map.py

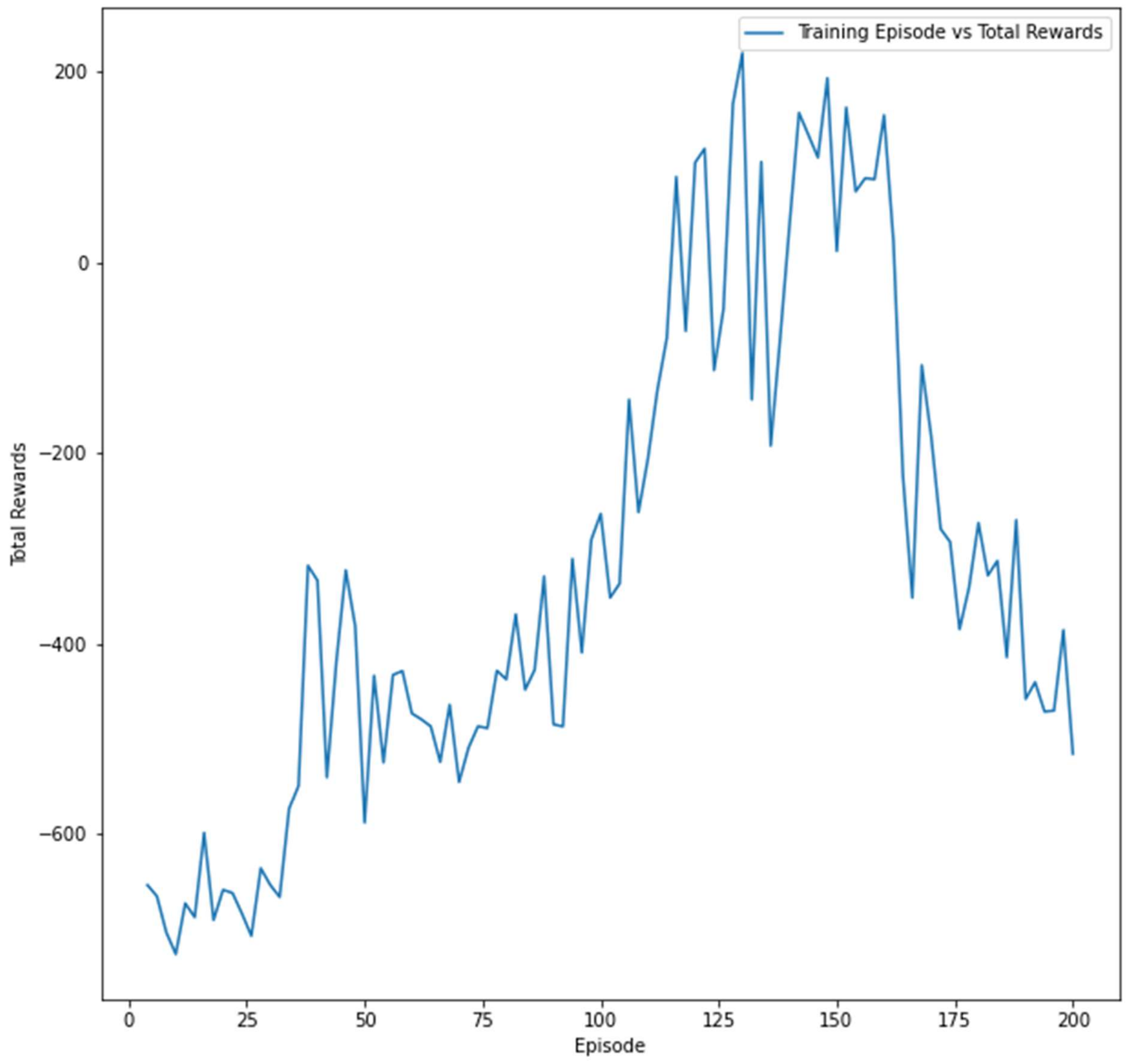
V. Analytics on Metrics Collected

The metrics collected during Training and Evaluation phases after each episode are as below:

i. Total Rewards
ii. On Road Count
iii. Off Road Count
iv. Hit Boundary Count
v. Hit Destination Goal Count

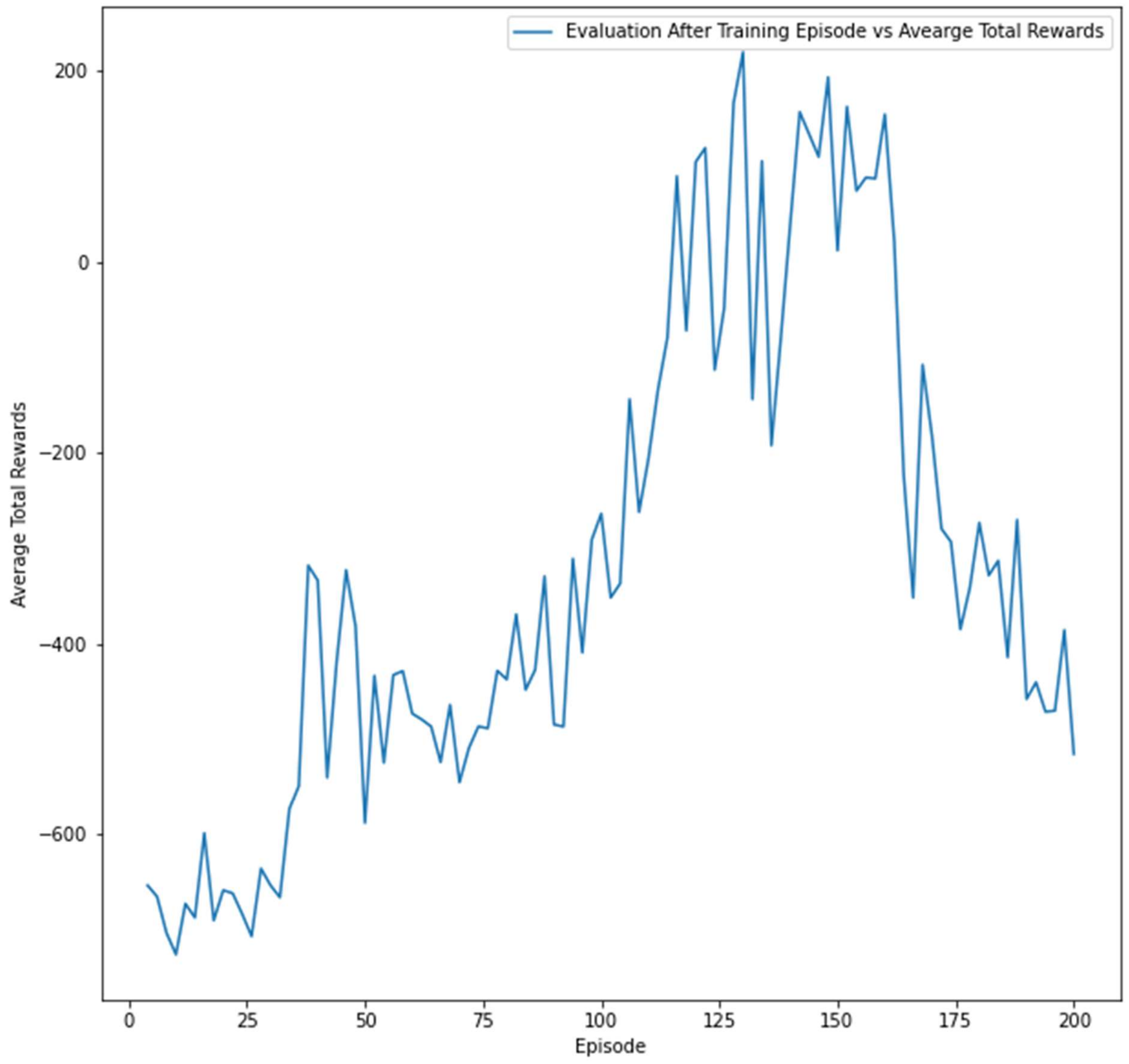
Plots are done on the collected metrics as below:

a. **Plot of Training Episode vs Total Rewards**



Analysis : From this plot, it is evident that total reward increases sequentially initially and reaches peak between 125 and 160 and then start declining

b. **Plot of vs Training Episode (i.e. Evaluations done for 500 steps 10 times after training episode) vs Average Total Rewards**

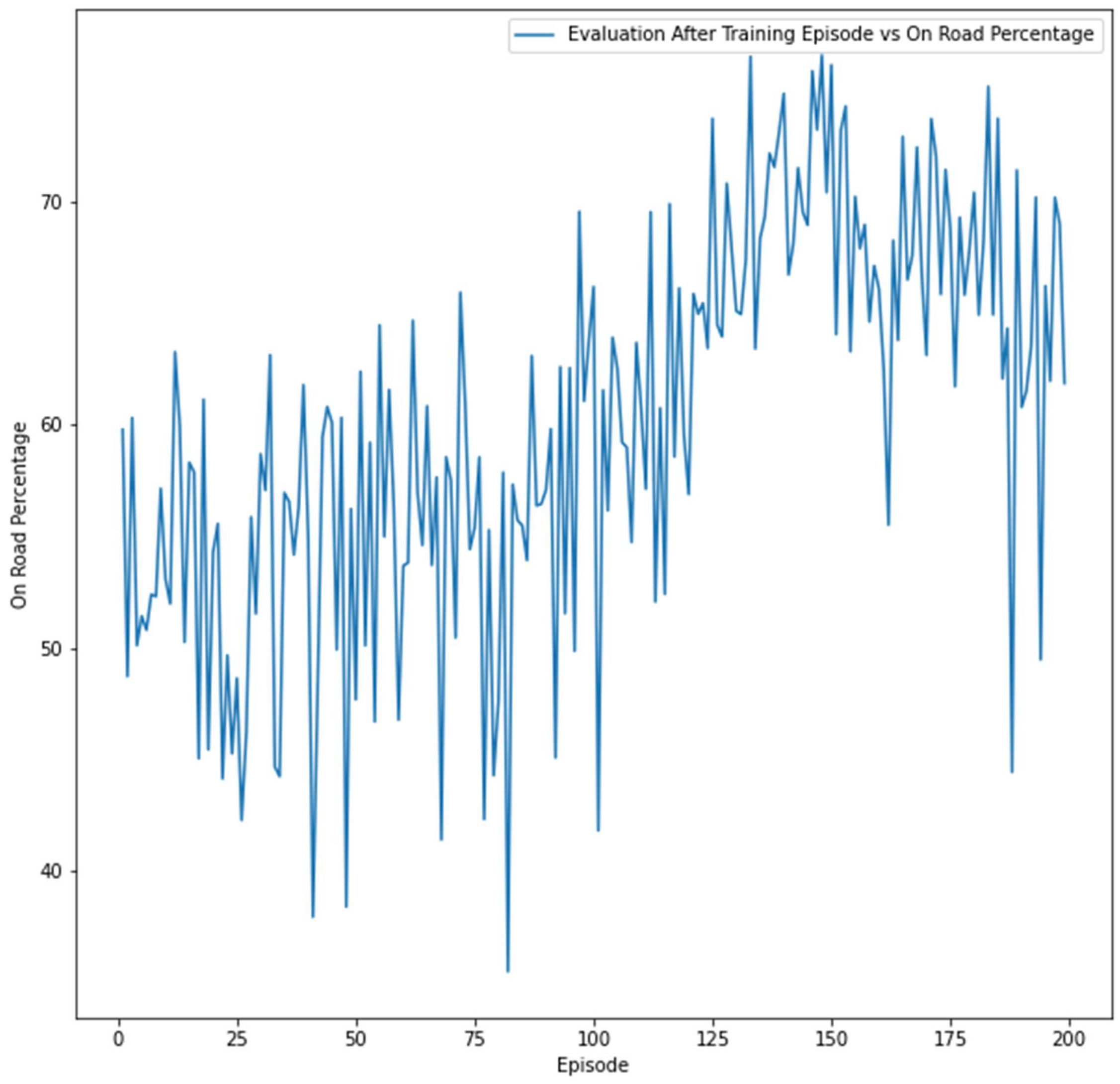


Analysis : From this plot, it is evident that total reward increases sequentially initially and reaches peak between episodes 120 and 165 and then start declining

c. **Plot of Training Episode vs On Road Percentage**

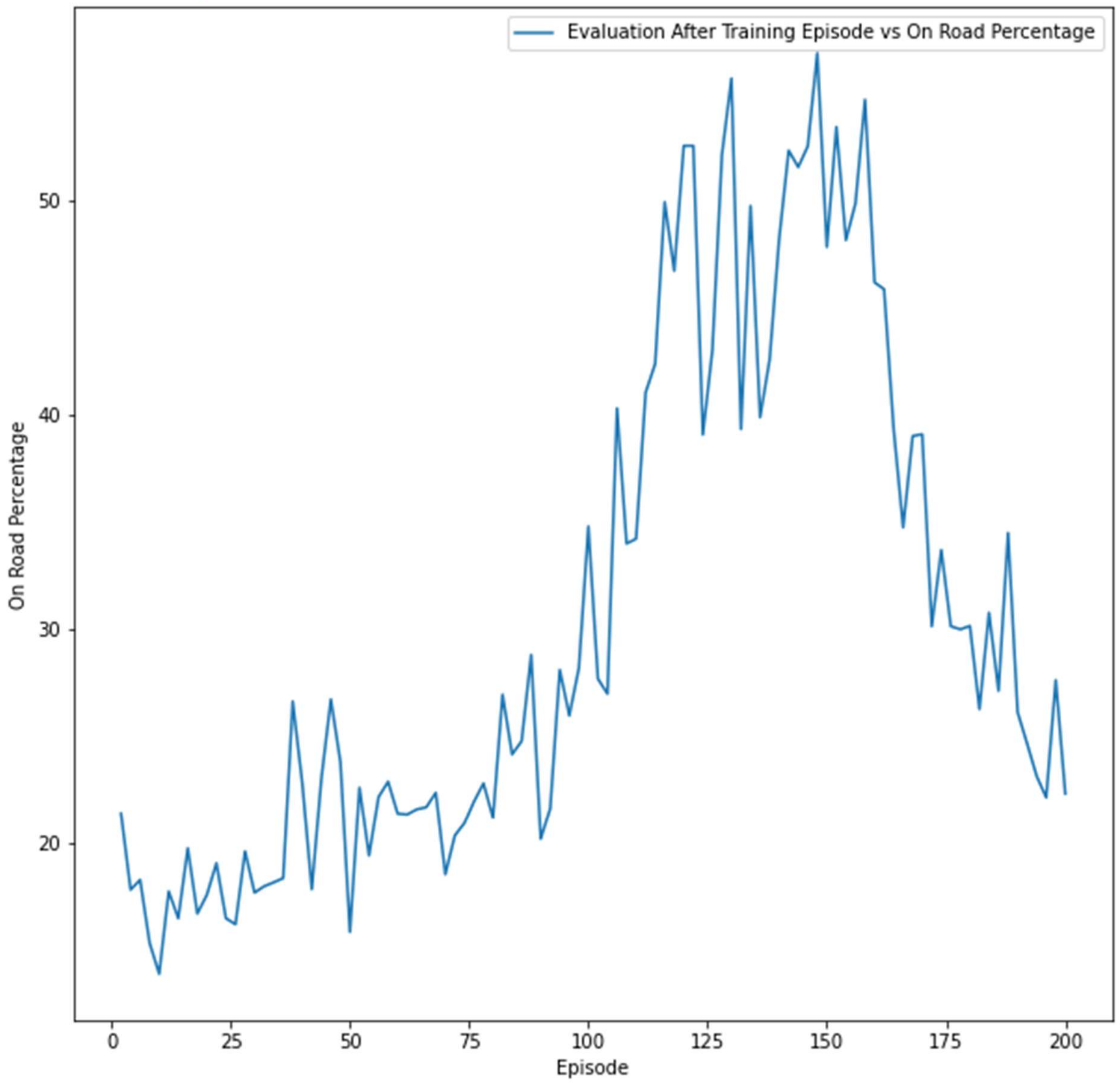
On Road percentage is calculated using formula

On Road Percentage = On Road Count / (On Road Count + Off Road Count) * 100



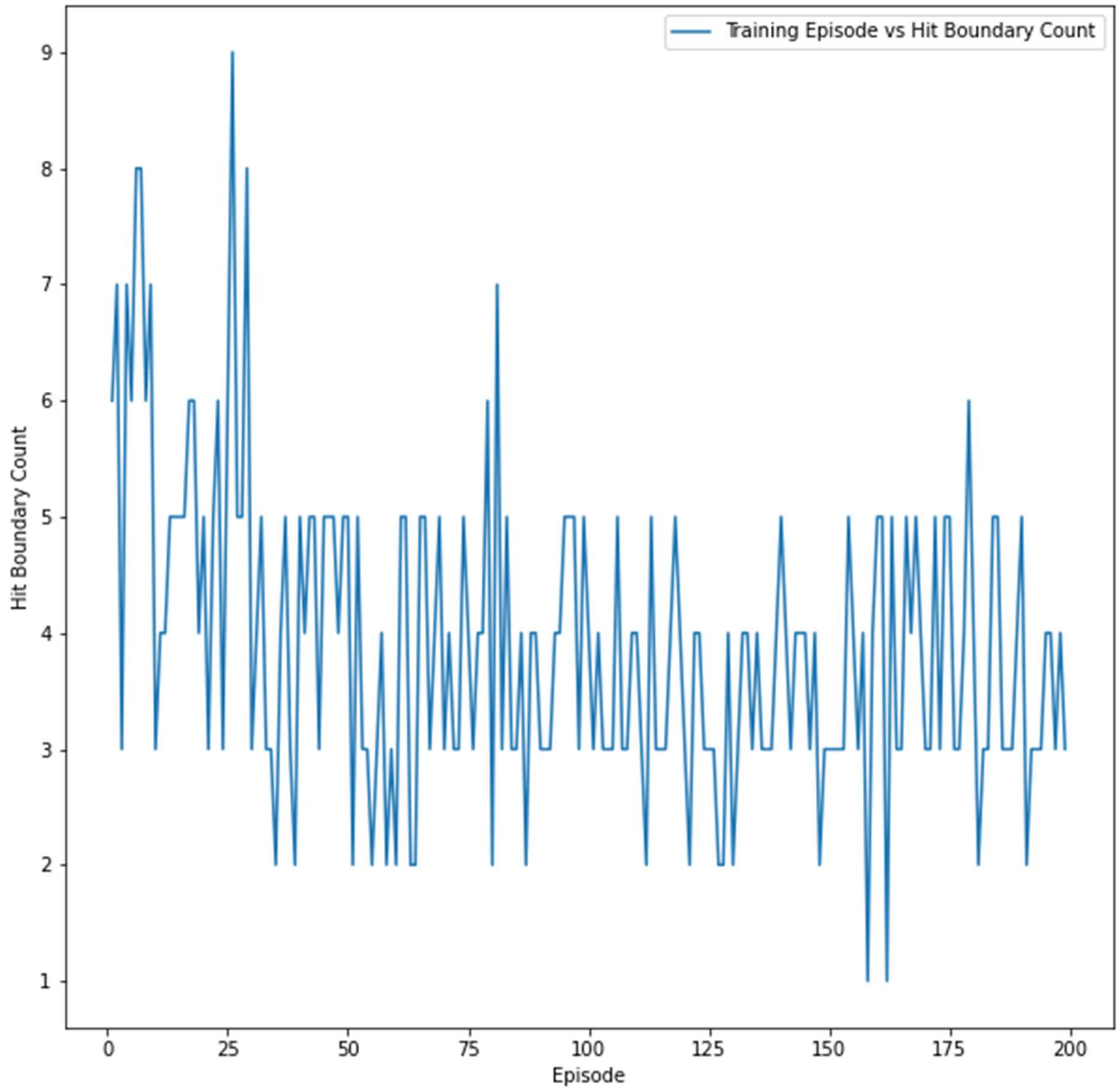
Analysis: From the plot, on road percentage is fluctuating but over episodes even though it is fluctuating the amplitude of fluctuation gets bigger. Around 125 to 160 episodes, the amplitude of fluctuation is in highest range which means car is performing better during these episodes

d. **Plot of Training Episode (i.e. Evaluation of 500 steps for 10 times after training Episode) vs Average On Road Percentage**



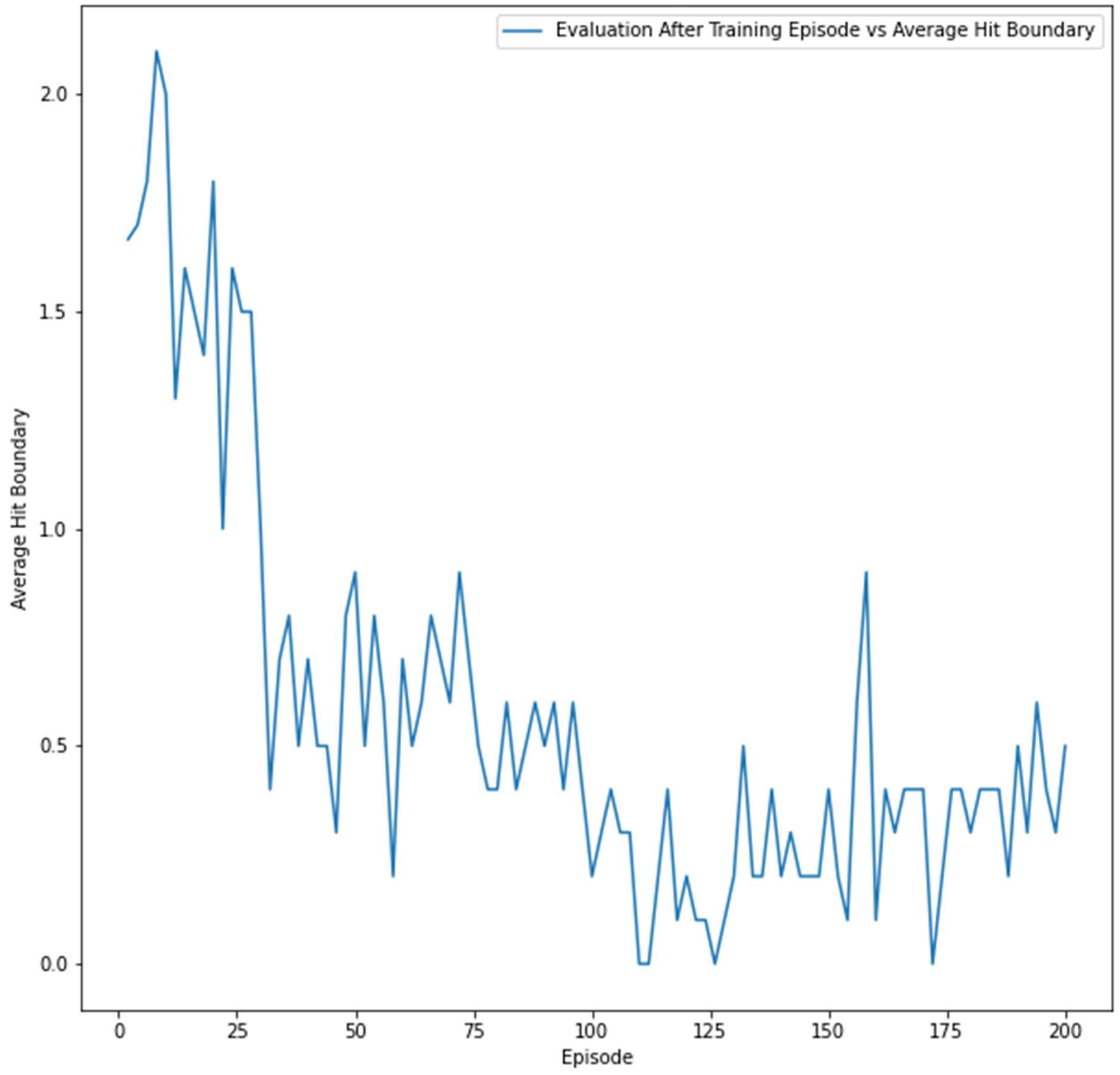
Analysis: From the plot, on road percentage increases over episodes and reaches peak between 125 and 160 and then decreases. I could see less fluctuations compared to training so looks like Policy Network is more stable, but during training as I am still using random samples, so system can be bit unstable

e. Plot of training episode vs On Boundary Hit Count



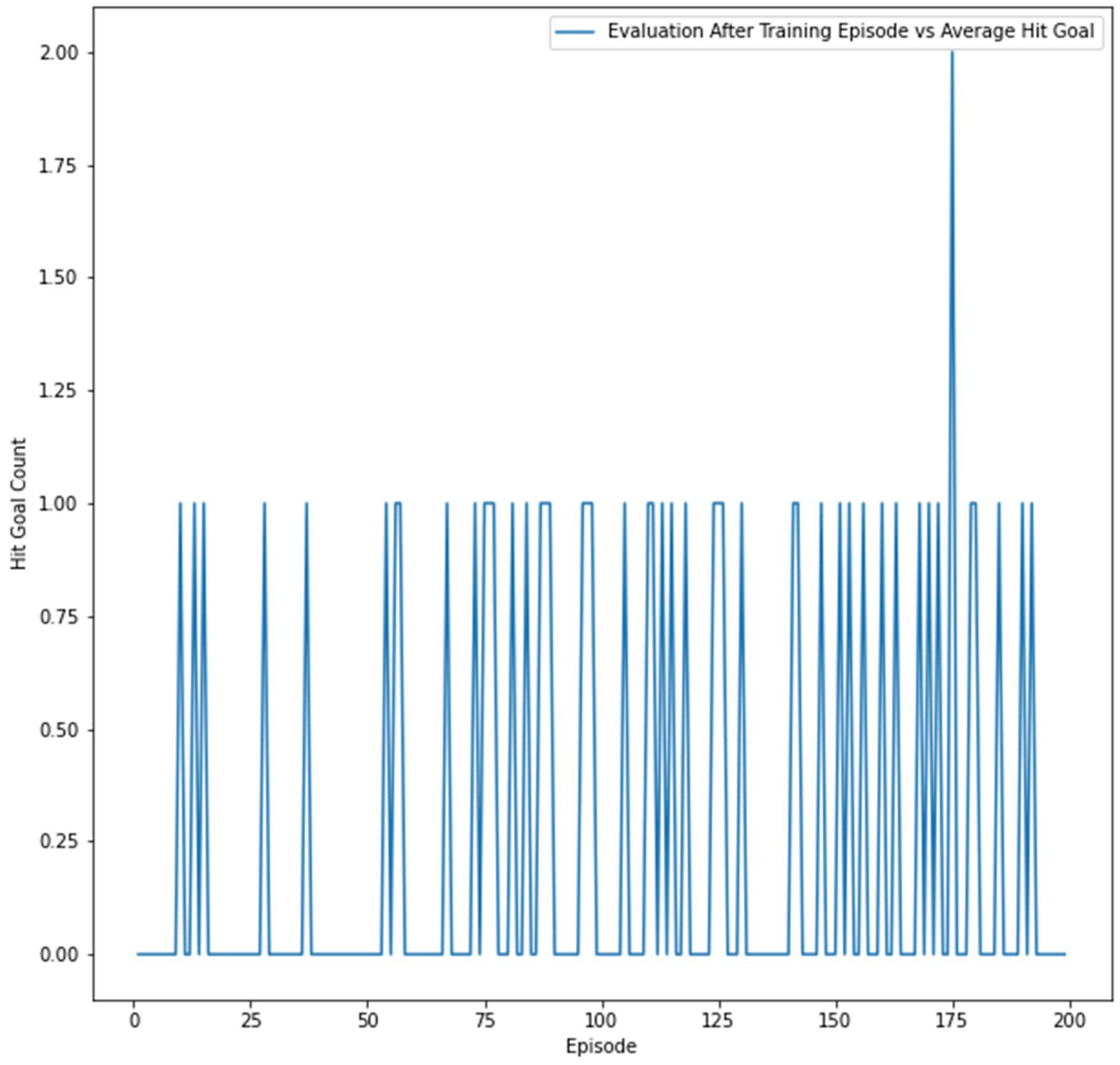
Analysis: From the plot I could see fluctuations in Boundary Hit count but the amplitude of fluctuations is more during the initial episodes but less during later episodes (although I could see some spikes as well).

f. **Plot of Training Episode (i.e. Evaluation of 500 steps for 10 times after training Episode) vs Average Hit Boundary**



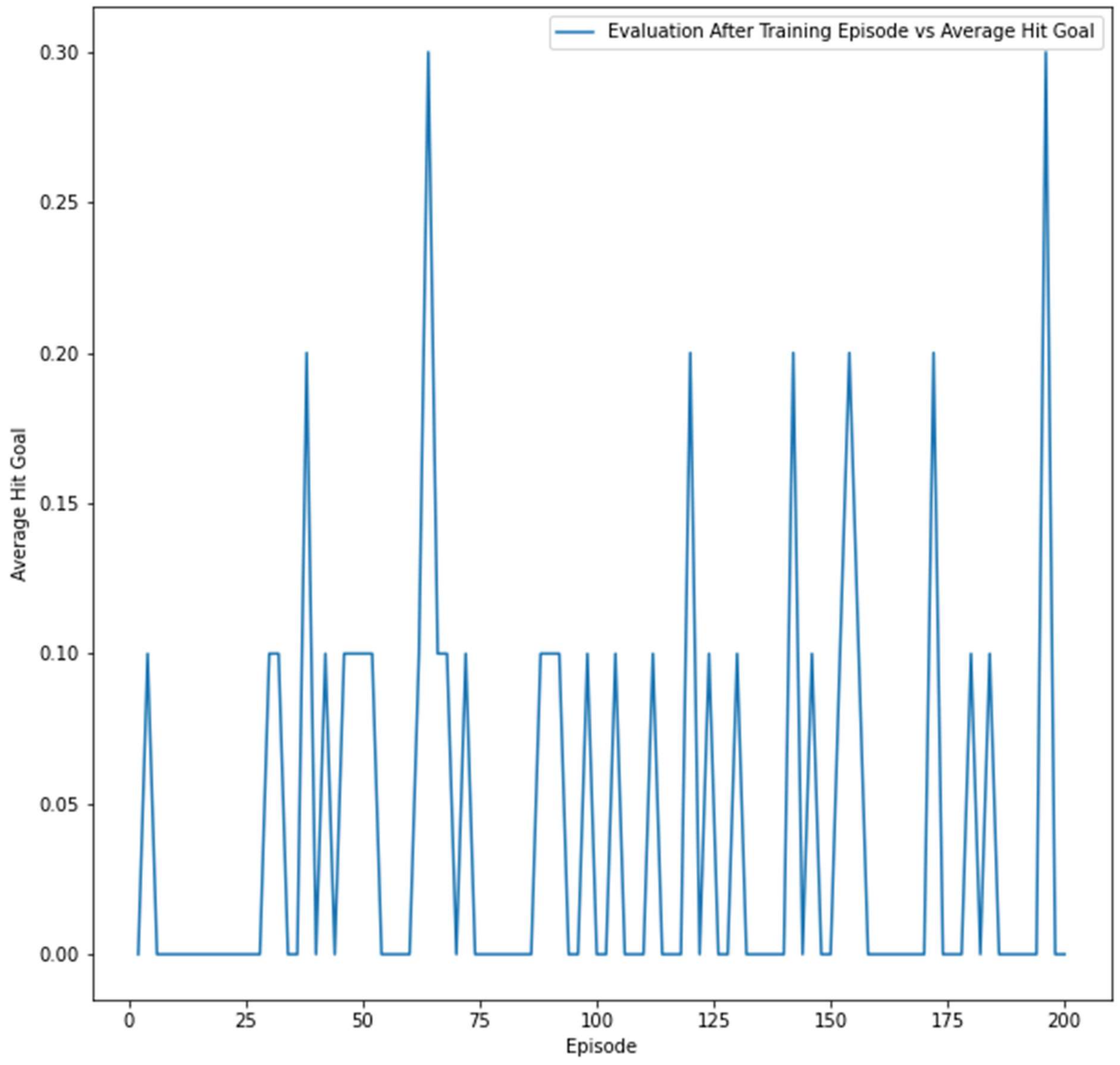
Analysis: From the plot, it is evident that average of boundary hit count is decreasing over episodes and lowest in the range 110 to 150 episodes after that there is a small spike but it stabilizes mostly in later episodes

g. Plot of Training Episode vs Goal Hit Count



Analysis: From the plot, it is evident that Goal Hit Count is 1 in some episodes other cases it is 0. Only episode around 175 has Goal Count 2

h. **Plot of Training Episode (i.e. Evaluation of 500 steps for 10 times after training Episode) vs Average Boundary Hit Count**



Analysis: From the plot, it is evident that Average Goal Hit Count is 0 in most episodes, 0.1 in some episodes. I could observe spike of 0.2 near episodes 35, 125, 145, 155, 175. I could see spike of 0.3 in two cases near 65 and near 200.

VI. Selection of Best Model

From the analysis it is evident that best performance of car is between 120 and 175. Next I checked the actual metrics and found that performance specially rewards are very good in episodes 116, 120, 122, 128, 130, 134, 140, 142, 144, 147, 148, 150, 151, 152, 154, 156, 158, 160

Next I loaded Actor and Critic model corresponding to the episode and do a full evaluation of 2500 steps for 5 times only using Policy Network and calculate average reward earned and also observe car is performing and note that in a table like below:

Episode	Average Reward	Comment
116	-16.50	Lot of times misses goal by whisker but good moves
120	+157	Average Reward is best, lot of good moves but little unstable
122	-436.40	Hit the goals twice but Average Reward score is very poor
128	-254	Average Reward is low
130	-290	Average Reward is low
134	-320	Average Reward is low
140	-387	Average Reward is low
142	-192	Average Reward is low
144	-263	Average Reward is low
147	-29.70	Very smooth, worth exploring, Average Reward is good
148	-77.20	Very smooth, worth exploring, Average Reward is good
150	-58.40	Very smooth, worth exploring, Average Reward is good
151	-251	Average Reward is Low
152	-275	Bit Shaky but worth exploring
154	-392.80	Average Reward is very low
156	-473.40	Average Reward is very low
158	-360.20	Average Reward is very low
160	-625.50	Average Reward is very low
199	-2105	Worst. Has Ghoomar effect

From all the observations I found that though episode 120 has highest reward but car moves are not very smooth. So I have decided to choose model from episode **147** which has good average reward but car moves are smooth. I used this mode l demo and any further fine tuning. I have stored all the other model weights as well so that it can used for future analysis

VII. Issues Faced During Training and how I handled

a. **Car hitting Boundary and remain there**: For handling this scenario, once car is within 0 pixels boundary, I am giving heavy penalty of -50 and moving the car to a random position to start with. As I am using fixes number of steps for each episode, I continue after hitting boundary
b. **Training is very slow**: Initially I tried CPU version of pytorch, I found it was taking very long time to train. So, I installed latest version of CUDA, CUDNN, Nvidia drivers and installed GPU version of pytorch, after that I saw significant improvement in performance
c. **Car is circling same place (Ghumar Effect):** For this is one of common issue that I also encountered. When it happens the action value of rotation, velocity become either near 5 and -5 and it stays there, it is like car is performing Ghumar dance (https://en.wikipedia.org/wiki/Ghoomar). I think this can happen when there are not enough random samples to explore and during training samples are taken from replay buffer which are fed Policy Network and Policy Network has not learnt yet. Here are a things I tried to overcome Ghumar Effect
    a. I have taken 10000 random experiences initially and even after that also I used exploration factor epsilon initialized to 0.9 and reduce it by 0.05 very episode till it reaches 0.2. This ensures replay buffer always have enough random experiences in buffer
    b. Whenever 360 degree rotation happens, clockwise or anti clockwise, I penalize the car agent by giving reward of -50. For checking if car has rotated 360 I check if the car.angle value is greater than 360 or less than 360 and then take modulus value of 360 (if clockwise) or modulus value of -360 (if anti clockwise) to set the new car angle so that I do not miss it next time
    c. The default learning of Adam optimizer is 0.001. I changed the learning to 1e-5 for actor and 2e-5 for critic. Also I used weight decay (L2 regularization) for Actor

VIII. Future Improvements

The There are lot of scopes for improvements

a. **Taking bigger patch from sand**: I have chosen 80x80 sand patch superimposed with isosceles triangle and numbered rating for state. A bigger dimension of say 160x160 can be tried, which can give better performance
b. **Multiple Goals**: Although during training I have used two goals, after hitting the first goal, the goal is changed to second goal. But in reality I saw goal is hit less number of times, so second goal has got very less exploration. Also, number of goals can be increased. To give equal weightage to all the chosen goals, one of the goals can be chosen randomly during episode
c. **Multiple cars**: In real world scenario, there will be multiple cars plying on the Road. Our agent should learn to handle this and should avoid collisions. For this MADDPG (Multiple Agent Deep Deterministic Algorithm) could be used
d. **Kalman Filter:** From Wikipedia **"**In statistics and control theory, Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe". Even though I have used LSTM but Kalman filter could give better results

IX. Code Structure:

File Name	Description
TD3_train.py	This is the main file for training using TD3
TD3_test.py	This is the main file for testing using TD3
map.py	Kivy Environment file
SimulatedGymEnvironmentFromKivyCar.py	This file provides API to simulate gym environment from real kivy environment using Events and Queues
car.kv	Kivy properties file

images folder	Includes all the images of car, triangular car etc.
generate_plots.ipynb	Jupyter Notebook to generate plots from the collected metrics

X. Logs for Debugging and Troubleshooting

File Name	Description
results/train_epoch_reward.csv	Rewards earned during training episode. Fields are Training Episode, Total Reward
results/eval_epoch_reward.csv	Rewards earned during training episode after train Training Episode. The fields are Training Episode, Evaluation Run No, Total Rewards
results/train_on_road_stats.csv	Contains on road statistics on training episode. The fields are: Training episode number, On road count, Off road count, Hit Boundary Count, Hit Goal Count
results/eval_on_road_stats.csv	Contains on road statistics during evaluation after a training episode. The fields are: Training episode number, Evaluation Run Number, On road count, Off road count, Hit Boundary Count, Hit Goal Count
results/ train_traversal_log.txt	Log file which captures the information about each step during training
results/eval_traversal_log.txt	Log file which captures the information about each step during evaluation
results/full_eval_traversal_log.txt	Log file which captures the information about each step during full evaluation
results/policy_action_file.txt	This file contains the policy values (rotation, velocity) calculated by Policy Network
results/full_eval_epoch_reward.csv	Rewards earned during each epoch for Full Evaluation mode
results/full_eval_on_road_stats.csv	On Road stats for Full Evaluation mode

sand_images/	After every 500 timesteps ,the sand image with superimposed triangle and the rating number is generated and stored

XI. Conclusion

In this project I have used TD3 algorithm to train a agent car in Kivy environment. I have faced lot of technical challenges and resolved one by one. One of the greatest learning I got from the project is I need to be "patient and try try again and never give up " to be successful in Deep Learning Projects and which will be helpful in future as well