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

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. Our task is:

1. To keep the car on the Road maximum number of times
2. To reach goal in minimum number of time steps. Once one goal is achieved the car moves for the next goal and vice versa

I have used Kivy Environment for this project

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

The observation space consists of tuple of three elements:

Element1: 80x80 Numpy Array containing pixel values of sand around 40 pixels around the car

Element2: orientation – The angle between the current car position and the goal. This is

Element3: Negative orientation

Element4: Difference of distance between current position to the goal and last car position to the goal

Environment

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

The actions are:

Rotation: The angle the car rotates along the x axis. The value is in the range

Velocity: The displacement of car along x axis. It can have value between 0. 4 and 2.4

The Rewards are given by environment at each step the agent

Condition	Reward	Comment
Car is off the road	-2	

Car is on the Road but distance to Goal is reduced	5.0	
Car is on the Road but distance to Goal is increased	2.0	
Car Hit the Boundary	-50	Car is moved to a random location after it hits the boundary
Cat Reaches the Goal	+100	Once goal is reached, the goal is swapped with another goal
Living Penalty	-0.5	

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

II. 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.

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

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. As target values are determined for both the critic and actor networks, copy of both of these networks and soft update their weights are periodically updated to the respective target networks using polyak averaging.

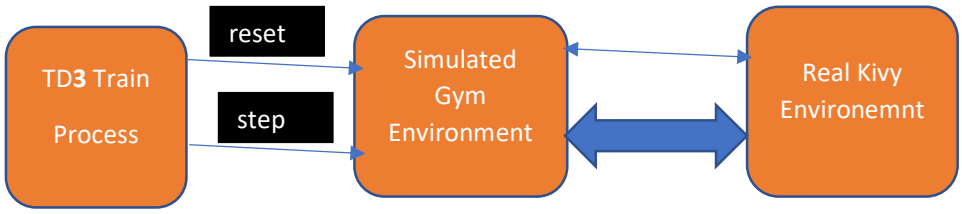
For Details of TD3 algorithm please visit:

https://github.com/monimoyd/P2_S9

III. Methodology and Solution approach

i. Simulated Gym Environment to encapsulate the Kivy Environment

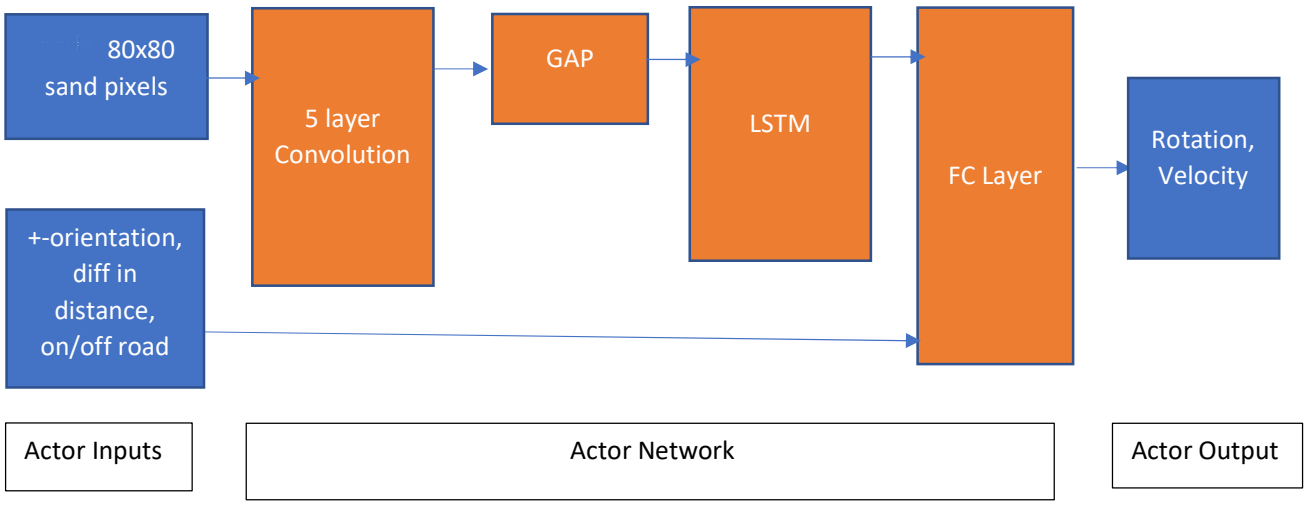
Kivy Environment does not provide methods like reset, step which is very easier to work for any RL project. To solve this I created a simulated Gym Environment which interacts with Kivy based on Multiprocess Queue and Event mechanism provided by Python. The real Kivy environment works on a separate process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



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

i. first element is a 80x80 Numpy array representing the pixel values of sand 50 pixels around the car position
ii. Second element is a Numpy Array having 4 parameters, these are
    a. Orientation of car to the goal
    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 32 3x3 filters with stride 2 and Elu Activation is used. Last layer has 32 3x3 filter with stride 1

GAP Layer:

Global average pooling layer is added after 5 convolution layer which transform into 32x1x1

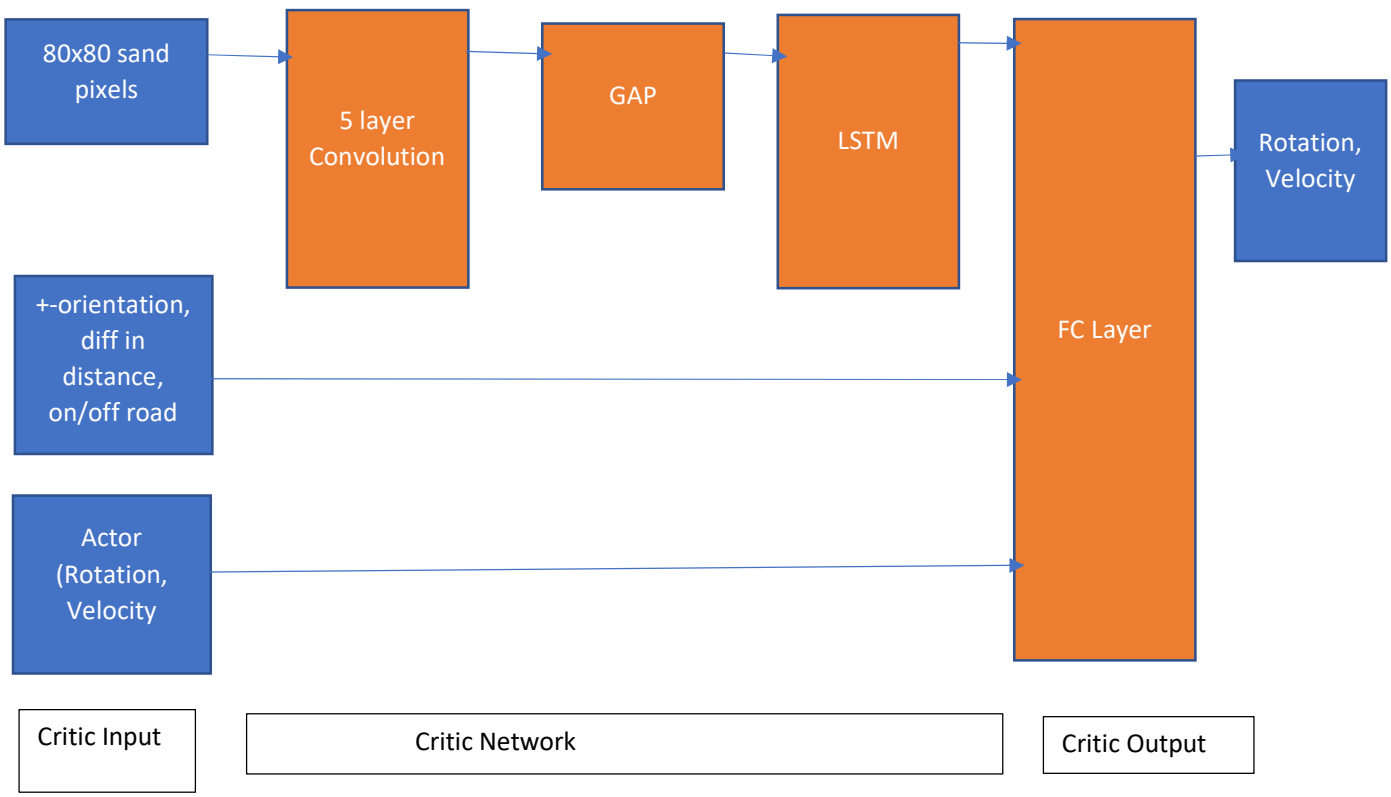
LSTM Layer:

LSTM layer takes the 1 d array and encode into hidden layer of 256 vector tensor

FC Layer:

There are three full connected layers.
First layer layer takes hidden layer output form LSTM and convert into 32 1D tensors and applied tanh activation
Second layer concatenates first layer output and the additional state information (+- orientation to goal, difference in distance to goal, On rod flag value) and output is 64 1d tensor and applied tanh 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
5 layer Convolution
GAP
LSTM
FC Layer
Rotation, Velocity
+-orientation, diff in distance, on/off road
Actor (Rotation, Velocity
Critic Input
Critic Network
Critic Output

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

i. first element is a 80x80 Numpy array representing the pixel values of sand 50 pixels around the car position
ii. Second element is a Numpy Array having 4 parameters, these are
    e. Orientation of car to the goal
    f. Negative orientation of car to the goal
    g. Difference in distance between current car position to the goal and previous car position and the goal divided by 4
    h. A flag on_road, whose value 1 means car is on road and -1 means car is off
iii. Actor output (Rotation, Velocity

Convolution Layer:

There are 5 convolution layers used to transform the road pixel input. Except last layer, each layer 32 3x3 filters with stride 2 and Elu Activation is used. Last layer has 32 3x3 filter with stride 1

GAP Layer:

Global average pooling layer is added after 5 convolution layer which transform into 32x1x1

LSTM Layer:

LSTM layer takes the 1 d array and encode into hidden layer of 256 vector tensor

FC Layer:

There are three full connected layers.
First layer layer takes hidden layer output form LSTM and convert into 32 1D tensors and applied tanh activation
Second layer concatenates first layer output and the additional state information (+- orientation to goal, difference in distance to goal, On rod flag value) and actor output (rotation, velocity) and output is 64 1d tensor and applied tanh activation and Third layer output form second layer transform to 1 tensor which represents Q value

iv. Hyper parameters Used

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

Batch Size: 128
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 environment step : 3
Exploration Noise : 0.1
Policy Noise : 0.2

v. Techniques used to Improve Learning

To improve the learning I have used the following techniques

i. 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. As the steps taken by car is a sequence problem LSTM used in the network to improve performance.
iii. As **LSTM** is used, I changed strategy not to sample randomly from replay buffer. As I used fixed number of timesteps for each episode, records for consecutive episodes are stored sequentially stored in replay buffer and it is easy to get records for a particular episode. As I randomly choose one episode from the list of completed episodes. Next from the timesteps used by the episodes, I choose a random starting point and take a batch size of records from there. In order to avoid
iv. For episode explorations, I used a epsilon value which is initialized to 0.9 and over 40 episodes, I reduce the value to 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

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

VI. Code Structure:

TD3_Train.py _ Main file for training

TD3_Test.py – Main file for evaluating Algorithm

map.py – Kivy Environment file

KivyCarEnvironment – Simulated Environment which mimics open ai gym