**Project title : EndGame**

## Car Navigation using TD3

## ( Twin Delayed Deep Deterministic Policy Gradient ) Deep Reinforcement Learning Algorithm:

**Submitted by : Chaitali Deb Purkayastha**

**Email id :** [**chaitalidebp@gmail.com**](mailto:chaitalidebp@gmail.com)

**Extensive Vision AI (EVA) TSAI Final Project**

**I. Overview**

**Problem Statement**



**We have been given a map of a city and an image of a car.**

Objective:

1. Car should be properly driving through the road .
2. Car should be able to reach the goal with minimum steps.

As guided , Kivy Environment is used:

<https://kivy.org/doc/stable/installation/installation-windows.html>

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

**III. Methodology and Solution approach**

1. **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.

reset

Real Kivy Environemnt

Simulated Gym Environment

TD**3** Train

Process

step

TTIn 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.

1. **Actor Network**

80x80 sand pixels



FC Layer

5 layer Convolution

LSTM

GAP

Rotation, Velocity

+-orientation, diff in distance, on/off road

Actor Output

Actor Network

Actor Inputs

**Actor Input**:

The Actor Network takes Input as two element tuple

1. first element is a 80x80 Numpy array representing the pixel values of sand 50 pixels around the car position
2. Second element is a Numpy Array having 4 parameters, these are
3. Orientation of car to the goal
4. Negative orientation of car to the goal
5. Difference in distance between current car position to the goal and previous car position and the goal divided by 4
6. 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

1. **Critic Network**

FC Layer

LSTM

GAP

5 layer Convolution

80x80 sand pixels

Rotation, Velocity

+-orientation, diff in distance, on/off road

Actor (Rotation, Velocity

Critic Output

Critic Network

Critic Input

**Critic Input**:

The Actor Network takes Input as two element tuple

1. first element is a 80x80 Numpy array representing the pixel values of sand 50 pixels around the car position
2. Second element is a Numpy Array having 4 parameters, these are
3. Orientation of car to the goal
4. Negative orientation of car to the goal
5. Difference in distance between current car position to the goal and previous car position and the goal divided by 4
6. A flag on\_road, whose value 1 means car is on road and -1 means car is off
7. 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

1. **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

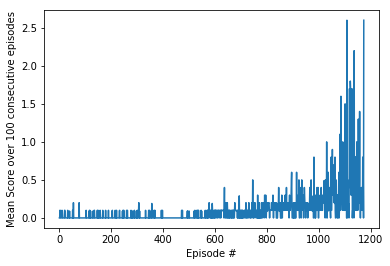
1. **Techniques used to Improve Learning**

To improve the learning I have used the following techniques

1. Initial 10000 timestamps the replay buffer is filled up random policy by choosing action randomly from the action space. This will help better exploration
2. As the steps taken by car is a sequence problem LSTM used in the network to improve performance.
3. 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
4. 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
5. Gaussian noise with mean value of 0 and standard deviation (sigma) of 0.1 has been added to explore states.

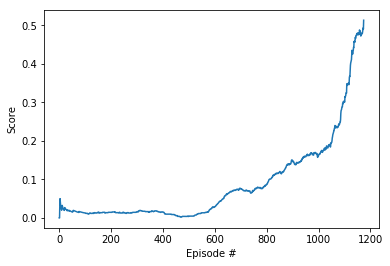
**IV. Results**

* **Plot of episode rewards vs episodes**

****

**Observation:** The scores looks like left skewed histogram which is initially low but gradually increases after 600 episodes

* **Plot for Average Score over 100 consecutive episodes vs episodes**

****

**Observation:**

The mean scores over 100 consecutive episodes are initially flat but increases almost exponentially after 600 episodes. Initial flat scores can be explained as there is a warm up period of first 400 episodes when no learning happens but experience is only added to replay buffer

**IV. Code Structure**

map.py - Python file for Kivy Environment for Car

TD3\_Train.py- This is the main for training using TD3

TD3\_Train.py- This is the main for training using TD3

**V. Conclusion:**

In this project we have used MADDPG algorithm with neural networks having hidden layer dimension of 128x256x128 for both Actor and Critic and for both the agents

1. We have trained the agent and achieved the desired mean score over 100 consecutive episodes of 0.5131 in 1174 episodes
2. Scores (Rewards) earned is a left skewed histogram
3. Mean score over consecutive 100 episodes are initially flat but increases exponentially after 600 episodes
4. To improve learning we have used various techniques like Learning rate reduction if the mean score over 100 episodes does not change by 0.05 for 100 episodes, also added initial warmup period of 400 episodes when no learning happens

**Further Improvements**

Further improvements to the project can be done using:

1. Prioritized Experience Replay
2. Robust Multi-Agent Reinforcement Learning via Minimax Deep Deterministic Policy Gradient (<https://people.eecs.berkeley.edu/~russell/papers/aaai19-marl.pdf>)