Autonomous Navigation of Car in Highway using Deep-Q Learning.

Mothish Raj Venkatesan Kumararaj
Engineering Department, University of Maryland, College Park, Prince George County, mr2997@umd.edu

Abstract: In this project, I try to address a problem statement of navigation of cars on the highway with dynamic traffic test cases, I used the Deep-Q learning method which is a type of reinforcement learning, and designed my model to navigate in a volatile environment. Additionally, we use another environment for the vehicle to park in a parking lot simulation which uses a model-based learning method.

Key Words: Deep-Q Learing, Model-Based Learing, Tensorflow, Pytorch, Neural Network

1 Introduction

Autonomous navigation has been a topic of the decade. In the current scenario, most of the automobile industries trying to achieve autonomous navigation for most of their models. Most of the cars run with the PID controller loop with obstacle avoidance.

In the project, I try to implement reinforcement-learning techniques to improve the performance of the navigation system. For the project, I use the Deep-Q learning method, which is a type of reinforcement learning that is a combination of Neural network learning and Q-learning. In place of the Q-table, we replace it with a neural network to predict the possible action for the next state, and the simulation is carried out for the dynamic environment, and results are plotted.

Additionally, I also implemented a model-based learning technique for the parking lot system since the parking environment is not subject to a drastic change as a highway environment. For the scope of the project for the parking simulation, I concentrated on the orientation of parking and the reaching the goal state. The simulation was also carried out for the parking lot and the results are plotted.

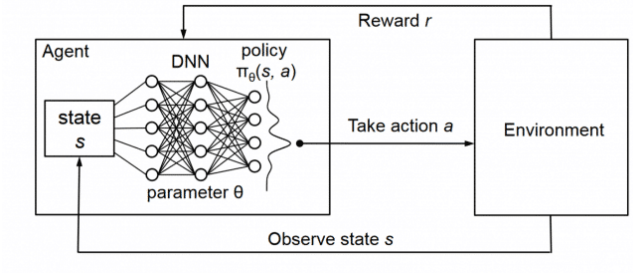

FIGURE 1. Deep-Q Learning Model

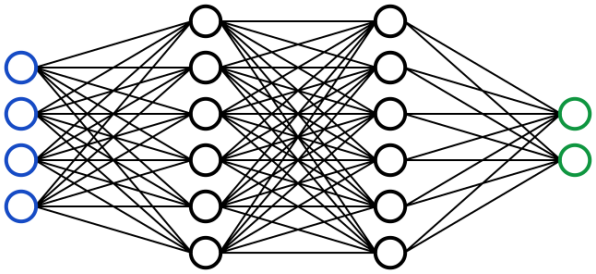

FIGURE 2. Neural Network Model

2 Related Works

2.1 Deep Q learning

Q-learning is a reinforcement learning algorithm that learns the value of the action in a particular state and it does not require any model of the environment hence it's model-free learning. Since the Q-learning technique traverse through the Q-table, if the size of the state space increases the iteration will be time-consuming to traverse through the table. Hence we introduce the Deep Q-learning where the Q-table is replaced with a neural network model.

The input is provided to the neural network that contains hidden layers and input layers according to the user's preference. The output is used to take the next course of action in the Deep Q learning model.

2.2 Model Based learning

Model-based learning is an approach where the information or assumption about the problem domain is made explicit in the form of a model. This model is then used to create a model-specific to learn or reason about the domain.

The reason for the introduction of model-based learning is because, in model-free learning, the model runs with reward as the reinforcement goal, but it can render an ineffective output to an environment that has only sparse rewards where the chance of getting a reward in a random pattern is almost negligible.

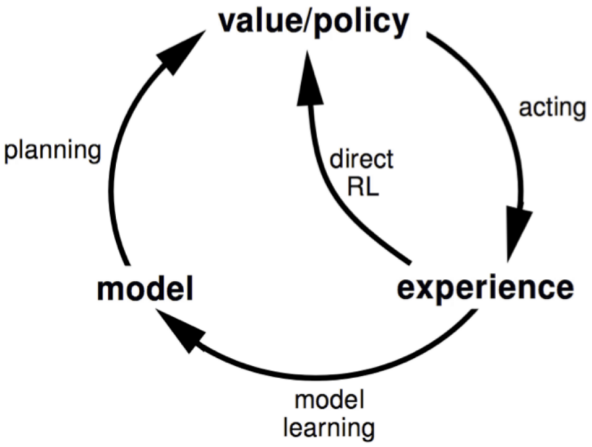

FIGURE 3. Model Based Learning

2.3 Linear Time-Invariant System

A linear time-invariant system (LTI) can be defined as a system that produces an output signal from any input which is subjected to the constraints of linearity and time-invariance. linearity can be defined as a mathematical model of a system based on the use of a linear operator and Time-invariance can be defined as the system function that is not a direct function of time. The general equation is given by

$$f_\theta(x,u) = A_\theta(x,u)x + B_\theta(x,u)u \tag{1}$$

3 Methodology

3.1 Highway Navigation

In highway simulation, we utilize the deep Q learning method where we create our own DQN Model and test it with a different types of inputs to the agent. For each state input, the neural network will estimate the Q-value for each action that can be taken from the state and the objective is to approximate the Q function to satisfy the Bellman equation (1).

$$Q_* = E[R_{t+1} + \gamma \max_{a'} Q_*(s', a')] \tag{2}$$

The loss function is then calculated from the difference in output Q value and target Q value. From the calculated output, the weights are then updated in the Neural network model using backward propagation.

3.1.1 Algorithm

The algorithm we follow to train the highway environment using Deep-Q Learning is mentioned below :

1. Initialize replay memory.
2. Initialize the policy networks.
3. Initialize the target network similar to policy network.
4. For each episode:
   (a) Initialize the starting state.
   (b) For each time step:
      i. Select an action.
      ii. exploration or exploitation
      iii. Execute selected action.
      iv. Observe reward and next state.
      v. Store experience in replay memory.
      vi. Sample random batch from replay memory.
      vii. Preprocess states from batch.
      viii. Pass batch of preprocessed states to policy network.
      ix. Calculate loss between output Q-values and target Q-values where we pass current state to the target network for the next state.
      x. Gradient descent updates weights in the policy network to minimize loss.
      xi. After certain time steps, weights in the target network are updated to the weights in the policy network.

3.1.2 Exploration vs Exploitation

In exploration, the agent will try to select the action whose environment is least explored and the exploitation part is where the agent prefers an action that is known and yields a higher reward. Initially, we use the epsilon greedy rate ϵ to be 1 then in the later stages the agent becomes greedy and the rate decays and then switches to exploitation.

3.1.3 Experience Replay

Experience replay technique is used during training of the network where we store the experience in the replay memory as the experience e_t at time t

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1}) \tag{3}$$

In this data set, s_t and a_t is the state and the action taken on the state. r_{t+1} is the reward of the state action pair s_t and a_t. s_{t+1} is the next state of the environment.

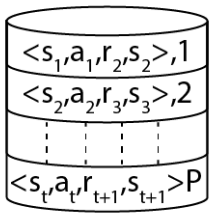

FIGURE 4. Experience Replay Dataset

The agent's experience at each time-step is stored in the replay memory which is a set of the finite size of the last experiences. We randomly sample from the replay memory for training the agent. The reason we do that is since initially, we are training the agent only with sequential experience from the environment which creates consecutive samples, which can result in a correlation between each data leading to inefficient training. By introducing experience replay, we can break the correlation and provide better training to the agent.

3.1.4 Loss Function

The loss function we use in our training model is the Mean Squared error. It is the simplest and most common loss function. To calculate the Mean Squared Error, you take the difference between your model's predictions and the ground truth, square it, and average it out across the whole data set.

3.1.5 Loss Calculation

After all the experience data set is stored in the replay memory we then sample random data from the replay memory then we send the batch of data into the model. After forward propagation, we receive the Q values for the state-action pair. Then, we subtract it from the optimal Q value for that state and action obtained from the right-hand side of Bellman's equation (1).

The loss calculation formula is given by,

$$loss = Q_*(s,a) - Q(s,a) \tag{4}$$

$$loss = E[R_{t+1} + \gamma \max_{a'} Q_*(s',a')] - E[\sum_{n=0}^{\infty} \gamma^n R_{t+n+1}] \tag{5}$$

In the given equation, the Q_* is the maximum Q value of the next state and action pair. Hence, we use forward propagation two times in each iteration to get the Q values of the current state and the maximum Q value of the next state. Backward propagation is done using the loss calculated. Gradient descent is performed to update the weights in the model.

3.1.6 Target and Policy network

In the above-given equation for the loss function, the q' is maximum Q value of the next state s'. If we are using the same neural network to forward propagate twice, then we need to update the weights to calculate the next state and Q values(s',q'). After updating the weights of the neural network, the Q-values move towards the Q-Target values and simultaneously the Q-Target values also moves away in the same direction which makes the optimization chase its tail and become unstable. Hence we create a target network which is an exact copy of the policy network. target network weights are frozen and we update the weights in the target network by taking the weights from the policy network after a certain specified time step.

3.1.7 Optimization

For the optimization of the loss function, I used the Adaptive Moment Estimation which is an algorithm for the optimization of gradient descent. It is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm, efficient with a large problem involving a lot of data or parameters.

3.2 Parking System

In addition to the proposed problem, I also have tried implementing a Model-Based reinforcement learning technique to simulate the parking environment. The parking system is a continuous control system and I have tried to implement Linear Time-Invariant system

3.3 Parking System

The methodology that is followed in the learning of parking environment is provided in the chart below, where we collect our own data set from different experiences from the environment which we use for training our model in the supervised learning fashion.

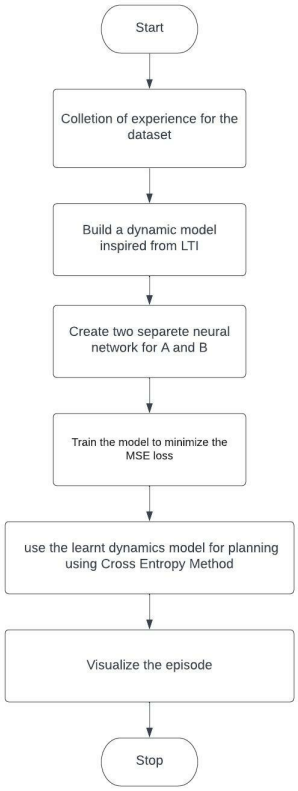

FIGURE 5. Algorithm chart - Model Based Learning

3.3.1 Model based learning

In this particular environment we consider a optimal control problem of Markov Decision Process where the reward function **R** is known to us ant the deterministic dynamics $s_{t+1} = f(s_t, a_t)$ is unknown

In the given environment, we learn the model of the dynamics $f_\theta \approx f$ using the regression on interaction data, and then we use the learn dynamic model to find the optimal trajectory

$$\max_{a_0, a_1, a_2..} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \tag{6}$$

3.3.2 Model

Initially, I randomly interact with the environment to collect the experience batch.

$$D = s_R, a_t, s_{t+1 t\epsilon|1,N|} \tag{7}$$

Then I designed a model to represent the dynamic system. For this project, I utilized a structured model inspired by the Linear Time-Invariant Systems(LTI) which can be represented as follows.

$$f_\theta(s, a) = A_\theta(s, a)s + B_\theta(s, a)a \tag{8}$$

where s and a are the state and action pairs and we parameterize A_θ and B_θ as two neural network.

3.3.3 Loss Calculation

For the loss function for the given problem we use the Mean Squared Error, calculated by the difference between your model's predictions and the ground truth, square it, and average it out across the whole data set. the MSE is given by the formula.

$$L^2(f_\theta; D) = 1/|D| \sum_{s_t, a_t, s_{t+1} \epsilon D} ||s_{t+1} - f_\theta(s_t, a_t)||^2 \tag{9}$$

3.3.4 Optimization

For the optimization of the loss function, I used the Adaptive Moment Estimation which is an algorithm for the optimization of gradient descent. For the particular problem, we use stochastic gradient descent, which is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.

3.3.5 Cross Entropy Method

Cross-Entropy Method (CEM) is a sampling-based optimization algorithm applicable to continuous problems. After training the model we utilize it for planning using CEM. Initially, we draw samples from Gaussian distributions over sequences of actions then we minimize the cross-entropy between the current distribution and a target distribution which is defined by top-k performing sampled sequences.

4 Simulation

For the simulation environment, I utilized the library called gym specially designed to train reinforcement models. For this specific project, I used the highway environment and parking lot environment which can be imported using the gym library.

4.1 Highway Simulation

The highway simulation is a dynamic environment that contains an ego vehicle that can be controlled by the user. The input commands for the ego vehicle are discrete with the commands that are provided below.

0 : Switch to left lane

1 : Idle

2 : Switch to Right lane

3 : Move Faster

4 : Move Slower

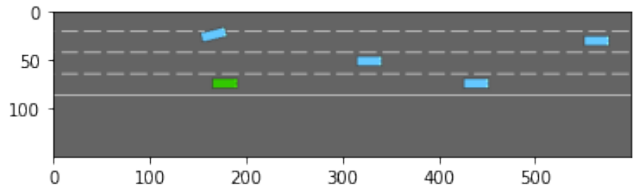

FIGURE 6. Highway Environment

The goal of the environment is to navigate with the ego vehicle quickly in different environment. The inputs we provide to the environment are of two types which are provided below.

4.1.1 Observation Input

The observation input is specific to the environment which provided in the observation stack. this input is provided to the Neural network input and the weights are altered accordingly. The input type and the environment are provided below

Vehicle	x	y	v_x	v_y
ego-vehicle	5.0	4.0	15.0	0
vehicle 1	-10.0	4.0	12.0	0
vehicle 2	13.0	8.0	13.5	0
...	...	...	...	...
vehicle V	22.2	10.5	18.0	0.5

FIGURE 7. Kinematic Observation

This is the type of input used in the observation input type of code. Since we are controlling only the ego vehicle, I only sent the observation of the eqo-vehicle which is the first row in the observation stack, which is a kinematic type observation that contains $[\text{"}x\text{"}, \text{"}y\text{"}, \text{"}vx\text{"}, \text{"}vy\text{"}, \text{"}cos_h\text{"}, \text{"}sin_h\text{"}]$ as the observation stack.

4.1.2 Image Input

For the input image, there is an observation stack of images at each states is available in the environment, but I tried to implement my own code to get the image and process the image from the environment to train the model. The reason I tried to get my own data from the environment is to increase the generality of the code so that it can work for any given environment with less modification. The processed input image is trained in the neural network environment.

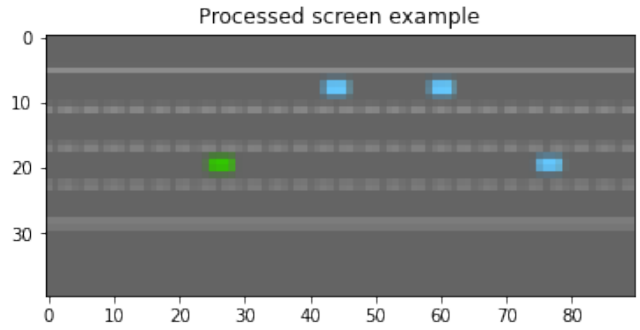

FIGURE 8.

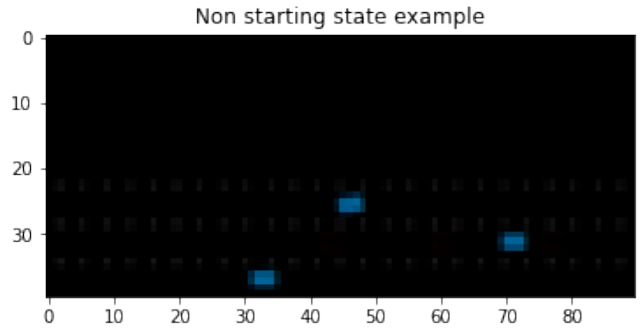

FIGURE 9.

4.2 Parking Simulation

Parking simulation is a model of a parking lot where the world is not a dynamic environment. The reward point for this environment is that the robot should reach its designated parking point. The parking simulation environment is a continuous control environment where it implements a reach-type task, where the agent observes their position and speed and must control their acceleration and steering to reach a given goal.

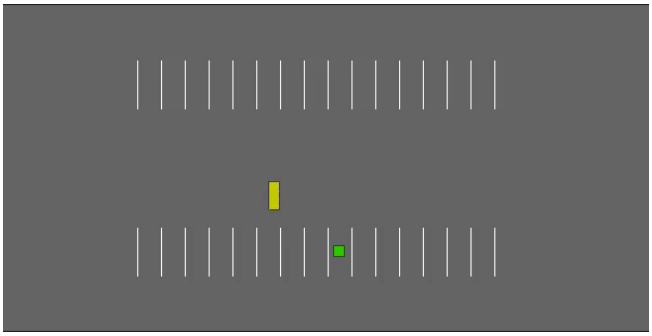

FIGURE 10. park Environment

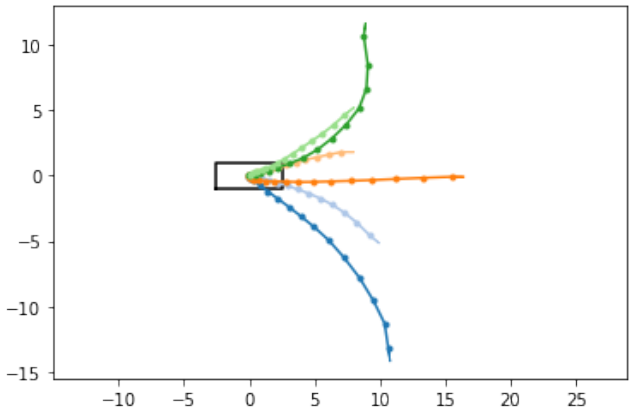

FIGURE 11. Car Trajectories

5 Results

The performance graph is calculated with moving average and loss calculation reduction over time. the performance graph for each parameter and for the different environments is provided below

5.1 Highway Simulation

5.1.1 Observation Input

This given moving average and the loss function below is for the observation type input of the parking lot simulation. which has 1 hidden layer and 5 output actions. The following result is trained with the following parameter

$Episode$: 300
$batchsize$: 256
$TimeStep$: 1
$EpsilonDecayRate$: 0.001
$InitialEpsilon$: 1
$Optimizer$: $Adam$
$LossFunction$: MSE

For the neural network in the DQN model the following parameters are provided as input
$InputLayer$: $NumberofStateFeatures$
$HiddenLayer1$: 24
$HiddenLayer2$: 32
$OutputLayer$: 5
$ActivationFunction$: $ReLu$

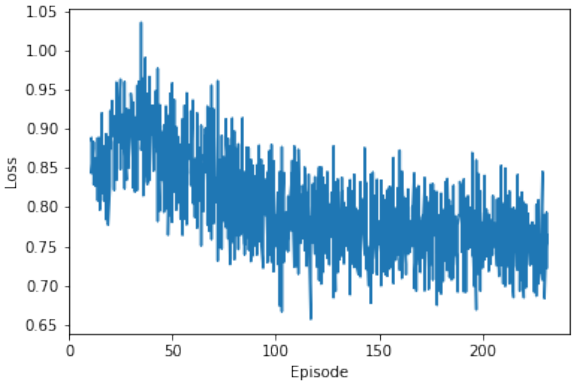

FIGURE 12. Loss Graph Observation Input - Highway

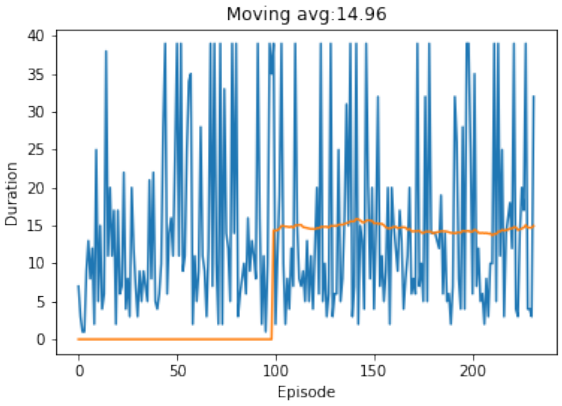

FIGURE 13. Moving Average Graph Observation Input - Highway

5.1.2 *Observation- Observation Input*

From the given graph, it is clear that the performance of the observation input is not great and the loss function curve is almost flat towards the end. One of the observations that have been seen in the environment is that, since we are controlling only the ego-vehicle in our kinematic observation stack, the vehicle tends to maintain one wheel and control the speed to avoid collision instead of shifting to another lane.

5.1.3 *Image Input*

This given moving average and the loss function below is for the image type input of the parking lot simulation. which has 1 hidden layer and 5 output actions. The following result is trained with the following parameter

$Episode$: 300

$batchsize$: 256

$TimeStep$: 1

$EpsilonDecayRate$: 0.001

$InitialEpsilon$: 1

$Optimizer$: $Adam$

$LossFunction$: MSE

For the neural network in the DQN model the following parameters are provided as input

$InputLayer$: $NumberofStateFeatures$

$HiddenLayer1$: 24

$HiddenLayer2$: 32

$OutputLayer$: 5

$ActivationFunction$: $ReLu$

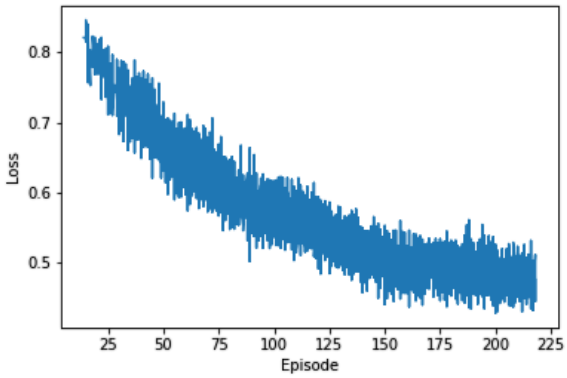

FIGURE 14. Loss Graph Image Input - Highway

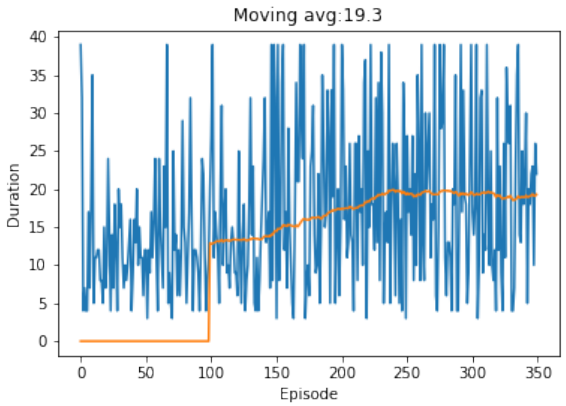

FIGURE 15. Moving Average Graph Image Input - Highway

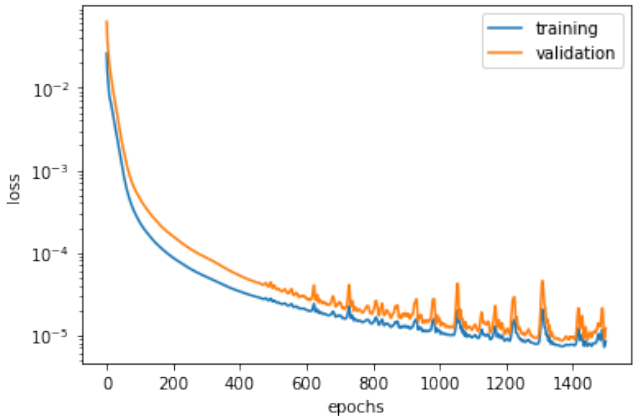

FIGURE 16. Loss Model Based Learning- Parking

5.1.4 Observation- Image Input

From the given graph, it is clear that the performance is better compared to the observation input type and the moving average is increased gradually over the 300 episodes. The performance of the overall system was not up to the mark because I was trying to generalize the code and did not use the observation image stack from the environment, instead I tried to create a code where a process image is generated by me. The performance can be improved by sending more than two images to the system and by training more episodes.

5.2 Parking Simulation

For the parking simulation, I calculated the loss graph and provided the results of the loss graph for the following parameters

$Episode : 1500$
$trainingratio : 0.7$
$Optimizer : ADAM$
$LossFunction : MSE$

The neural network for A and B on the Linear Time Invariant (LTI) system is given as following:

$InputLayerA : Statesize + Actionsize$
$InputLayerB : Statesize + Actionsize$
$HiddenLayer1A : 64$
$HiddenLayer1B : 64$
$OutputLayerA : Statesize * Statesize$
$OutputLayerB : Statesize * Actionsize$
$ActivationFunction : ReLu$

5.2.1 Observation- Model Based Learning

The results provided were performing at a good rate and we filtered the best episodes using the Cross entropy method. The loss graph curved and reduced gradually and provided the desired results compared to the model-free learning.

6 Conclusion

Overall, even the environment is different it is clear that the model-based learning performs better than the model-free learning. But one of the major flaw for the model based learning is the dependency to the model environment. Between the different types of input in the DQN technique, the image has performed better because the observation stack which was trained only controls the ego vehicle movements but does not take consideration of the other vehicle position which is not feasible for the dynamic environment. The image based input performance can be increased by sending better processed image and multiple images to the neural network model.

7 References

1. Highway Environment and Parking Environment `https://highway-env.readthedocs.io/en/latest/index.html`
2. Edouard Leurent and Jean Mercat. Social attention for autonomous decision-making in dense traffic. In Machine Learning for Autonomous Driving Workshop at the Thirty-third Conference on Neural Information Processing Systems (NeurIPS 2019). Montreal, Canada, December 2019. pp. 331–334.