Self-driving Car using Deep Reinforcement Learning

Feras Harah
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
ferasharhcr7@gmail.com

Abstract. An significant aspect of the Self-driving vehicle is the steering system that can mimic the actions of human drivers as a self-driving car controller. This removes the need for human engineering to predict what is critical in the picture and to set down all the appropriate rules for safe driving. Deep Renforcement Learning (DRL), due to its ability to perform and the interaction with the virtual simulation environment, is the most mature computer training system for this mission. In my project lab, my car in Carla simulator is being trained using DRL algorithms, our code is pythoned with the sublime text editor and we have used anaconda3 as a prompt for instructions, and we have also taken Colab to use it. First, we used the DQN(Deep Q-Learning) algorithm to train our model to see whether our agent will take ongoing steps, and we try using a DDPG algorithm that is useful in continuing tracking of behavior.

Keywords:
Carla, DQN, DDPG, Reinforcement learning, self-driving car.

1 Introduction

Self-driving vehicles have become popular, and need little human interaction to prevent road accidents. The environment must be conscious of the number of sensors mounted on the car. There have been numerous suggestions for learning about the driving strategy for taking regulation steps. Due to advancement of CNN, the end-to-end tracking of learning is being used increasingly to train the model of the neural network across vast volumes of time-consuming data. Reinforcing learning (RL) can however be trained without the comprehensive labeling of data needed for supervised learning. It was recently seen in the area of self-employed vehicle analysis as a promising strategy for driving policy.

End-to-end supervised neural network (CNN) learning is commonly used, requiring a high level of data. By testing and error, DRL attempts to learn an optimal environment-related approach. We will use the DQN DDPG algorithms for continuous performance Based on the problem domain, the space of potential acts can be discrete or continuous, which can have an influence on the range of algorithms that must be implemented. It is difficult to simply switch between different discrete control values at short intervals such that continuous control is easier.

During this job, I will prepare CARLA simulator for training and attach the requisite objects such as vehicles, the camera and the sensors of collision. I then proceed to incorporate our RL algorithms. First, I developed our environment class with which our agent communicates and the incentives functionality used in training has been introduced. Next, I built our agent, we must use the reply memory to practice in DQN in order to make the model more reliable, and tried to do more random stuff at the start of the training because the model prediction is more sluggish. Then I added numerous situations I wanted to prepare our agent.

I've concentrated on two models I used to train our agent here. I used a much complex Xception model with additional parameters. After that I have turned the model into a much easier 64*4 CNN model. In addition to the reward functions we carried out, I showed the effects on the tensorboard to test our model such as accuracy and loss of model. Lastly, on the second model, our agent checked good behavior, but turning right or left it takes hundreds of thousands of episodes to be conditioned in order to take good action. We cannot train more than 12 000 episodes because of our limited capabilities.

.

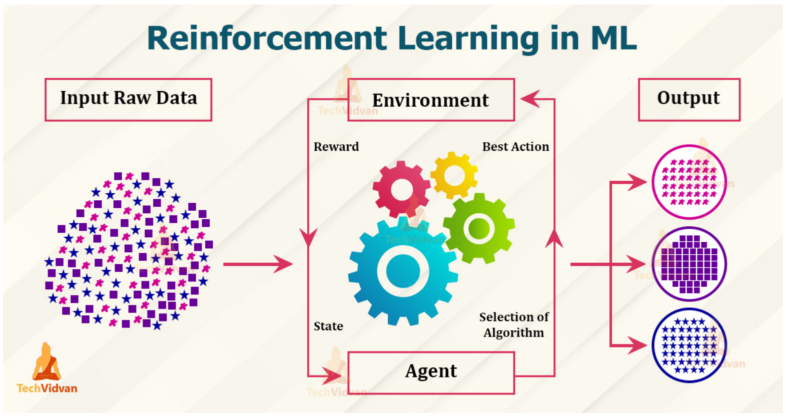


Figure 1: Reinforcement-Learning-in-ML

2 The Analysis Methodology:

At the beginning our Supervisor asked to obtain a theoretical perspective on reinforcement learning and the algorithms used during self-driving training in vehicles, we looked for the 5 most important articles, which cover the subject and included Q-learning (DQN) and DDPG algorithms.

The sentdex tube channel we used to attempt to construct a DQN algorithm for our first model. The sublime editor, Python 3.7 and Anaconda 3 were also used for model training. At the beginning we face challenges in teaching the models. I also tried to use colab, but because of GPU restrictions we had difficulties importing CARLA simulators into colab, so we tried to train our model in anaconda3 in Xception model up to 1000 episodes, and in 64*4 CNN model up to 12,000 episodes.

Every week we updated our supervisors about the work results, the challenges raised, the directions to address them, and the next mission to be carried out.

.

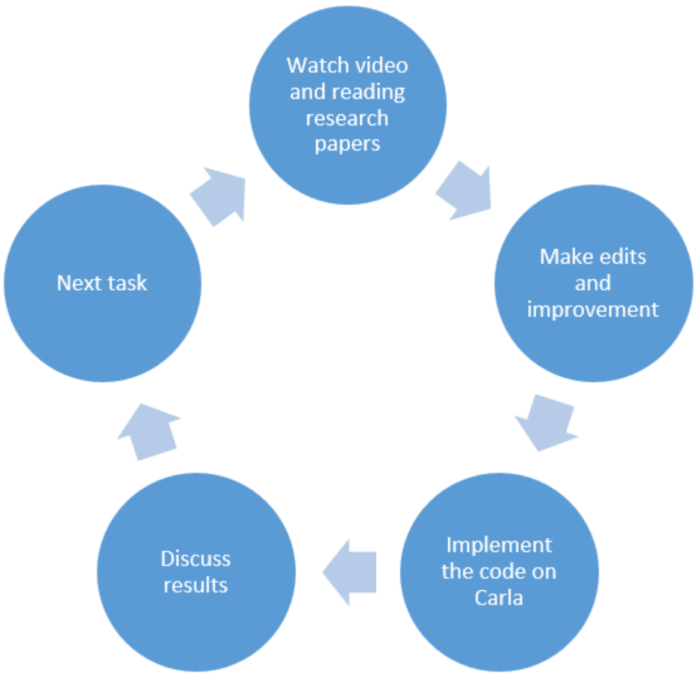

Watch video and reading research papers
Make edits and improvement
Implement the code on Carla
Discuss results
Next task

Figure 2: Working path

3 Results of the project:

I would have a description of each task and how we treat it and then use algorithms to find the best solution to it. We will speak about several tasks with the requisite details and results.

3.1 Theoretical background

First, I gathered the most important documents on our subject and received a clear understanding of how to prepare our agent.

3.1.1 Description

In one of the papers, to address the issue of the continuous space for intervention the author suggests an updated Deep Deterministic Policy Gradients (DDPG) algorithm such that continuous steering and acceleration angles can be achieved since the space for operation must be continuous and conventional Q learning must be not addressable. In addition, designs a more suitable route to clear obstacles in accordance with the vehicle limitations including internal and external

Another paper based on the various types for used algorithms, the author implements a DRL lane holding assist method. The space for potential acts can be discrete or continuous, depending on the problem domain. For a discrete group of actions it covers Deep Q-Network Algorithms (DQNs), whereas the Deep Deterministic Actor Critical Algorithms (DDACs) are included in the continuous action category.

3.2 Carla's setup:

First, I had to download the CARLA simulator, train my agent, evaluate and execute some commands on the simulator to produce the real world.

3.2.1 Description

After installing Carla, we have everything we need to run on the main CarlaUE4 directory file to connect vehicles, we need to run the spawn npc.py code, and can define the car number as well. For the code. manual-control.py, this creates another window (pygame) that can control the car by using the WASD keys, python 3.7 must be used to run the codes here.

3.2.2 Experiment

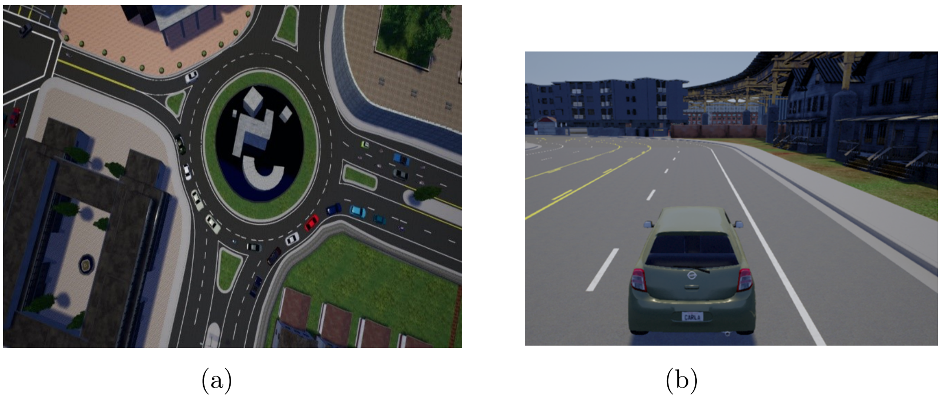
At the fig(a) below you can see that many cars are spawned at our environment, and at fig(b) a new window called pygame is created to control the agent.

3.3 Car management and data acquisition

In Carla there are different kinds of objects I have to create like the 'world.' This is our setting, the actors in the world. Actors are like vehicles, my vehicle sensors, pedestrians. Then we have the blueprints that are our actors' characteristics.

3.3.1 Description

First I will connect to our server to build a car on the environment, then access the Tesla model 3 drawings, now that the car can spawn at any random spawn point, my car will run on our server for 5 seconds and then be re-mediated.



(a) (b)

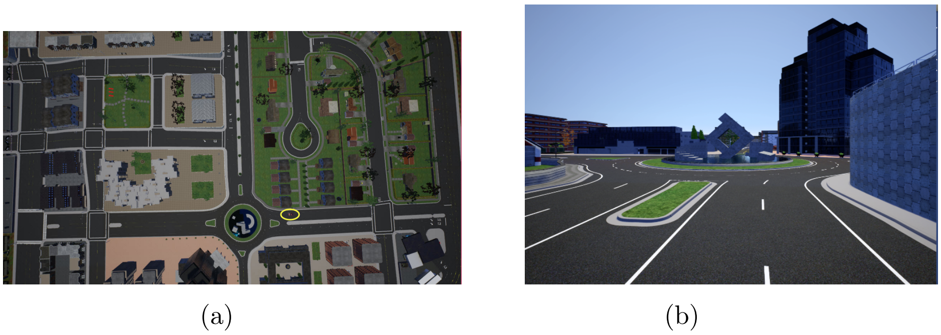
Figure 3: (a) spawning cars (b) pygame in Carla

Second, I need to make a camera on my car and find out how to access it. Ideally this cap camera is our main sensor. We load on the sensor blueprint and set those attributes.

We might have the camera sensor positioned relative to our vehicle, let's say we want to move 2.5 and 0.7 ahead. Finally yet importantly, to get sensor pictures, because we want to hear. We will take the sensor data and send it to the process img feature.

3.3.2 Experiment

We could see our vehicle, labelled with the yellow circle on fig (a). fig(b) shows a new window for showing the camera sensor and transmitting these images to our model.



(a) (b)

Figure 4: (a) The agent car (b) Camera surrounding environment

3.4 Reinforcement Learning Environment

The establishment of CarlaOpenAI was a leader in opening up strengthening learning environments. We will build our environment class with which our agent will communicate, and incorporate our award feature used for training.

3.4.1 Description

A phase method that returns to our world: observation, reward, completed, extra info, as well as a reset method that restarts the environment on some kind of flag.

I created a colliding sensor that tells a history of events and saves the colliding to see whether or not we collided, we wait 4 seconds before we start making the collision sensor and not detect the collision when the car breaks from the sky. Then we can log the actual start-time for the episode for the reset function if all is OK, make sure the brake and throttle are not being used and send back our first observation.

We take action and then return the observation, reward and execution of any extra info as normal in the strengthening learning model. We have three acts in our case: turn left, straight, and turn right.

$$ifaction = \begin{cases} 0 \;\; \text{steer left} \\ 1 \; \text{straightforward} \\ 2 \;\; \text{steer right} \end{cases} \tag{1}$$

The speed of the vehicle is converted to KMH for our incentive feature, in order to prevent the officer from only driving in a close circle.

$$KMH = 3.6 * \sqrt{{V_x}^2 + {V_y}^2 + {V_z}^2} \tag{2}$$

$$reward = \begin{cases} -200 \;\; \text{if three is collision} \\ -1 \;\; \text{if KMH<50} \\ +1 \;\; \text{else} \end{cases} \tag{3}$$

3.5 Reinforcement Learning Agent

We will build our class of agents who will communicate with this environment and house our actual model of reinforcement.

3.5.1 Description

We have an increasingly changing main network, and then a target network, which we upgrade every n move or episode and use to decide what potential Q values will be. We want to change Q values (fitting) every step we take, but we also attempt to forecast our model.This means that we're training and predicting at the same time. This makes our model even slower and unpredictable so, we almost all train batches on neural networks. One way to solve this is through a memory replay principle that saves current status, behavior, reward, new state transitions.

We used premade Xception model for the first time even later on other models such as convolution neural networks. We add GlobalAveragePooling to our output layer, and of course we also add the output of 3 neurons that the agent will take for any action possible. We start the training when there is enough samples in replay memory and use our model in order to obtain current Q values and target model to obtain future Q values.

$$New\_q = reward + discount * max\_future\_q \tag{4}$$

Then we fit in all samples as a single batch and only log per episode, not as a training phase, and every five episodes upgrade our target model. In the end, we only need to practice using random knowledge of data to initialize, but our trainer would really like to be finished as soon as possible.

3.6 Action of Reinforcement Learning

Our models are now ready to practice, but before that, we want to have the agent at the start of our training, as the estimation of the model is lighter.

3.6.1 Description

In addition to the rewards and epsilon values, we used tensorboards for visualising our measurements such as accuracy and loss of our model. Typically, a log file and a datapoint per fitment are given.

When an agent learns an environment, it transitions from "exploration" to "exploitation." Now, the model is greedy and always useful for maximum Q values. We need an agent to explore and so use epsilone value to allow the agent to perform random steps. We will have a high epsilone when we start, which means a high probability of choosing a random action instead of predicting it via our neural network. A random choice would be much quicker than a predictable process.

Now we are able to continue to iterate over as many episodes as possible. There will be an atmosphere until the end flag, as we play, we want to either perform a random action or find out our current action based on our model agent and eventually the epsilon value will decay over the episodes.

3.7 Scenarios of the training

This is the most difficult aspect that takes a lot of time. Due to the minimal machine capability we encountered several challenges before choosing to pick a scenario.

3.7.1 Description

At the beginning after downloading the necessary packages, we attempt to train our model using anaconda3 on my local laptop. The model successfully began

Algorithm 1 Pseudo code for Self-driving Car using Reinforcement Learning

```
1: while true do
2:     if np.random.random() > epsilon : then
3:         action = np.argmax(agent.get_qs(current_state)) //model predict
4:     else
5:         action = np.random.randint(0, 3) //Get random action
6:     new_state, reward, done,_ = env.step(action)
7:     agent.update_replay_memory((current_state, action, reward, new_state, done)) //update replay memory
8:     current_state = new_state
9:     if done: then
10:        break
```

training but several times stopped because of GPU's limited ability. After downloading our CARLA simulator with several steps on Colab, we didn't train our model because we had a GPU run-time constraint so we need better machine or Colab premium. We tried Google Colab because it provides a higher efficiency in training. Finally, we agreed to practice on our local laptop with anaconda and we used checkpoints for the last training episode.

We also tried to train our agent in a real-life setting, and we tried to spawn 100 different vehicles of different sizes, ranging from bicycles, cars and trucks.But because of the colliding with the scatter agents, the training ended early. Then, we tried to train the agent with complex situations, which change illumination conditions and weather over time. However, training fails again because the RGB photos from the front camera need more image treatment under inadequate conditions in order to achieve acceptable performance, because our computer capacities are limited.

Finally, we agreed just to keep the world bright under regular circumstances without dynamic transportation so that things are easy and machine sophistication minimized. If the officer was trained in these conditions for 12,000 episodes, we could see a good enough model that went very well on right roads and turned right and left as needed.

3.8 Training using Xception model

We train our model first with Xception, a successful model for self-driving testing vehicles. It's a Keras model with71 cached layers.

3.8.1 Description

For 1000 episodes, we first developed our model with Xception to see if it was good for our training. It has practically 23 million trainable parameters when testing the Xception model (with our speed layer): And we found that the model was fairly reliable, (like 80-95 %), so it means to me that we've been almost surely overfit every time, too. We can't expect an outstanding agent who performs very well but in our case the agent has done nothing but loop the same.

3.8.2 Experiment

We will find that the accuracy value is approximately 95 per cent higher than anticipated; we will have to decrease it to prevent such an explosion because of explosion of Q-values as the crash rewards are -200. At the beginning, epsilon values will drop from 1 for a random procedure and then decrease in episodes to predict behavior for the model. The model can be regarded as the average reward over time, we can see from the test that the agent simply learned to do 1 thing and take one action because its q values are static (model outputs same Q values no matter the input)

.

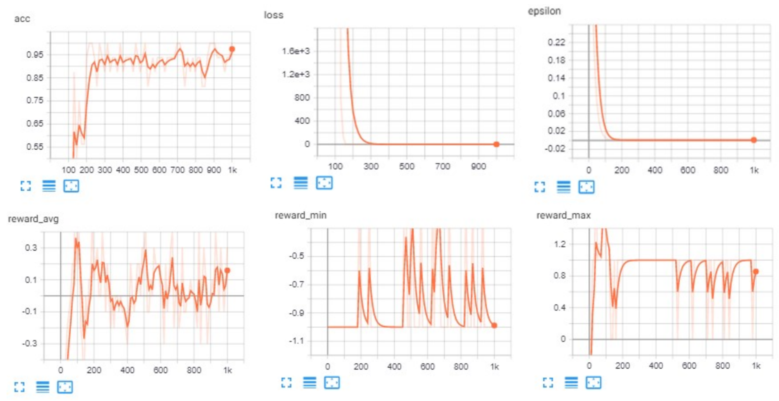


Figure 5: Xception model graphs results

3.9 Training with CNN model

In order to simplify our model, we have used a 64x4 neural neural network (CNN) with 3 million parameters, so there are lower parameters to understand.

3.9.1 Description

Our neural network have of 4 hidden layers as show in the fig.6 It is easier and lets us solve the loss function explosion problem and the Q values.We many times trained the agent with the same model with control points before 12,000 episodes were achieved. Figure displays our CNN, consisting 3 hidden layers that use average pooling ,flatten layer and output layer of 3 neurons to predict 3 actions.

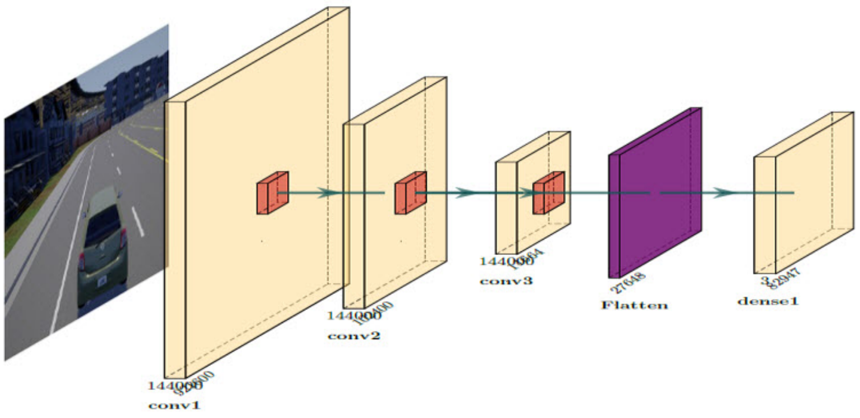


Figure 6: CNN Model

3.9.2 Experiment

We can see from these graphs that our model is about 85 percent reliable on average. No explosions.so, our loss function strengthens our model. The epsilon value then fluctuates as an agent tries to experiment many times.

For rewards, we can see that over the course of time the maximum reward trends up to 12,000 episodes, and more time could have been given. The minimal reward (that is, the worse of an agent), seems to have not significantly improved. Finally, we can see that the total reward improved steadily, so the net gain was reached, and this experiment was our highest performing model.

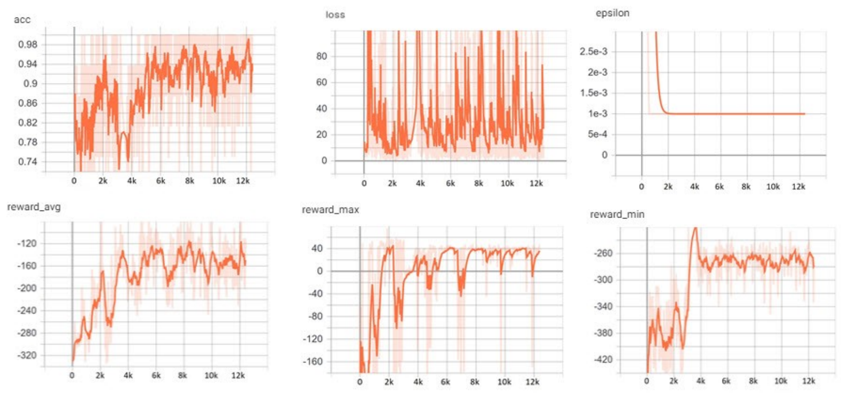


Figure 7: CNN model graphs results

3.10 Test Stage

I monitor our Xception and CNN models on my local machine with anaconda3 to see whether they are qualified or not.

3.10.1 Description

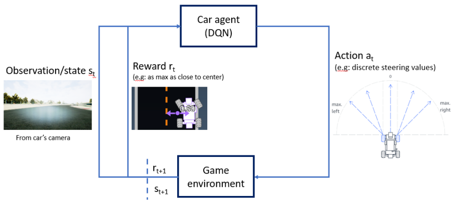
We checked our agent with the training models in the CARLA simulator, we also posted training results and tests on the Git-hub to enable you to display the trained models in a different vehicles environment.

3.10.2 Experiment

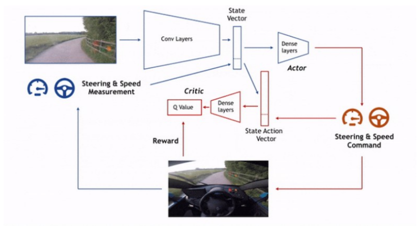
The car then drives very well without reaching another car, but it does not go as fast as it wants to in the picture. For a longer period of time the switch button can be pressed.

3.11 Summary

We could summarized our work with the charts below.



(a)



(b)

Figure 8: Process in RL with DQN

Acknowledgments

The author thanks Khalid Kahloot for leading the project and offering advice in the event of any difficulties.

References

[1] sentdex: https://pythonprogramming.net/Self-driving cars with Carla and Python

[2] sentdex: https://pythonprogramming.net/q-learning-reinforceme nt-learning-python-tutorial/

[3] Zong, X., Xu, G., Yu, G., Su, H. et al., "Obstacle Avoidance for Self-Driving Vehicle with Reinforcement Learning," SAE Int. J. Passeng. Cars â Electron. Electr. Syst. 11(1):30-39, 2018.

[4] E. Sallab, M. Abdou, E. Perot, and S. Yogamani, âEnd-to end deep reinforcement learning for lane keeping assist,âarXiv preprint arXiv:1612.04340, 2016.