Steering Model for Autonomous Driving System using Deep Reinforcement Learning

Kamel Mina Milad Thabit
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
minapower50@gmail.com

Abstract. An important part of the Autonomous Driving System (ADS) is the steering system which supposed to emulate the behavior of human drivers as a self-driving car controller. This eliminates the need for human engineers to anticipate what is important in an image and to foresee all the necessary rules for safe driving. The most mature machine learning framework that can be put forward to do such task is Deep Reinforcement Learning (DRL) due to its ability to work and interact with virtual simulation environment. In this thesis work, we try to train our car in Carla simulator using DRL algorithms, our code is implemented in python with PyCharm editor and we used anaconda3 as a command prompt. First, we used DQN(Deep-Q-Learning) algorithm in order to train our model to see if our agent can take continuous actions. We trained using two models, the first is Xception model which is using Xception that is a good model for training self-driving cars. It is a model in Keras with71 hidden layers. And the other model is a 64x4 convolutional neural network (CNN) that is much simpler and has less parameters to learn compared to the first model.

Keywords:
DQN, CNN, Reinforcement learning, self-driving, ADS.

1 Introduction

Self-driving cars have become popular that doesnât require human input in order to avoid road collisions. It must be aware of the surrounding environment using number of sensors built on the vehicle. Many ideas have developed to learn it the driving policy to take control actions. due to development of convolutional neural networks (CNN), end-to-end supervised learning is increasingly used to train the neural network model through large amount of data thatâs is time consuming. On the other hand, reinforcement learning (RL) can be trained without abundant labeled data required in supervised learning. Recently, reinforcement learning has been considered as a promising technique to learn driving policy in autonomous vehicle researches.

end-to-end supervised learning with convolutional neural network (CNN) is widely used, but requires large amount of data. DRL tries to learn an optimal strategy according to environment reward through trial and error. for output

continuous action, we can use DQN and DDPG algorithms Depending on the problem domain, the space of possible actions may be discrete or continuous, a difference which has a profound effect on the choice of algorithms to be applied. Itâs impossible to switch among different discrete steering values at short intervals in reality so, continuous steering control is better.

In this work, I am going to prepare CARLA simulator to be ready for the training process, and attach the necessary objects like cars, camera and collision sensors. Then I keep going to implement our RL algorithms, first, I created our environment class that our agent will interact with, and implemented the reward function used in training. Secondly, I created our agent that we need to train in DQN using reply memory to make the model more stable and tried the agent takes more random actions at the beginning of training as the model prediction is slower. After that I introduced different scenarios that I tried to train our agent.

I focused here on two models that I used to train our agent. I used Xception model that is much complex as it has more parameters. Then, I changed the model to be a 64*4 CNN model that is much simpler. I showed the results on tensorboard to evaluate our model such as accuracy and loss of used model besides to reward functions that we implemented. Finally, we tested our agent on the second model that showed good actions in straightforward paths, but when turning right or left, it needs to be trained hundreds of thousands of episodes to be able to take good actions. Due to the restricted capability we cannot train our agent more than 12000 episodes.

2 The Technique for Research:

At the beginning of the semester, our supervisor asked to gain theoretical background about reinforcement learning and algorithms used in training self-driving cars, we searched and summarized the most relevant papers about the topic and understood Q-learning., Deep -Q-learning (DQN) and DDPG algorithms that are used in continuous control actions for self-driving cars.

We used sentdex's tube channel to try to build our model in DQN algorithm, and used PyCharm and anaconda 3 to train the model , we face problems to train the models at the beginning as I am new in the field of machine learning , I tried to use also colab but I found problems in importing CARLA simulator in colab , until we tried to train our model in anaconda3 in Xception model up to 1000 episodes, and in 64*4 CNN model up to 12,000episodes. We had a consultation every week with our supervisor to show the progress in the work, what problems we faced, how to solve it and keep going with the next task the following chapters.

3 Project Tasks:

I will give an overview of each task and how we deal with the problem, and then use algorithms to find an appropriate solution. We will discuss 10 tasks with the information and results needed.

3.1 Theoretical background

First, I collected the most relevant papers related to our topic, I gained good knowledge that I used in training our agent.

3.1.1 Description

In paper [3], the author introduces DRL system for lane keeping assist depending on different categories for the used algorithms. Depending on the problem domain, the space of possible actions may be discrete or continuous. For the discrete actions category, he deals with Deep Q-Network Algorithm (DQN) while for the continuous actions category, with Deep Deterministic Actor Critic Algorithm (DDAC) is used.

In paper [4], the author proposes an improved Deep Deterministic Policy Gradients (DDPG) algorithm to solve the problem of continuous action space, so that the continuous steering angle and acceleration can be obtained as the action space must be continuous which canât be dealt with by traditional Q-learning. Also, designs a more reasonable path for obstacle avoidance according to the vehicle constraints include inside and outside

In paper [5], the author mainly focuses on continuous steering control as itâs impossible to switch among different discrete steering values at short intervals in reality, he first quantifies the degree of continuity by formulating the steering smoothness. Then proposes an extra reward penalty. Results show that the proposed penalty improves the steering smoothness.

3.2 Setting up Carla:

First, I had to download CARLA simulator where I can train my agent, and also test it and perform some commands to generate real world at the simulator.

3.2.1 Description

Carla is an open-source autonomous driving environment that also comes with a Python API to interact with it.the main idea of Carla is to have the environment (server) and then agents (clients). After downloading CARLA, I am ready to spawn different types of vehicles also manual_control.py command can be used to control our car with WASD keys.

3.2.2 Results

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

(a)

(b)

Figure 1: (a) spawn vehicles (b) pygame window

3.3 Controlling the Car and getting sensor data

There are several types of objects in Carla which I have to create like the "world." That is our environment, the actors within this world. Actors are things like car, the sensors on my car, pedestrians. and finally, we have blueprints which are the attributes of our actors.

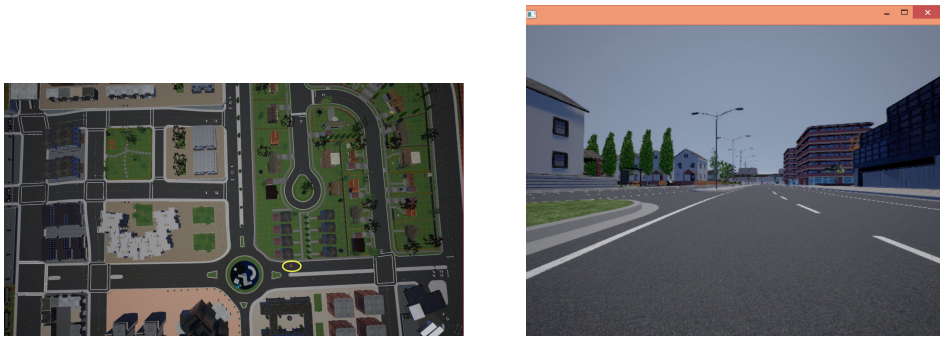
3.3.1 Description

First to create the car on the server (environment), I'll connect to our server, get the world, and then access the blueprints for the Tesla model 3, now the car can be spawned at any random spawn point, my car can be runed for 5 seconds in our server and then cleaned up.

Second, I have to create a camera on our car, and to figure out how to access that data. This hood camera would ideally be our main sensor. we load in the blueprint for the sensor and set some of the attributes. Camera sensor could be placed at a relative position of our car, letâs say we want to move forward 2.5 and up 0.7. finally, to get the images from the sensor, so we want to listen. Weâre going to take the data from the sensor, and pass it through some function called process_img.

3.3.2 Results

You can see our car that are going to be trained marked with yellow circle at fig (a) also, fig(b) shows a new window to display the camera sensor and feed these images to our model.



(a) (b)

Figure 2: (a) our agent (b) camera preview

3.4 Reinforcement Learning Environment

OpenAI pioneered the open sourcing of reinforcement learning environments, we are going to create our environment class, which our agent will interact with and also implement our reward function that used in training process.

3.4.1 Description

our environment will have a step method, which returns: observation, reward, done, extra_info, as well as a reset method that will restart the environment based on some sort of done flag.

I have created collision sensor that reports a history of incidents and stores it at collision_hist to see if we've crashed or not , before creating the collision sensor we wait 4 sec to get things started and to not detect a collision when the car spawns/falls from sky.so, for reset function when everything is ok , we can logging the actual starting time for the episode, make sure brake and throttle aren't being used and return our first observation.

Our step function takes an action, and then returns the observation, reward, done, any_extra_info as per the usual reinforcement learning paradigm. In our case we have three actions: turn left, straightforward and turn right.

$$if action = \begin{cases} 0 & \text{steer left} \\ 1 & \text{straightforward} \\ 2 & \text{steer right} \end{cases} \tag{1}$$

For our reward function, vehicleâs speed is converted to KMH to avoid the agent learning to just drive in a tight 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 are going to create our Agent class that will interact with this environment and house our actual reinforcement learning model.

3.5.1 Description

we have a main network, which is constantly evolving, and then the target network, which we update every n steps or episodes and used to determine what the future Q values. every step we take, we want to update Q values (fitment), but we also are trying to predict from our model. This means that we're training and predicting at the same time. That make our model very slower and unstable so, we almost always train neural networks with batches. One way this is solved is through a concept of memory replay that stores the transitions of current state, action, reward, new_state, done.

we used first the premade Xception model also other models like convolutional neural networks will be used later. we're adding GlobalAveragePooling to our output layer, as well as obviously adding the 3-neuron output that is each possible action for the agent to take. We will begin our training if there are enough samples in replay memory and use our model to get the current Q values, and target model to get the future Q values, then we update Q values if it is not a terminal state according to:

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

Then we fit on all samples as one batch and log only per episode, not actual training step and we update our target_model every 5 episodes. Finally, we just need to actually do training by using some random data to initialize, weâd also like our trainer to go as quick as possible. To achieve this, we can use either multiprocessing, or threading.

3.6 Reinforcement Learning in Action

Now, our model is ready for training process but before that we want the agent to take random action at the beginning of our training as the model prediction is slower.

3.6.1 Description

We used tensorboard for visualization our metrices like accuracy and loss of our model besides to rewards and epsilon values. Normally, there's a log file per fitment, and a datapoint per step.

As an Agent learns an environment, it moves from "exploration" to "exploitation." Right now, our model is greedy and exploiting for max Q values always, we need the agent to explore, so we use epsilon value that allow the agent to take random actions at the starting. when we start, we'll have high epsilon, which means a high probability that we'll be choosing an action randomly, rather than predicting it with our neural network. A random choice is going to be much faster than a predict operation.

now, we're ready to start iterating over however many episodes.an environment will run until it's done flag, as we play, we either want to take a random action, or figure out our current action based on our agent model, and finally we want to decay the epsilon value over the episodes until we're done decaying it.you can see the DQN algorithm at the code below.

Algorithm 1 Pseudo code for Reinforcement learning in self driving cars

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

3.7 Training scenarios

That is the most challenge part that takes much time, we faced many problems due to the limited of our computational capabilities until we decided to choose one scenario.

3.7.1 Description

First, we try to train our model on my local laptop using anaconda3, after installing the required packages. the model started training successfully but it stopped many times due to the restricted capability of GPU. We tried to use Google-Colab as it offers higher GPU, after installing CARLA simulator on Colab, we failed to train our model due to disconnection of GPU-runtime. Finally, we decided to train using anaconda on our local laptop and used checkpoints to complete training at the episode of the last training.

Also, we tried to train our agent in an environment that resembles the real world, we tried to spawn 100 different vehicles of various sizes ranging from motorbikes, cars, and truck .But the training stopped quickly due to collision with the spawned agents. Secondly, we tried to train the agent with environment

with dynamic conditions that alter the lighting conditions and weather conditions randomly over time. But again, the training failed as the RGB images from the front camera at bad conditions require extra image processing in order to give reasonable results, due to limited of our computational capabilities.

Finally, we decided to just keep the environment in normal bright conditions without any dynamic traffic to make things simple and reduce the computational complexity. When the agent was trained for 12,000 episodes in these conditions, we could see a decent enough model that was running very well on straight roads and also made decent left and right turn in the road when it was required occasionally.

3.8 Training using Xception model

We first train our model using Xception that is a good model for training self-driving cars. It is a model in Keras with71 hidden layers.

3.8.1 Description

We first trained our model with Xception model for 1000 episodes to see if it good for our training. when we examine the Xception model (with our speed layer), it has basically 23 million trainable parameters: And we were finding that our accuracy for the model was quite high (like 80-95%), so this tells me that we were almost certainly just overfitting every time. we cannot expect excellent agent that runs well, but in our case the agent was not even doing anything other than just circle around in the same.

3.8.2 Results

we can see that the accuracy value is about 95% which is higher than expected, the loss function has explosion due to explosion of Q values as the collision rewards is -200, we might have to decrease it to avoid such explosion .epsilon values decrease from 1 at the beginning to take random actions then it decreases over the episodes to allow the model to predict actions. we can judge the model form the average rewards that increase over the episodes, we can see from the test we can see that agent has learned to really just do 1 thing and to take one action because your Q values are actually static (model outputs same Q values no matter the input)

.

3.9 Training using CNN model

To make our model simpler, So, we used 64x4 convolutional neural network (CNN) that has 3 million trainable parameters in this case, so it seems better as there are less parameters to learn.

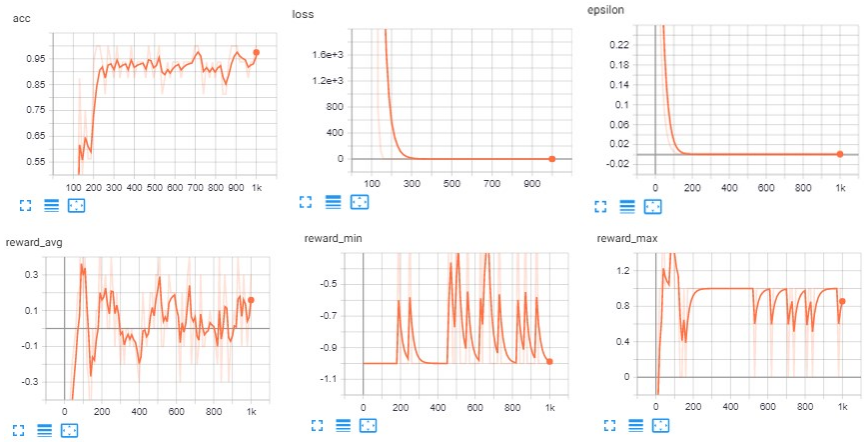


Figure 3: results of Xception model

3.9.1 Description

Our neural network consists of 4 hidden layers as show in the fig.4 it is simpler and help us to solve the problem of explosion in loss function and Q values. we trained the agent several times using the same model with checkpoints until we reach to 12000 episodes.You can see at figure our CNN that consists of 3 hidden layers that use average pooling ,flatten layer and output layer of 3 neurons to predict 3 actions.

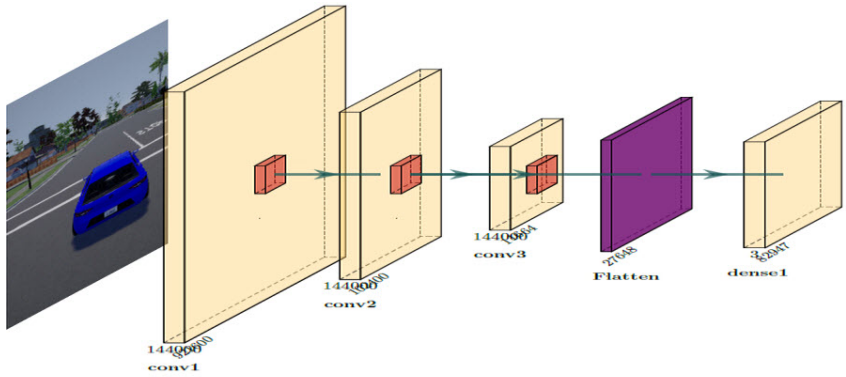


Figure 4: architecture of CNN

3.9.2 Results

From these graphs, we see that the accuracy for our model on an average was around a good 85%. Our loss function has no explosions.so, our model is im-

proved. Next, we see that the epsilon value fluctuates many times as the agent want to explore.

For rewards, we can see that the max reward over time does trend up until the 12,000 episodes, and may have continued back up given more time. The minimum reward (ie: the worst an agent did), didn't seem to change drastically. Finally, we can see that the reward average improved slowly, which meant that overall, the model did improve, and this model turned out to be our best performing model.

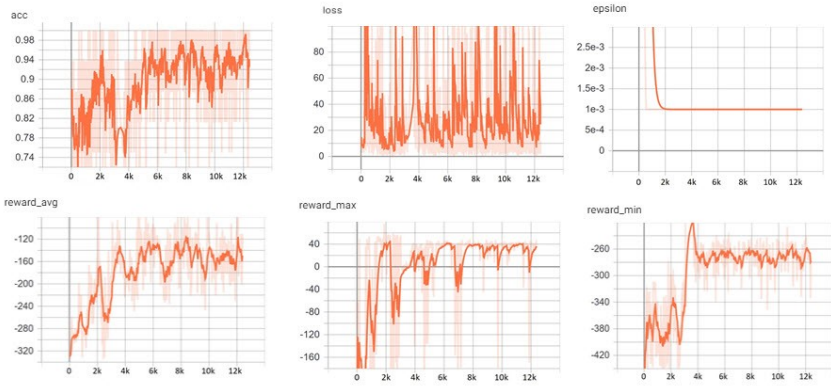


Figure 5: results of CNN model

3.10 Test our agent

After training our Xception and CNN models, we test our agent using anaconda3 on my local laptop to see if it is trained well or not.

3.10.1 Description

We tested our agent in CARLA simulator with the obtained trained models in an environment including different types of vehicles, we also uploaded the results of training and testing on Git-hub so, you can see them. From the test we noticed that the agent can drive successfully in straightforward but it finds problems when it turns right or left as we must train our agent much more episodes. Also, we may have to use other algorithms like A3C and DDPG as they are better in continuous actions. this work will be our tasks to implement in Diploma thesis2.

3.10.2 Results

From the test of our CNN model we reached to about 5.5 FPS , it is not enough due to not sufficient training episodes, during each step in the episodes we can see the model output three Q_values that represent three actions(turn_left-,straightforward,turn_right) and the agent takes the action that corresponding

to the maximum value of Q . so, from the figure below you can see the agent takes the action 1 (straightforward) that corresponding to the Q_max at the middle of the Q_values.

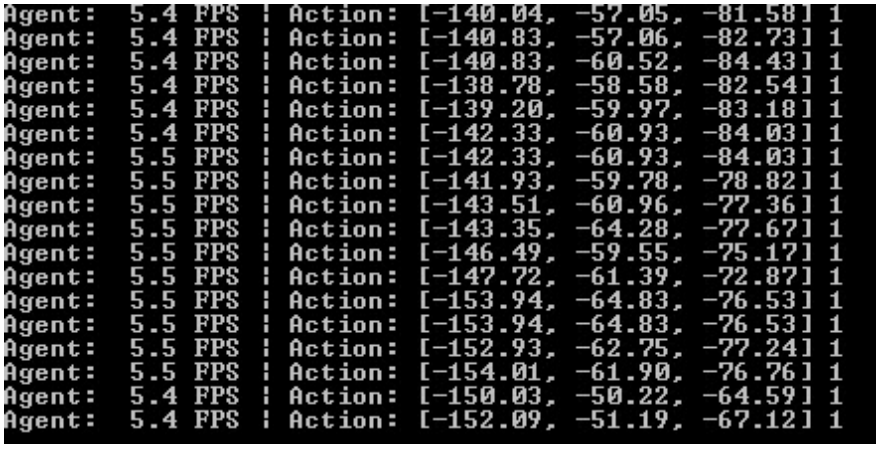


Figure 6: test of CNN _model

Acknowledgments

The author would like to express his thanks to Khalid Kahloot for directing the project and giving advice in case of problems.

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] 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.

[4] 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.

[5] Yang, Fan Wang, Ping Wang, Xinhong. (2018). Continuous Control in Car Simulator with Deep Reinforcement Learning. 566-570.10.1145/3297156.3297217.