REINFORCEMENT LEARNING PROJECT

SELF DRIVING CAR

Submitted in Fulfilment of Requirements
for the Degree of Master in Data Science
of the Northeastern University by

Ativeer Patni

Khoury College of Computer Sciences

August 25, 2021

Supervisor: **Prof. Nik Bear Brown, and**
Prof. Kylie Bemis

Table Of Contents:

Abstract

This project presents the implementation of a deep learning model to act as a self-driving car agent to map raw pixels from a single front-facing camera directly to steering commands. This end-to-end approach proved surprisingly powerful. With minimum training data from humans the system learns to drive in traffic on local roads with or without lane markings and on highways. It also operates in areas with unclear visual guidance such as in parking lots and on unpaved roads.
The system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. I never explicitly trained it to detect, for example, the outline of roads. Compared to explicit decomposition of the problem, such as lane marking detection, path planning, and control, the end-to-end system optimizes all processing steps simultaneously. We argue that this will eventually lead to better performance and smaller systems. Better performance will result because the internal components self-optimize to maximize overall system performance, instead of optimizing human-selected intermediate criteria, e. g., lane detection. Such criteria understandably are selected for ease of human interpretation which doesn't automatically guarantee maximum system performance. Smaller networks are possible because the system learns to solve the problem with the minimal number of processing steps.
The model was tested under three different traffic conditions to determine its performance statistically. The best model is the model with neural network configuration that approximates to the optimal Q-learning function. The source code of this project can be found on https://github.com/Ativeer/Self-Driving-Car-Simulation

1. Introduction

Driving a car requires precision and active attention. Any distraction to the driver could lead to accidents and losses. According to the data provided by Singapore Police Force, there were 94.34% of the road accidents led by human errors in 2016 which could be avoided, for instance, failing to keep proper lookout, failing to have proper control and change lanes without due care. Other road users are affected by the road accidents even though they follow the rules and regulations. Lately, drivers divert their attention to their smartphones to answer calls, send messages or navigate to their destinations. It increases the chances of road accidents when drivers are not focusing on the road condition. The automotive industry is innovating on the smart features of cars to further improve the safety and driving experience for users. The automatic braking system or collision avoidance system is one of the innovations that uses sensors such as radar to detect and mitigate collision.

The innovation in preventive safety mechanism catalyses the development process of autonomous vehicles that can drive to the destination autonomously and then reduce the number of road accidents that were caused by human errors by prioritising the safety of passengers and other road users including motorcyclists and pedestrians. Self-driving cars are the future development in the automotive industry.

The breakthrough in performance of parallel processing hardware and innovation in neural network design allows an artificial intelligence(AI) agent to control a vehicle autonomously. An AI agent is more capable of sustained attention and focus than humans. However, building an autonomous vehicle is a long-standing goal and complex task. Driving requires three main capabilities including recognition, prediction and planning[1].

First of all, an AI agent requires you to recognize the environment. Convolutional neural networks (CNN) are the most successful neural network model that is scalable and able to extract important spatial features from spatial data. Moreover, an AI agent has to predict the future states of the environment and perform best action at the moment. Furthermore, the planning component integrates the ability of the model to understand the environment (recognition) and its dynamics (prediction) to plan the future actions to avoid unwanted situations (penalties) and drives safely to its destination (rewards). The video feed from the camera and output of Light Detection and Ranging(LIDAR) are mainly used as the visual feed to AI agents

1.1 Background

An autonomous vehicle requires to carry out multiple tasks to drive on the road with dynamic traffic conditions. The tasks include following the steering wheel, road marking, keeping a safe distance with vehicles, reacting to emergency situations accordingly and navigating to the destination, etc. Udacity has partnered with Didi Chuxing to organize the first self-driving car challenge in 2017 to come up with the best way to detect obstacles using camera and LIDAR data. It could ensure self-driving cars prioritize the safety of road users and pedestrians. Besides that, Udacity has organized another challenge to predict the steering angle with deep learning based on the image inputs from a camera mounted to the windshield. However, the quality of the driving skill while most of the research and designs focus on the basic features of self-driving cars. The driving skill of a driving agent becomes the priority to deliver the best transportation service compared to a manned vehicle. For instance, a self-driving car agent should be capable of switching to the optimal lane and avoid any obstacle. It is important because passengers expect their ride to be free of jerks and sudden moments.

1.2 Objective

The objective of this project is to design and implement a self-driving car-agent to steer the vehicle properly through curves and avoid any obstacles. A simulation environment such as a steering wheel image is enough to show how much the wheel of the car will rotate. A neural network model is essential to build a self-driving car-agent that interacts with complex traffic conditions. Reinforcement learning technique is used to train the model to adapt to the dynamic environment with infinite action-state sequences. The agent is required to learn the optimal policy to choose the optimal available option which optimizes the curve of the steering wheel.

1.3 Scope

The scopes of this project are to optimize and match the actual steering wheel motion. The code is written in a Python environment and steering wheel motion is changed with the help of OpenCV libraries. The autonomous vehicle is assumed to function perfectly to steer along the road.

2. Literature Review

2.1 Reinforcement Learning

Reinforcement learning is one of the learning approaches for neural networks. Reinforcement learning implements differently compared to supervised learning and unsupervised learning. Reinforcement learning makes an agent to generate the optimal policy in an environment and maximises the reward [2]. An agent can learn the rewards given by taking action in each state and subsequently learn the proper action for each state without having any predefined rules and knowledge about the environment [3]. The agent learns by trial and error, or in other words, the actions with higher reward are reinforcement while actions with penalties are avoided in the future. It is different with supervised learning because supervised learning requires a huge number of labelled dataset to train the model [4]. The training process is repeated with the same set of data until the model converges. It requires efforts in labelling the data and the process is error-prone[5]. Reinforcement learning is also different from unsupervised learning because reinforcement learning aims to maximise the reward while unsupervised learning does not.

2.2 Markov Decision Process

Markov Decision Process (MDP) can make a sequence of decisions based on the utilities of the environment state. MDP [1] is defined by (S, A, R, $\mathbb{P}$, y), in which a set of possible environment states(S), a set of possible actions(A), discount factor(y), reward function (R), and state transition probability($\mathbb{P}$). Environment states S are sequences of states with initial state s0. Actions A are available actions for environment states S. Reward function R is a map of immediate reward given (state, action) pair. State transition probability $\mathbb{P}$ is the probability of transition from states S_0….S_i to stateS_{i+1}. However, there are infinite state sequences in the environment and calculation of transition probability for all state sequences is not feasible[6]. Thus, MDP follows Markov assumption in which the transition probability to state $S_{i+1}$is only depends on S_i and not previous states [7].

$$\mathbb{P}(S_{i+1}|S_i ,a_i, \ldots., S_0, a_0)\approx \mathbb{P}(S_{i+1}|S_i, a_i)$$

Discount factor (y) discounts the rewards of future states contributing to the cumulative reward [2]. The discounted future rewards can be formulated with the formula below.

$$R_t(s) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n = \sum_{t\geq 0} \gamma^t R_{a_t}(s_t, s_{t+1})$$

The value function can estimate the value of a state sequence start with state S, as shown follows:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t\right]$$

The optimal value function is the value function that gives the highest value for all states.

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

2.3 Q-Learning

Q-learning is a reinforcement learning mechanism that compare the expected utility of available actions given a state. Q-learning can train a model to find the optimal policy in any finite MDP [8]. The Bellman equation suggests the Q-value function[9] as shown in equation below:

$$Q(s,a) = r + \gamma \max_{a'} Q(s', a')$$

The difference of the Q-value function in equation 7 with the value function in equation 3 is that the Q-value function estimates the maximum future reward for taking action a given a state *S*. The calculation of the Q-value function takes into account the maximum discounted future reward for an agent to move from state *S* to state *S′*. The Q-value function can be iteratively converged to the optimal Q-value function [10] by the difference of the estimated utilities between current state S_t, and next state S_{t+1}, with learning rate α, at time step unit *t*, as shown below:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a} Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

Then, Q-learning function can substitute the value function, as shown below

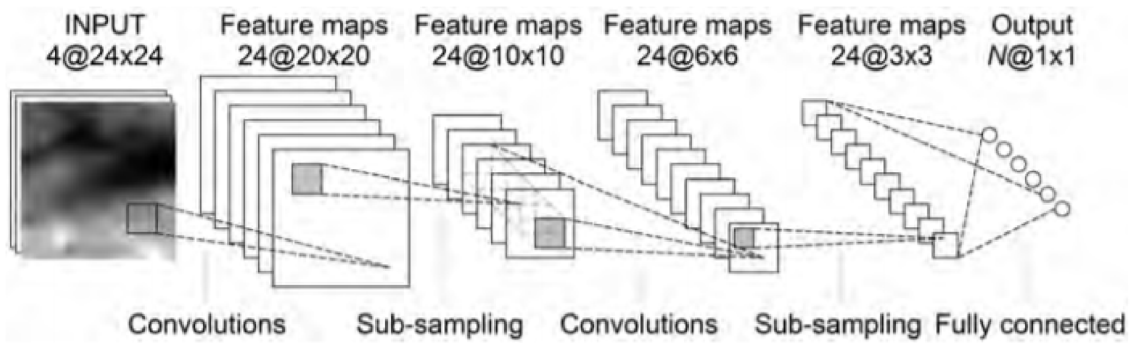
$$V^*(s) = \max_a Q^*(s,a)$$

The optimal policy can be retrieved from the optimal value function with

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

2.4 Convolutional Neural Networks

The efficiency and learning ability of the brain have inspired the innovation of the convolutional neural networks (CNN). The ability of animals to perceive features from complex visual maps and generate responses has biologically guided the construction of CNN [11]. The local correlation can be extracted spatially by allowing a local connectivity pattern between neighbouring cells of adjacent layers. Figure below shows that each filter is applied across the entire input map and generates a feature map. The output of the feature map is connected to the adjacent layer with features extracted. Each feature map explores unique features regardless of the location across the visual field.



2.5 Deep Q-Learning Networks (DQN)

Neural network is utilized as the function approximator of the Q-learning algorithm to form the Deep Q-learning network (DQN). The learning goal of DQN is to find the best settings of the parameter $\theta$of $Q(S, a; \theta)$ or the weights w of the function approximator. The objective of DQN is to minimise the Mean Square Error (MSE) of the Q-values. Besides that, experience replay technique uses first-in-first-out(FIFO) method to keep experience tuples in replay memory for every time step. The replay memory stores experience tuples for several episodes to ensure that the memory holds diversified experiences for different states. The experience tuples are sampled from replay memory randomly during Q-learning updates. The implementation of experience replay can remove correlations in the observation sequence and smoothing over changes in the data distribution by randomizing over the data. In practice, the most recent N experience records are sampled uniformly and randomly from replay memory. The random sampling gives equal probability to all experience tuples.

3. Data Collection

This data is collected by Udacity and is open sourced. 223GB of image frames and log data from 70 minutes of driving in Mountain View on two separate days, with one day being sunny, and the other overcast. *Note: Along with an image frame from the cameras, they also include **latitude, longitude, gear, brake, throttle, steering angles** and **speed**.*
More information related to the dataset can be found here.

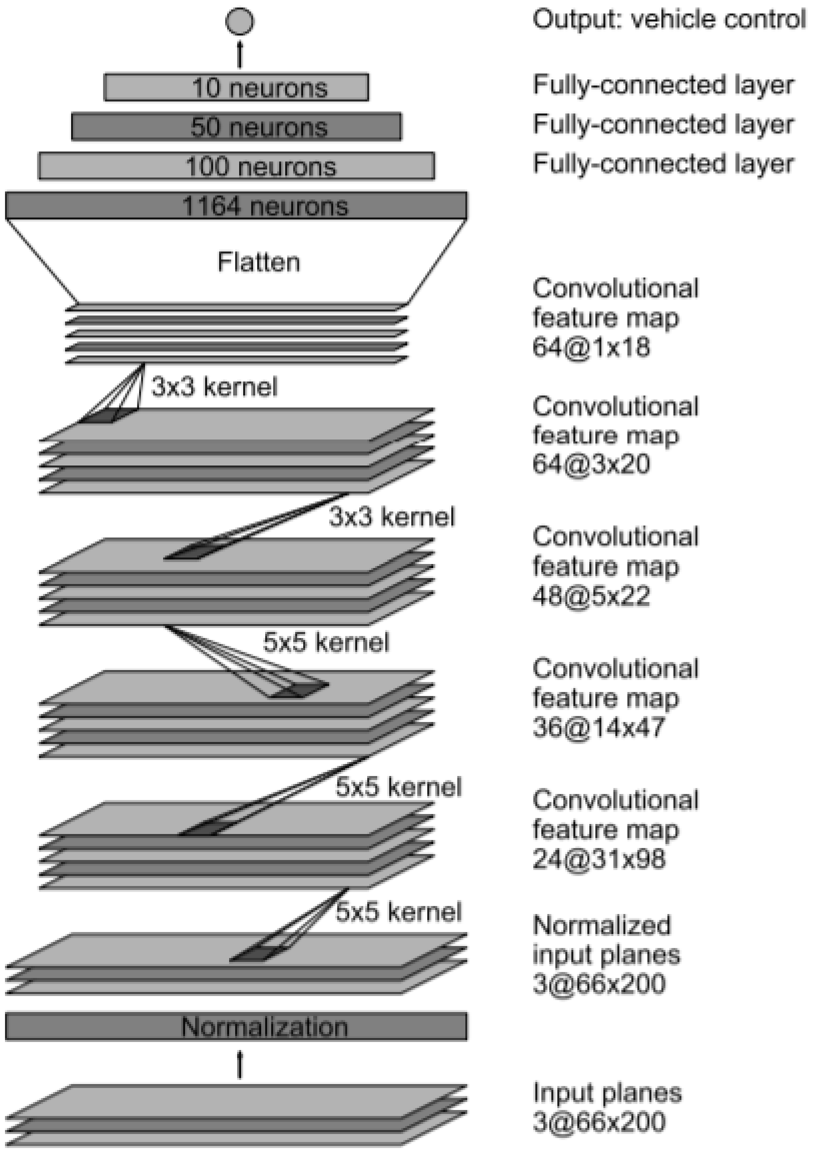
4 Network Architecture

I trained the weights of the network to minimize the mean squared error between the steering command output by the network and the command of either the human driver, or the adjusted steering command for off-center and rotated images. The network architecture is shown in Section 4. The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network. The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing. The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. We use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers. We follow the five convolutional layers with three fully connected layers leading to an output control value which is the inverse turning radius. The fully connected layers are designed to function as a controller for steering, but we note that by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractor and which serve as controller.

5 Training Details

5.1 Data Selection

The first step to training a neural network is selecting the frames to use. The collected data is labeled with road type,and the driver's activity (staying in a lane, switching lanes, turning, and so forth). To train a CNN to do lane following we only select data where the driver was staying in a lane and discard the rest. I then sample that video at 10 FPS. A higher sampling rate would result in including images that are highly similar and thus not provide much useful information. To remove a bias towards driving straight the training data includes a higher proportion of frames that represent road curves.



5.2 Augmentation

After selecting the final set of frames I augment the data by adding artificial shifts and rotations to teach the network how to recover from a poor position or orientation. The magnitude of these perturbations is chosen randomly from a normal distribution. The distribution has zero mean, and the standard deviation is twice the standard deviation that was measured with human drivers. Artificially augmenting the data does add undesirable artifacts as the magnitude increases.

6. Experiments

6.1 Experimental Setup

The experiment is carried out in a simulation program as shown in Figure below. The dataset from Udacity is too huge for a small process to work with, thus I used a small 25 minute video (over 45,000 images) and it was divided into training and testing sections (80:20 split). Other Hyperparameters are provided in the table below. The setup was done to train on around 19 minute of video and test it on the remaining 6 minutes. The output is determined by the degree in which the steering wheel was rotated in the simulated output vs the actual angle the steering wheel was rotated. The network, as explained earlier, uses atan (TAN) activation function. The first idea is to check what a linear model (mean model) would determine. Then compare it later with what the complex model will determine.

6.2 Hyperparameters

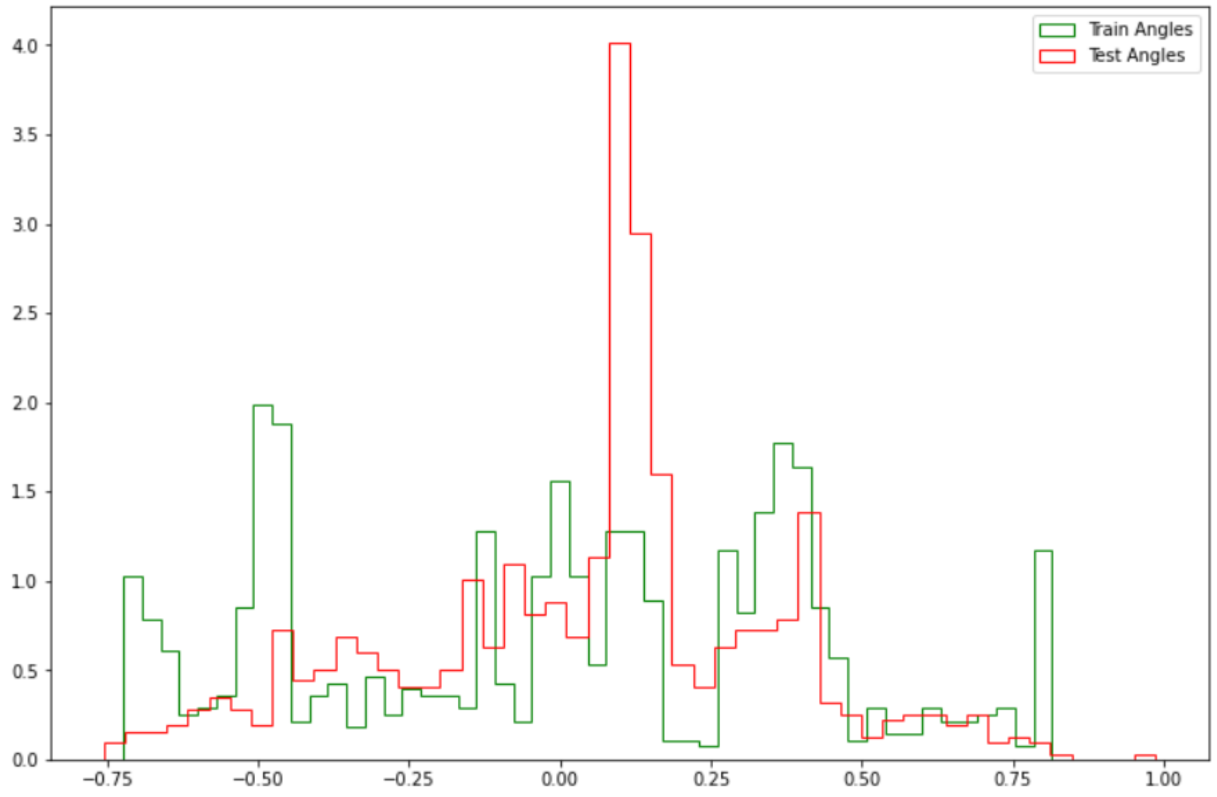
The Hyperparameters were selected based on the values provided by the NVIDIA paper found here : https://arxiv.org/pdf/1604.07316.pdf%5D

- The people working at NVIDIA tested with variety of models and found this to work the best for the DQN model.
- This practice could take days to just train for one particular set of hyperparameters, thus it is better to use just the set of input to check if it is valid for a DQN as well.

7. Results

The results are a comparison of the test angles of the steering wheels vs the actual angle by which the steering wheel is rotated. The angles are trained first on a mean model then on DQN. The results are as follows:
MSE with mean model : 0.19 radians
MSE for DQN : 0.05 radians.

The graph below also shows a more in depth idea about the test angles vs actual angles. Its the graph of steer angle v/s the average count



The graph shows that the model was able to learn when steer, although, not rotate enough for a few extremes. Thus the model needs to be more robust and needs more training.
The Output video can be found here.

8. Conclusions

I have empirically demonstrated that CNNs are able to learn the entire task of lane and road following without manual decomposition into road or lane marking detection, semantic abstraction, path planning, and control. A small amount of training data from less than a hundred hours of driving was sufficient to train the car to operate in diverse conditions, on highways, local and residential roads in sunny, cloudy, and rainy conditions. The CNN is able to learn meaningful road features from a very sparse training signal (steering alone). The system learns for example to detect the outline of a road without the need of explicit labels during training. More work is needed to improve the robustness of the network, to find methods to verify the robustness, and to improve visualization of the network-internal processing steps.

Thus, by using a variant of online Q-learning that combines stochastic minibatch updates with experience replay memory to ease the training of deep networks for RL. A small amount of training data from less than a hundred hours of driving was sufficient to train the car to operate in diverse conditions, on highways, local and residential roads in sunny, cloudy, and rainy conditions.

The CNN is able to learn meaningful road features from a very sparse training signal (steering alone). The system learns for example to detect the outline of a road without the need of explicit labels during training. More work is needed to improve the robustness of the network, to find methods to verify the robustness, and to improve visualization of the network-internal processing steps.

References

1. Sallab, A.E., Abdou, M., Perot, E., and Yogamani, S.: ‘Deep reinforcement learning framework for autonomous driving’, Electronic Imaging, 2017, 2017, (19), pp. 70-76

2. Littman, M.L.: ‘Markov games as a framework for multi-agent reinforcement learning’: ‘Machine Learning Proceedings 1994’ (Elsevier, 1994), pp. 157-163

3. Mahadevan, S., and Connell, J.: ‘Automatic programming of behavior-based robots using reinforcement learning’, Artificial intelligence, 1992, 55, (2-3), pp. 311-365

4. Srivastava, N., Mansimov, E., and Salakhudinov, R.: ‘Unsupervised learning of video representations using lstms’, in Editor (Ed.)^(Eds.): ‘Book Unsupervised learning of video representations using lstms’ (2015, edn.), pp. 843-852

5. Donmez, P., Carbonell, J.G., and Schneider, J.: ‘Efficiently learning the accuracy of labeling sources for selective sampling’, in Editor (Ed.)^(Eds.): ‘Book Efficiently learning the accuracy of labeling sources for selective sampling’ (ACM, 2009, edn.), pp. 259-268

6. Ng, A.Y.: ‘Shaping and policy search in reinforcement learning’, University of California, Berkeley, 2003

7. Hermanns, H.: ‘Markov Chains’: ‘Interactive Markov Chains’ (Springer, 2002), pp. 35-55

8. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M.: ‘Playing atari with deep reinforcement learning’, arXiv preprint arXiv:1312.5602, 2013

9. Sutton, R.S.: ‘Learning to predict by the methods of temporal differences’, Machine learning, 1988, 3, (1), pp. 9-44

10. Boyan, J.A., and Moore, A.W.: ‘Generalization in reinforcement learning: Safely approximating the value function’, in Editor (Ed.)^(Eds.): ‘Book Generalization in reinforcement learning: Safely approximating the value function’ (1995, edn.), pp. 369-376

11. LeCun, Y., Kavukcuoglu, K., and Farabet, C.: ‘Convolutional networks and applications in vision’, in Editor (Ed.)^(Eds.): ‘Book Convolutional networks and applications in vision’ (IEEE, 2010, edn.), pp. 253-256