SELF-DRIVING CAR

PROJECT REPORT FOR CS354

Presented by

Kunal Gupta (150001015)

Bhor Verma (150001005)

Computer Science and Engineering

3rd Year

Under the Guidance of

Dr. Aruna Tiwari

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Spring 2018

CONTENTS

INTRODUCTION

Just over a decade ago, the idea of being driven around by a string of zeros and ones was ridiculous to pretty much everybody. But now after Tesla, Google, Uber and the like started research in this area, the technology reached a point where no automaker could ignore it. It has now become a trending area of research to improve the driverless scenario in the automobile industry. Companies like Ford, General Motors, Nissan, Tesla, Mercedes, and the rest have started pouring billions into their own R&D.

Autonomous cars use a variety of techniques to detect their surroundings, such as radar, laser light, GPS, odometry and computer vision. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage.

But, the real job is to endlessly improve the software, which is powered by machine learning, used by those computers to correctly interpret the data from all those sensors.

MOTIVATION

Nearly 1.3 million people die in road crashes each year, on average 3,287 deaths a day. An additional 20-50 million are injured or disabled.

Self-driving cars can avoid the major causes of such accidents:

- **Distraction:** Self-driving cars are dedicated to driving and can notice more, from all angles, and react more quickly. Anything that distracts a human such as text messages or calls or conversations cannot bother a computer system.
- **Speeding:** Speeding due to human emotions such as frustration, panic, excitement, thrill etc., are also have no effect on a machine learning algorithm trained to stay focused on the rules while following all traffic rules and signs promptly.
- **Drunk Driving:** Computers (currently) do not have the ability to get drunk, so this cause is also struck off.
- **Recklessness:** Unlike, computer systems are much better at risk detection and do not take risks just for saving time.

Apart from this, self-driving cars have benefits of their own. These include reduced mobility and infrastructure costs, increased safety, increased mobility, increased customer satisfaction and reduced crime. Specifically they cause a significant reduction in traffic collisions; the resulting injuries; and related costs,

including less need for insurance. Also they can increase the traffic flow due to all cars being driven by a set of algorithms, evolving, obeying the perfect rules and maybe even communicating with each other to make decisions.

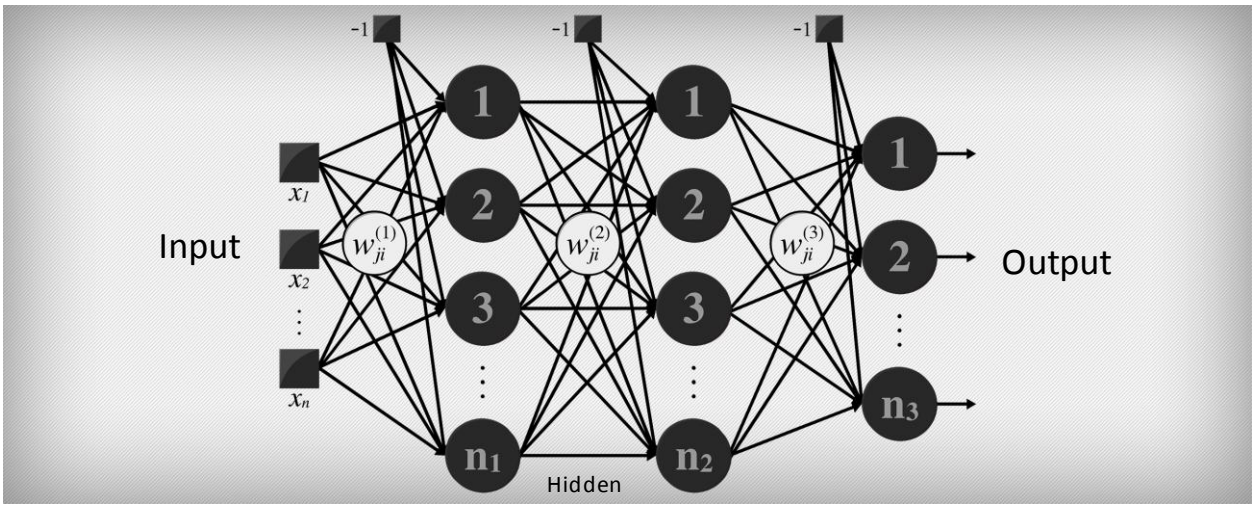
PROBLEM DEFINITION AND OBJECTIVES

The key objective is to design a scaled down version of self-driving system using an electronic car built using a Raspberry Pi, open source software and other electronic equipment (as required). A camera is the only sensor that provides information of the environment around the car.

The car should be able to navigate a custom built track itself. We use Multi-Layer Perceptron Model with back-propagation, a supervised-learning based algorithm, for the same.

STUDY OF THE ALGORITHM

Multi-Layer Perceptron consists of the input layer, output layer, and one or more hidden layers. Each layer of MLP includes one or more perceptrons directionally linked with the neurons from the previous and the next layer.



The outputs to the next layer are obtained by multiplying the weights with the respective inputs of the current layer, and thus the network is feed-forward.

BACKPROPAGATION

Learning occurs in the model by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result, and thus is rightly called **back**-propagation.

ANALYSIS AND DESIGN

The car consists of required electronic apparatus (motors, breadboard, etc.) along with a Raspberry Pi 3 mounted on top and a Camera on the front. The electronics at the bottom are used to power and drive the car. The Camera is used for receiving the video feed to get the driver's perspective as input. The Raspberry Pi is used for sending control signals to the car as well as receiving inputs from the Camera and doing the required computation.

DATA COLLECTION AND PREPROCESSING

Since the goal of the project is to build a machine learning model for a self-driving car, the inputs need to be in the form of visual perspective of a manual car driver and the correct steering direction for learning.

IMAGES AS INPUTS TO THE MODEL

The visual information is in the form of video coming as input from the camera. The video is converted into images from its frames. To simply things, we scale down the video into a 200×200 pixels frame. Since driving direction would only depend on the track in front of the car, we crop and only retain the lower half of the frame to reduce dataset size. Further, we convert the frames to gray scale as the colour information will be complex and unnecessary.

But neural networks take only numerical value as inputs and provide numerical value as outputs. Thus, comes the task of representing the images in the suitable way so they can be utilized as inputs into the neural network.

The information in the images is actually every pixel's intensity value ranging from 0 to 255. Since the determining factor for the steer direction is actually information in the image, we can use these intensity values for each pixel as

inputs to the model. To further, better out the inputs, we normalize the inputs to the range [0, 1] by dividing each pixel's intensity by a factor of 255.

All this processing reduces the inputs to the model to a very small and feasible size for learning.

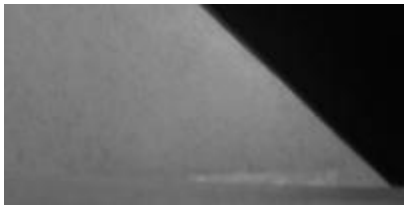
CORRECT STEER DIRECTIONS AS TARGET OUTPUTS TO THE MODEL

Since we are following a supervised-learning based algorithm, we require the correct target outputs for carrying out our training of the model.

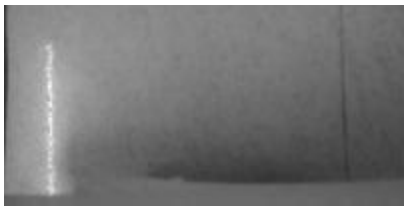
Obtaining the correct steering direction is similar as to how one learns by looking at other people drive: correlating the visual inputs to the steer direction.

This is done by designing a program to drive the car on the track based on user input, via the keyboard. The training data (video frames) is paired with the human input as target outputs and is saved.

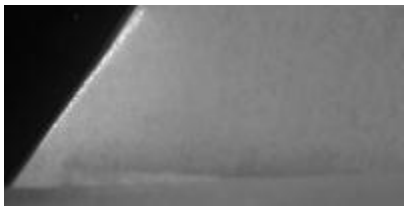
The steering direction is in the form of directions such as LEFT/RIGHT/FRONT and needs to be in a format (that is again real numbers) to be understood by the model. We assign values 0, 1 and 2 to LEFT, FRONT and RIGHT steering directions respectively.



Left



Front



Right

APPLICATION AND IMPLEMENTATION OF ALGORITHM

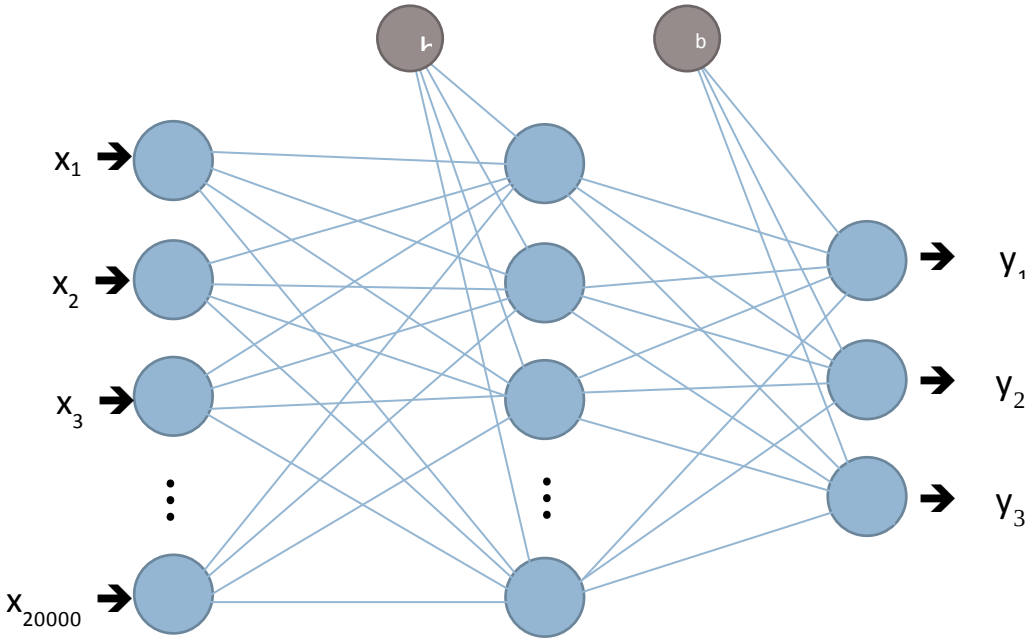
After the preprocessing we obtain $200 \times 200 \times \frac{1}{2} = 20000$ as the input layer size and 3 as the output layer size (each neuron representing one of LEFT, FRONT and RIGHT and will output 1 for it being the steering direction and 0 if not):

$$\begin{matrix} 1 \\ 0 \\ 0 \end{matrix} = LEFT, \quad \begin{matrix} 0 \\ 1 \\ 0 \end{matrix} = FRONT, \quad \begin{matrix} 0 \\ 0 \\ 1 \end{matrix} = RIGHT$$

After multiple trials and errors, we obtained 40 as a good number of neurons in the hidden layer as it provided a good accuracy.

Thus we propose a 3 layered model with:

- 20,000 inputs
- 40 as the hidden layer size.
- 3 outputs



The collected data is trained on this model using Back-Propagation.

CONVERSION TO MATRIX FORM

One of the basic methods one can write the code for a neural network is by implementing a perceptron and then instantiating it multiple times so as to get a neural network. But this method is very slow and inefficient and can use a large amount of memory.

Thus we need a faster way for the implementation. We convert the methods used in back-propagation into their respective matrix form to be used in Python's NumPy (compiled with BLAS enabled).

Considering a neural network with 4 Layers, and f_i is the activation function used in the i^{th} layer.

The forward propagation can be derived rather easily and is very intuitive:

$$Input = x_0$$
$$Hidden\ Layer1\ output = x_1 = f_1(W_1 x_0)$$
$$Hidden\ Layer2\ output = x_2 = f_2(W_2 x_1)$$
$$Output = x_3 = f_3(W_3 x_2)$$

Stochastic update loss function $E = \frac{1}{2}\left|\left|z - t\right|\right|_2^2$, where *t* is the target output and *z* is the output of the ANN.

The weight update function is $w = w - \eta\frac{\partial E}{\partial w}$ for all the weights *w*.

Back-propagation equations can be derived by repeatedly applying the chain rule.

First we derive for the weights in W_3:

$$\frac{\partial E}{\partial W_3} = (x_3 - t)\frac{\partial x_3}{\partial W_3}$$
$$= \left[(x_3 - t) \circ f_3'(W_3 x_2)\right]\frac{\partial W_3 x_2}{\partial W_3}$$
$$= \left[(x_3 - t) \circ f_3'(W_3 x_2)\right] x_2^T$$
$$Let\ \delta_3 = (x_3 - t) \circ f_3'(W_3 x_2)$$
$$\frac{\partial E}{\partial W_3} = \delta_3 x_2^T$$

Here $\circ$ is the Hamdard product (multiplication of corresponding elements in the matrices).

Next for the weights in W_2:

$$\frac{\partial E}{\partial W_2} = (x_3 - t)\frac{\partial x_3}{\partial W_2}$$

$$= \left[(x_3 - t) \circ f'_3(W_3 x_2)\right]\frac{\partial W_3 x_2}{\partial W_2}$$

$$= W_3^T \delta_3 \frac{\partial x_2}{\partial W_2}$$

$$= [W_3^T \delta_3 \circ f'_2(W_2 x_1)\frac{\partial W_2 x_1}{\partial W_2}$$

$$= \delta_2 x_1^T$$

And Similarly for W_1:

$$\frac{\partial E}{\partial W_1} = [W_2^T \delta_2 \circ f'_1(W_1 x_0) x_0^T$$

$$= \delta_1 x_0^T$$

Thus for forward pass:

$$x_i = f_i(W_i x_{i-1})$$

$$E = \frac{1}{2}\left||z - t|\right|_2^2$$

And backward pass:

$$\delta_L = (x_L - t) \circ f'_L(W_L x_{L-1})$$

$$\delta_i = [W_{i+1}^T \delta_{i+1} \circ f'_i(W_i x_{i-1})$$

And finally the weight updates:

$$\frac{\partial E}{\partial W_i} = \delta_i x_{i-1}^T$$

$$W_i = W_i - \eta\frac{\partial E}{\partial W_i}$$

TRAINING AND TESTING

We manually collected around 3,000 images taken from the car's camera consisting of roughly 1,000 each of all possible steer directions – Left, Forward and Right.

This data set was converted to gray scale, intensity values extracted into matrices and normalized.

After splitting it into a ratio of 80:20 labeled as training and test data (to prevent memorization), we fed the training data into the ANN along with the target labels for training. This gave us an accuracy of nearly 99% sometimes even reaching 100%. The test data's labels were then predicted which mostly resulted in near 100% accuracy.

Now the trained model was transferred to the onboard RPi on the car and it was left to navigate a course.

PERFORMANCE MEASUREMENT

The performance measurement criteria we used, was how well the car performs on the track. For a total of 50 trials, the car crashed into the footpath only once.

Next we placed the car in different rooms to test for the same. This time if the lighting conditions of the room were significantly different from that of the training room, the car crashed much frequently. Also it performed poorly on glossy floor which reflected a lot of light, whereas performed quite well on the rough non-reflecting platforms.

CONCLUSION

The model performs well for similar lighting conditions but poor for lighting conditions that differ significantly, thus requiring much diverse input training data. The glossy surfaces play a challenge to the model due to high reflectivity and again requiring diverse input data for them to be navigated. Also more number of sensors are required for better performance of the car.