DEEP LEARNING FOR SELF DRIVING CARS

STRATO-IT

BY

SUDARSHAN KAMATH

KAUSTUBH VYAS

ANURAG SONTHALIA

ABSTRACT

Deep Learning has often been used in self driving vehicles on the roads as well as in the industries. Image classification benchmarks have been established quite popularly using neural networks. We realize the underlying potential of neural networks and hence aim to create a line following RC car using deep learning techniques. We shall be using the video obtained through an FPV camera for input. There are various approaches to this problem. We shall be going through those, while justifying why we approached the problem using a simple fully connected neural network. We would also be describing the difficulties that we made and the mistakes that you could make and hence be on a lookout for. In the end, we would be describing the results and would also provide a few points for improvement over our model.

METHODOLOGY

We have used the following approach to this problem in order to gain an optimum solution :

1. Try and achieve complete control over the RC car's throttle and steer through a command line interface or PC control over WiFi using an Arduino board.
2. Attach camera on the car at a suitable position on the car and code it to run simultaneously with the car with minimal lag.
3. Design a track in a way such that it doesn't challenge the limitations of the car.
4. Use the camera and the car to gain training data for the ML/DL model by driving it ourselves.
5. Find potential models which can be trained with the help of this data.
6. Train model and then test it.
7. Find scopes for improvement and try and improve accuracy of model.
8. If step 6 does not work, then try another model.
9. The above steps 4 to 7 are repeated until we get a model which gives a proper working for the data we have with least deviation from the followed path.

We would be describing each of these steps in detail throughout this document.

HARDWARE

We have used the following hardware for the project :

1. Arduino Uno x 1
2. RC car
3. FPV camera
4. Laptop (for data collection and commands)

5. Wifi Shield for Arduino JSN270
6. Connecting wires
7. M to F / M to M / F to F converter wires
8. Li-Po Batteries
9. Oscilloscope
10. Brown board
11. Connecting Pins
12. Soldering apparatus
13. Tapes, other equipments (screwdrivers, glue, etc)

SOFTWARE AND GPU

The entire code ran on a Intel-Core i5-8400 CPU with clock speed of 2.80Ghz with 6 cores. The GPU is a Nvidia GeForce 1050 Ti. We used UBUNTU Debian 16.04 operating system. The codebase was entirely written in python 2.7.13 installed with the help of anaconda version 4.4.0. OpenCV version 2.4.11. CUDA v8.0.61 for python2, 64bit was used.

We had to later switch to Windows 10, python 3 and OpenCV 3 due to some NVIDIA driver errors causes boot time problems in UBUNTU.

Sketch was used to write the code for arduino. Computing package like numpy was also used. We will mention about the other packages as we encounter them during the description of the code. Do check the references in the end for the complete code.

SKETCH CODE :

Sketchbook was used to make the code which ran the arduino. The JSN270 wifi shield provided connectivity with the laptop. A client code was written which first connected to the WiFi and then searched for a server with a given IP address. A web server(Listener) was started on the laptop/PC with the same IP address and PORT number. Once connection was established, we could give control commands from the command line to the arduino.

Initially an attempt was made to connect with the PC using bluetooth. However connectivity was sketchy and not stable. We used a Jarduino with an inbuilt bluetooth module for the same. We received a lot of garbage values in the bluetooth buffer and a lag in command sending and receiving. Hence we decided to reject it.

There was a radio receiver attached to the RC car, which received signals from the remote control and then sent PWM signals to the on-board controller of the car. We removed this radio receiver and used the arduino as a receiver of command line signals. The arduino would then interpret this signals and provide the suitable PWM signals to the steer and throttle control motors.

The required values of the PWM signals were first measured with the help of an oscilloscope attached to the radio receiver.

The remote control provided an almost continuous range of steer angles and throttle speeds, including the ability to reverse. We decided that we did not require such excessive functionalities in our car for this project. Hence we made it possible to iterate back and forth over a few discrete steer angle values which provided us enough control over the car to make it possible to manually maneuver the car over our track.

Based on your requirements you may modify this code slightly to provide more or less control. We would be commenting the code as much as possible so as to make it easy to read and understand.

IMPROVEMENTS REQUIRED :

- Initial connection establishment takes some time(upto a minute).
- First start the web server application and make sure to RESET the arduino every time before you start it.
- On starting, the module picks up some garbage values and hence a few power impulses are picked up by the throttle motor(3-4 times). So be sure to hold the car and lift the hind wheels till connection is established.
- When driving, sometimes the throttle gets unpredictable impulses throwing the car off the track.

PYTHON CODE FOR DATA COLLECTION :

The python code simulates a TCP web server for the client to connect to at the IP address and host port mentioned in the code. We make sure the car and the arduino are connected to the same wifi connection. This code also simultaneously captures the video input from the camera. After establishing connection, it waits for user commands. We have given multiple control keys for the user to increase/decrease throttle as well as to control steering angles.

The steer angles are written into a CSV file along with the input image converted from a 240*320*3 input image array to a row of integers in a CSV file where the first column of each row corresponds to the steer command whilst the rest of the 230400 columns each contain an integer representing the pixel value of the image array between 0 and 255. The size of the image was decided intuitively. We have resized the image to half the size to reduce the dimensionality of the feature vectors. Since the image still retains a decent quality, we do not think that this would reduce or effect the accuracy of the network in any way.

Later these pixel values are processed and used as features into the neural network whilst the first column values are used as their corresponding labels.

The commands correspond to an increment or decrement in steer angles and throttle values over discrete steps. We have not allowed the user to provide absolute angle and speed commands to the car for ease of usage.

The CSV file is written over whenever a user steer angle input in provided. We have also ensured that the user is unable to provide steer commands to the wheel after achieving the

maximum physically possible steer angles. Currently the lag is almost negligible and the commands are executed instateously.

Currently no improvements are required for this code.

THE TRACK, THE CAMERA AND THE ENVIRONMENT :

These are a few important decisions we had to make. Setting up the track such that it is not too simple for the car but also not beyond its physical capability to maneuver required of us to test out the car manually on different tracks. For this we set the car at a very low speed and tried to understand the smallest/sharpest turns the car could make. Then the track was made with the help of a yellow tape on a green floor. We avoided carpeted floors as the motor had to be set to a high torque value to overcome the static friction and would then pick up high speeds when friction turns kinetic.

The environment was not given a lot of importance as we believe that after achieving our first set of targets and training the network successfully, after some data processing we would be able to feed images from other environments and tracks into our neural net and still achieve a high accuracy. If we are able to achieve this, then we can think of targeting benchmarks.

The camera was placed in front of the car at an angle tiled a bit towards the ground from the horizontal such that the images capture a decent part of the tracks in front and avoids objects which are far away.

DATA COLLECTION :

Our first target was set at making a smart car with the capability to maneuver itself over the same track on which we trained it. We did multiple rounds of data collection.

Trying to drive the car around manually was quite difficult at first, but we made the keyboard keys set to A,S,W,D so that we can control the car just like we controlled a car while playing a video game. We also reduced the speed drastically to improve user control.

We were able to maneuver the car manually. But these attempts were futile as we realised that we could create much better quality data by simply holding the car at different positions and intuitively predicting the angles it would take now. We followed one thumb rule. If the camera was on the line in the image then we had to go straight, if the camera was to the left, then we take a right and if the camera was to the right then we take a left. We collected around 4 GB of data in this manner.

DATA PROCESSING :

The data was simply processed to bring the standard deviation of the features to 1 and the mean to 0 using scikit learn.

THE NEURAL NET :

There are multiple approaches to this problem. In an ideal situation, we should use a RNN in order to store temporal features. However, given the short time frame of the project and keeping in mind the slow speed of the car and other hardware limitations, we decided to go with a general feedforward neural network. Currently we have decided to try a simple two layered neural net. We referred to a project by Ryan Zotti, in which he had used a neural net with a single hidden layer to run his car. We decided to try out a neural network with two hidden layers since our task was a bit more complex and also a two layered network can theoretically fit any non-linear function. So we are currently using a neural network with 300, 30 units each in two hidden layers with ReLU activation functions. We have also implemented drop out in order to prevent overfitting. We are still experimenting with the model and trying to find out which one would produce the best results. The aim is to make the model an extremely simple yet efficient one.

All the details of the hyperparameters would be commented in the code. Currently we are achieving an accuracy of above 99% for both or test and train set. However we believe that this might be due to the model overfitting over the test set, since it is a bit similar to the train set.

We would be getting more insights and making further corrections after we run our first tests.

END OF PART 1