

Udacity Self-Driving Car Engineer Nanodegree

Behavioral Cloning Project

-By Avni Sharma

Introduction:

The goals / steps of this project are the following:

1. Use the simulator to collect data of good driving behavior
2. Build, a convolution neural network in Keras that predicts steering angles from images
3. Train and validate the model with a training and validation set
4. Test that the model successfully drives around track one without leaving the road
5. Summarize the results with a written report

Files Submitted & Code Quality

Submission includes all required files and can be used to run the simulator in autonomous mode. My project includes the following files:

1. model.py containing the script to create and train the model
2. model.ipynb python notebook containing the code to create and train the model
3. drive.py for driving the car in autonomous mode
4. model.h5 containing a trained convolution neural network • writeup_report.md summarizing the results

Submission includes functional code. Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing `python drive.py model.h5`.

Submission code is usable and readable. The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

An appropriate model architecture has been employed. I decided to test the model provided by NVIDIA as suggested by Udacity. I used the Keras lambda layer to normalize the data. The data was cropped to only reflect the road region. Below is the network architecture:

Attempts to reduce overfitting in the model The model contains a dropout layer after the first fully connected layer with a dropout rate of 25%.

Model parameter tuning

No. of epochs = 5

Learning rate = 0.001

Generator batch size = 32

Optimizer = Adam

Appropriate training data : The data set was split in a 85%-15% ratio. The data was flipped and augmented in order to familiarize the network about moving in the counterclockwise direction too. The corresponding steering angles were hence negated. A correction factor of 0.2 in the steering angles was introduced. Hence, the left steering angle was increased by 0.2 and right was decreased by 0.2.

Model Architecture and Training Strategy

1. Solution Design Approach :

My first step was to use a convolution neural network model similar to the one provided by NVIDIA autonomous team.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting. To combat the overfitting, I modified the model to incorporate a drop out layer with the drop out rate of 25% after the first fully connected. This helped my case and so no further actions were required. The final step was to run the simulator to see how well the car was driving around track one. With the image pre-processing, training set augmentation and measures to reduce overfitting already done, my testing worked fine. I tuned my steering angle correction to 0.2 as suggested by the lecture videos. By the end the vehicle was able to drive autonomously on track one.

2. Final Model Architecture :

The final model architecture consisted of a convolution neural network suggested by NVIDIA team but an additional drop out layer.

I used the ELU activation instead of RELU as I found they worked better in my architecture.

3. Creation of the Training Set & Training Process I tried to take the training data as per suggestions given in the classroom but being bad at maneuvering the car properly and not obtaining correct results, I decided to go for the sample data set given by Udacity