Behavioral Cloning: Predicting Steering Angles from Camera Images

This project aims to mimic the human behaviour in driving a car in a simulator. Training data is collected by running the simulator in training mode to drive the car around the simulated track for a few laps. The training data comprised of sequence of images of a simulated camera put on the front center of the dashboard of the car. Additional cameras (front left and right) can be used although not necessary to this project. The steering angles and the thottle and with their corresponding image filenames are recorded in a csv file.

During training, the car can be drived using the keyboard or the mouse, as in a video game.

Here is a video demo showing how the trained AI agent drives the simulated car in a fully autonomous mode:

https://www.youtube.com/watch?v=Dhr72ryP0Co

How to Run

First, record your driving in the simulator for few laps. The simulator will save the image snapshots and the csv data that will be used to train the neural network. Edit the 'model-AWS.ipynb' file to include the local path of the saved images and csv file path.

Run the 'model-AWS.ipynb' to train the deep neural network.

Put the saved model of the deep neural network named 'model.h5' in the same folder of drive.py. This will be used to clone your driving behavious and control the simulator to drive the car autonomously.

python drive.py model.h5

and if you want to record the autonomous driving into a video:

python drive.py model.h5 run1

The snapshots of the autonomous driving will be recorded in the run1 folder. To create the video:

python video.py run1

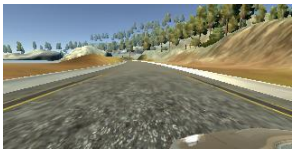
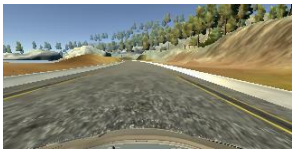
Data Collection, Augmentation and Preprocessing

For this project I have used the training data of Ryan Moore:

https://nd013.s3.amazonaws.com/track1.zip

It consisted of a few laps and few recovery recording when the car was just about to go out of the lane and the recording was during recovering the car toward the center of the street. This was an important addition to the dataset in order to teach the deep neural network how to recover when such conditions happen. The dataset fit the memory and we did not need python generator.

Example of the training images:


Left front camera Center front camera Right front camera

Example of the recovery training images:

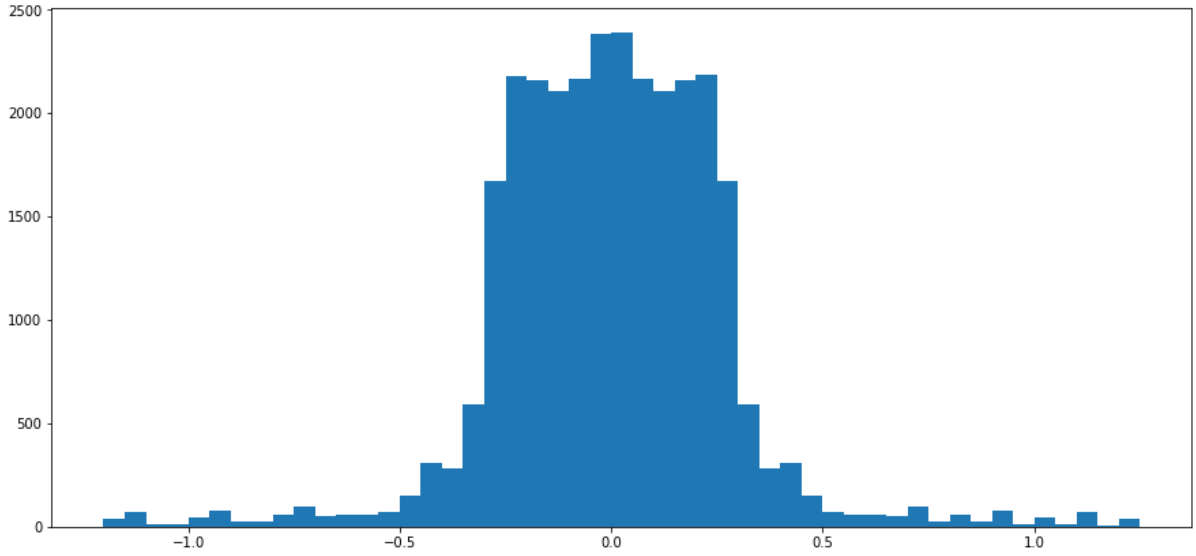


Left front camera Center front camera Right front camera

For data augmentation, we have added a vertically flipped copies of all images. Since the training data of the simulator was collected from a counter clockwise driving around a lap, most of the training data was with left turning. So to prevent inbalance of the training samples, copying a flipped version of the images not only increased our sample size but corrected this inbalance ratio of left to right turned driving samples. The steering measurements of the flipped copies are the same values muliplied by -1.

Most of the time the simulated car during training was going straight without steering. So the 0 degree steering data was significantly larger than any other steering measurement. To prevent over predicting 0 degree steering by the neural network in the autonomous mode, I had to drop out around 9 out of ten images with 0 degree steering measurements.

The final dataset had a nice bell curved shape histogram:



Designing Model Architecture

This task is a computer vision task, so the convolutional neural network architecture was the best candidate to succeed in this project. We used a modified LeNet architecture her as the follows:

- Data normalization layer: Scale all image pixel values within the range [-1, 1], via the following operation: `x/255.0 - 0.5`
- Cropping out the upper part of the images which is do not give any useful information for the network and consists mainly of the sky which makes about 70 pixels at the top of the frame, and cropping out the lower part of the image which most of it consist of the dashboard which is about 25 pixels at the bottom
- Convolution with 5x5 kernel, stride of 1, depth of 6, valid padding

- RELU activation
- Max Pooling 2x2
- Convolution with 5x5 kernel, stride of 1, depth of 6, valid padding
- RELU activation
- Max Pooling 2x2
- Flattening
- Fully-connected with 120 hidden units
- Fully-connected with 84 hidden units
- Fully-connected with 1 unit (steering angle output)

Optimizer: Adam

Loss: MSE (which is better suited for regression problems like ours.

Training and Parameter Tuning

Validation was 20% of the training data, however the validation did not give consistent results with the performance of the Autonomous driving, so I have decreased the validation dataset portion into just 2%, just to let me know if there is a bug in my coding. The extra images that went to the training set was hopefully helpful to increase the performance. Batch size of 16 and five epochs training was enough, since beyond that did not give better training loss or validation loss. Total number of parameters of the network was 723,091, and it took around 50 seconds of training time for each epoch on the AWS g2.2xlarge instance. Note that the “test set” was not necessary, since the generalizing capabilities of the network was being tested on the simulator itself. The adam optimizer for optimizing the mean square error (MSE) was used. The MSE is perfectly suited for regression problem like predicting the angle of the steering. To prevent overfitting, we did not prefer to train with more than 5 epochs. We noted that after 5 epochs no gain we got in term of validation loss. Learning rate of 0.001 seemed optimal for the project and gave better results.

We tried NVIDIA's "End to End Learning for Self-Driving Cars" (http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf) but the LeNet architecture gave better results.