

Project 3 - Behavioral Cloning - Udacity Self-driving Car Nanodegree

Behavioral cloning write up

Goals of project

- Gather data from simulator
- Train model on gathered data
- Create Convolutional Neural Net to train data
- Use the Nvidia model for training
- Test model on simulator (autonomous mode)

Structure of project directory

- data (CSV file used for training)
- IMG (File not included in github repo - images used for training)
- drive.py (script used to interact with simulator in autonomous mode) -model.h5 (saved model after training)
- utils.py (file that contains all of the helper functions needed for model.py)
- model.py (training file)
 - imports utils.py as a module

Model used in training

NVIDA model used

Image normalization to avoid saturation and make gradients work better.

Convolution: 5x5, filter: 24, strides: 2x2, activation: ELU

Convolution: 5x5, filter: 36, strides: 2x2, activation: ELU

Convolution: 5x5, filter: 48, strides: 2x2, activation: ELU

Convolution: 3x3, filter: 64, strides: 2x2, activation: ELU

Convolution: 3x3, filter: 64, strides: 2x2, activation: ELU

Drop out (0.5)

Fully connected: neurons: 100, activation: ELU

Fully connected: neurons: 50, activation: ELU

Fully connected: neurons: 10, activation: ELU

Fully connected: neurons: 1 (output)

- the convolution layers are meant to handle feature engineering
- the fully connected layer for predicting the steering angle.
- dropout avoids overfitting
- ELU(Exponential linear unit) function takes care of the Vanishing gradient problem.

Return model

This is found in model.py lines 34-74

Training Data chosen




Epochs, Batch size, and sample per Epoch

EPOCHS USED:5

BATCH SIZE USED: 40

Sample per Epoch : 30000

The training data used to train the model was gathered by driving around the simulator track. I drove the car around the track ~4-5 times trying my best to stay in the center and keep a safe turning angle when encountering a sharp turn. After many iterations in training, I finally found a batch of data points that would give me the best (all be it not exactly perfect on the right turn...I tried my best to get that near perfect :))results when paired with a good ConvNet.

| Example of Center image | Example of Left | Example of Right |
|---|---|---|
|  |  |  |

When the training data had been collected, a CSV was created along side it. I used the CSV file to read in the data for my model and the steering angles of the images as well. These data points were placed into a data frame using pandas.

Final Steps

The final steps in my process included researching a generator approach for augmenting the data, this is found in the utils.py file. Originally I was flipping all images at a rate of 60% which caused a weird side effect I didn't foresee until it happened. When I tested my model on track one, the model would do well on all of the left turns(which I had been having trouble with after crossing the bridge around the dirt patch) but it would not complete the right turn near the water. This would cause the car to attempt to make the turn but nose dive into the water shortly after. I kept seeing the angle try to steer hard right with a 25 degree angle(or some variation of that) and it puzzled me for sometime. I then took a quick video of the problem, and headed to the forums to consult with my peers. Graciously, a solution was proposed and it turned out that my 60% flip rate was the problem.

I had been flipping so many images that the ones I needed for the right turn were becoming few and far between because of the many left turns in the track. So in short, my car was biased to left turns. I removed the 60% flip rate and the car hustled around the track like a champ!! **#WINNING**