Behavioral Cloning

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

Behavioral Cloning Project

The goals / steps of this project are the following: * Use the simulator to collect data of good driving behavior * Build, a convolution neural network in Keras that predicts steering angles from images * Train and validate the model with a training and validation set * Test that the model successfully drives around track one without leaving the road * Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files: * model.py containing the script to create and train the model * drive.py for driving the car in autonomous mode * model.h5 containing a trained convolution neural network * writeup_report.md or writeup_report.pdf summarizing the results

2. 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
```

3. 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

1. An appropriate model architecture has been employed

My model consists of a convolution neural networkis with three 5x5 filter and two 3x3 filter sizes and depths between 24 and 64 (model.py lines 65-69)

The model includes RELU layers to introduce nonlinearity (code lines 65-69), and the data is normalized in the model using a Keras lambda layer (code line 63).

2. Attempts to reduce overfitting in the model

At first I tried using dropout layer at line 70 but model behaved better without it.

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 89). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 88).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I colected data from center lane driving.

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to try diferent models and data sets and applying different Dropout values along with modifying number of epochs. Goal was to find the best structure to reduce the overfit and find model that works best on test data.

My first step was to use a convolution neural network model similar to the LeNet architecture. I thought this model might be appropriate because it worked well on image recognition of road signs.

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 so that instead using LeNet architecture I used NVIDIA end-to-end deep learning model

Then I augmented my data by flipping every image so that data was twice as big.

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track so to improve the driving behavior in these cases, I reduce the number of epochs for training and also used a prepocessing step for images converting to YUV color space system.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 62-75) consisted of a 5 convolution neural networks (with sizes of (24,5,5), (36,5,5), (48,5,5), (64,3,3), (64,3,3)) with the following 4 Fully connected layers and layer sizes of (100, 50, 10, 1) respectively.

3. Creation of the Training Set & Training Process

I tried using more data but I had best results with using less. I recorded one lap on one track than I augmented that data by flipping the images. Besides center camera images I used left and right camera images and that improved my model.

After the collection process, I had 4212 number of data points. I then preprocessed this data by using YUV filter as it is used in NVIDIA model.

I finally randomly shuffled the data set and put 20% of the data into a validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 3 as evidenced by mse. Using more epochs caused mse on validation set to increase while on training set mse wasn't decresing significantly. I used an adam optimizer so that manually training the learning rate wasn't necessary.

To capture good driving behavior, I first recorded one lap on track one using center lane driving. Here is an example image of center lane driving (center, left right camera):