Behavioral Cloning

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 and 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

See function get_model1(). get_model0() is a LeNet architecture based model that also succeeds in driving the car around the lap but with more jerky motions.

My final model (given by the function get_model1()) consists of a convolution neural network with 3x3 and 5x5 filter sizes and depths between 24 and 64 (model.py lines 68-90). The model is a slight adaptation of the Nvidia end to end model taken from the class notes.

The model includes RELU layers to introduce nonlinearity (code lines 72, 74, 76, 78, 80), and the data is normalized in the model using a Keras lambda layer (code line 70).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 73, 75, 77, 79, 81, 84 and 86).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 96). 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 89).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road.

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 make sure that the model correctly predicts the steering angle given the image from the center camera of the car.

My first step was to use a convolution neural network model similar to the LeNet architectre. I thought this

model might be appropriate because it served as a good model in the Traffic sign classifier lab which involved feature extraction from images to classify images. Although this project is different in that it is a regression task, the first few layers of the LeNet architecture are expected to extract meaningful features that the final fully connected layer can operate on to predict the required steering angle.

I got reasonable results with the LeNet architecture. However, I resorted to using a more sophisticated network as shown by the function get_model1() and described in the previous section.

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 the first three convolutional layers used subsampling of (2,2) and all following layers used Dropouts with probability of 0.5.

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, especially right before it enters the bridge and near the dirt track. To improve the driving behavior in these cases, I augmented the data by flipping the images and adjusting the steering angle measurements accordingly (simply multiplying by -1). Also, as mentioned in the class notes, it is obvious that the camera images contain more information than is needed to drive the car around the track. For example the pixels near the top and near the bottom of the image are not necessary to predict the steering angle. Hence I used the Keras Cropping2D layer to crop off those pixels from the input images.

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 69-90) consisted of a convolution neural network with the following layers and layer sizes.

1. Convolution layer with 24 filters of size 5x5 and striding of (2,2).
2. Convolution layer with 36 filters of size 5x5 and striding of (2,2).
3. Convolution layer with 48 filters of size 5x5 and striding of (2,2).
4. Convolution layer with 64 filters of size 5x5.
5. Convolution layer with 64 filters of size 5x5.
6. Flatten layer outputting 1164 neurons
7. Fully connected layer with 100 neurons
8. Fully connected layer with 50 neurons
9. Fully connected layer with 10 neurons
10. Fully connected layer with 1 neurons