Behavioral Cloning

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
- video.mp4 containing video recording of my vehicle driving autonomously at least one lap around the track.

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 5 convolution neural layers with 5x5 and 3x3 filter sizes and depths between 24 and 64, there are also 3 full connected layer and 1 output of final layer(model.py lines 69-81).
The model includes RELU layers to introduce nonlinearity, the data is normalized and cropping in the model using a Keras lambda layer and cropping method (code line 69-81).

2. Attempts to reduce overfitting in the model

The model doesn't contain dropout layers here, sine the number of epochs I used is 3.
The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 22-60). 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 80).

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

Model Architecture and Training Strategy

1. Solution Design Approach

My strategy was trying easy and going complicated. First I built the model using only fully connected layer, then add the convolution layers and take the reference model from Nvidia.

The model was trained fine and the loss converged quickly without any problem. But the performance of model was quite unsatisfied. I moved my focus on the data collection and pre-processing.

2. Final Model Architecture

The final model architecture consisted of a convolution neural network with the following layers and layer sizes. Normalization and cropping are also done here. Here is the architecture.

- model.add(Lambda(lambda x: x / 255.0 - 0.5, input_shape=(160, 320, 3)))
- model.add(Cropping2D(cropping=((65,20), (0,0))))
- model.add(Convolution2D(24,5,5, subsample=(2,2), activation='relu'))
- model.add(Convolution2D(36,5,5, subsample=(2,2), activation='relu'))
- model.add(Convolution2D(48,5,5, subsample=(2,2), activation='relu'))
- model.add(Convolution2D(64,3,3, activation='relu'))
- model.add(Convolution2D(64,3,3, activation='relu'))
- model.add(Flatten())
- model.add(Dense(100))
- model.add(Dense(50))
- model.add(Dense(10))
- model.add(Dense(1))

3. Creation of the Training Set & Training Process

I used the Udacity sample data for general driving behavior training. These data also record the vehicle recovering from the left side and right sides of the road back to center, so that the vehicle would learn how to come back to center when driving outside roads.
To augment the data sat, firstly I randomly choose one image among ['center', 'left', 'right'] for each line. Then I add a correction to left image and substract a correction to right image, where the correction value need to be tuned. I also randomly flipped images and angles so that this would make the data more general and completed.
After previous process, I had 6428 number of data points. I then preprocessed this data by cropped top 65 x 320 and 20x320.
Since I randomly shuffled the data set and put 20% of the data into a validation set., the validation set helped determine if the model was over or under fitting. The number of epochs used here was 3 since the validation loss did not change so much after 3 epochs. Also I used an adam optimizer so that manually training the learning rate wasn't necessary.