Behavioral Cloning

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.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 network with three 5x5 and two 3x3 filter sizes (model.py lines 70-74)

The model includes RELU layers to introduce nonlinearity (code lines 70-74), and the data is normalized in the model using a Keras lambda layer (code line 67).

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 15-17). 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 83).

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 curving**. The number of total images is about **24000**.

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 derive more features from the images and the car can use these features to know how to curve.

My first step was to use a convolution neural network model similar to the LeNet. I thought this model might be appropriate because it can be used to detect numbers or traffic signs in the real world. Maybe this model can detect lanes.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set.

The final step was to run the simulator to see how well the car was driving around track one. There were a spot where the vehicle fell off the track. To improve the driving behavior in this case, I changed the net architecture to NVIDIA net architecture.

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 70-79) consisted of a convolution neural network with the following layers and layer sizes.

Layer 1 Normalized input planes 3@90x320

Layer 2 Convolutional feature map 24@43x156

Layer 3 Convolutional feature map 36@20x76

Layer 4 Convolutional feature map 48@8x36

Layer 5 Convolutional feature map 64@3x17

Layer 6 Convolutional feature map 64@1x8

Layer 7 Fully-connected layer 100 neurons

Layer 8 Fully-connected layer 50 neurons

Layer 9 Fully-connected layer 10 neurons

Output Layer vehicle turning angle

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded three laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover from the sides to center. These images show what a recovery looks like starting from right side:

To augment the data set, I also used left and right side camera images and correction angles thinking that this would help increase the training set. For example, here are images from right and left side cameras.

I also collected more data from curving. These images were quite important to train the car how to turn around.

After the collection process, I had 20000+ number of data points. I then preprocessed this data by cropping and normalization.

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 5 as evidenced by loss. I used an Adam optimizer so that manually training the learning rate wasn't necessary.