Behavioral Cloning

Files Submitted & Code Quality

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 convolutional neural network
- writeup_report.pdf summarizing the results

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

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.

Model Architecture and Training Strategy

An appropriate model architecture has been employed

My model consists of a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 and 64 (model.py lines 48-52)
The model includes RELU layers to introduce nonlinearity (code line 48-52), and the data is normalized in the model using a Keras lambda layer (code line 46).

Attempts to reduce overfitting in the model

The model is trained for only 3 epochs in order to reduce overfitting (model.py line 64).
The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 64). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 62).

Appropriate training data

Only the sample dataset provided by Udacity was used.

Architecture and Training Documentation

Solution Design Approach

The overall strategy for deriving a model architecture was to start with a proven model and build upon it as necessary.

My first step was to use a convolutional neural network model from Nvidia. I thought this model might be appropriate because it was introduced in the class lecture.

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 it only trains for 3 epochs instead of 10 by default.

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 at the first lake on right when it trained for 10 epochs. To improve the driving behavior in these cases, I limited the training epochs to 3.

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

Final Model Architecture

The final model architecture (model.py lines 44-58) consisted of a convolution neural network with the following layers and layer sizes, all in two dimensional when applicable:

1. Lambda layer for color normalization, with original picture input shape
2. Cropping layer that crops 70px from the top and 25px from the bottom of every image
3. Convolutional layer with 24 filters, each with a kernel size of 5 rows and columns, RELU activation, and subsampling or stride of 2 by 2 pixels
4. Same convolutional layer as above, with 36 filters
5. Another convolutional layer with 48 filters
6. A convolutional layer with 64 filters with no subsampling specified so default stride of 1 by 1
7. Another exact same 64 filter convolutional layer
8. A flatten layer to make the resulting weights 1 dimensional
9. A fully connected layer with an output shape of 100
10. A fully connected layer with an output shape of 50
11. A fully connected layer with an output shape of 10
12. Last fully connected layer that has only 1 output which is the predicted steering angle

Creation of the Training Set & Training Process

To capture good driving behavior, I used the sample data provided from the classroom.

To augment the data set, I also flipped images and angles horizontally to create driving examples, as if the car has driven another track that is the same as the one in the dataset, except when the track turns

left in the dataset the augmented track turns right. The steering angle is also flipped so however well the car does in the original dataset it would do the same in the augmented dataset.

After the collection process, I had 38572 number of training data points and 9644 number of validation data points. I then pre processed this data by normalizing the color of each image, and then cropping the top and bottom of each image.

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 the validation results when I trained for 10 epochs. The validation loss decreased at first but then increased after 3 so that's where I made the cutoff. I used an adam optimizer so that manually training the learning rate wasn't necessary.