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

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
- video.mp4 with video of car driving in autonomous mode for one full lap
- 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 5 convolutional layers (first 3 convolutional layers are 5x5 filter sizes and last 2 convolutional layers are 3x3 filter sizes), with depths between 34 and 64 (model.py, lines 69-73).

The model includes RELU layers to introduce nonlinearity (model.py, lines 69-73), and the data is normalized in the model using a Keras lambda layer (model.py, line 63).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py, lines 76-80).

The number of epochs was chosen so that the loss falls monotonically with increasing epochs, to avoid overfitting.

The model was trained and validated on augmented data sets to ensure that the model was not overfitting. Augmented data sets were generated by flipping the images horizontally. 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 84).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, and 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 use the Nvidia pipeline with dropout layers and augmented training images formed by

horizontally flipping the training data to avoid overfitting. Also, the training data was a combination of center lane driving and off center driving (with a ratio of 5:1, center driving:off-center driving).

My first step was to use a convolution neural network model similar to the Nvidia pipeline. I thought this model might be appropriate because its performance has already been demonstrated successfully in a similar context.

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 to add dropout layers in each of the fully connected hidden layers.

Then I made the training size as large as possible to also avoid overfitting. Training images were flipped horizontally, with the corresponding steering output negated, and augmented to the training set.

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. To improve the driving behavior in these cases, I added training when the car is off the center of the lane.

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 62-81ß) consisted of a convolution neural network with the following layers and layer sizes

Input layer, with image values scaled to be between -0.5 and 0.5

Lambda Layer, to scale all values between -0.5 and 0.5.

Cropping Layer, to remove top 70 and bottom 20 pixels of each image

Convolution layer, 5x5 filter, 2x2 stride, valid padding, 24 output maps, RELU

Convolution layer, 5x5 filter, 2x2 stride, valid padding, 36 output maps, RELU

Convolution layer, 5x5 filter, 2x2 stride, valid padding, 48 output maps, RELU

Convolution layer, 3x3 filter, 2x2 stride, valid padding, 64 output maps, RELU

Convolution layer, 3x3 filter, 2x2 stride, valid padding, 64 output maps, RELU

Fully Connected Layer, 100 output nodes, dropout with 0.7 dropout prob.

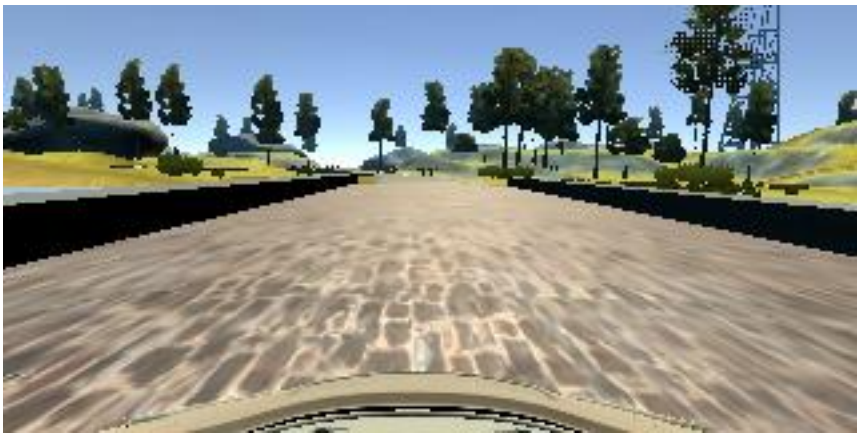
Fully Connected Layer, 50 output nodes, dropout with 0.7 dropout prob.

Fully Connected Layer, 10 output nodes, dropout with 0.7 dropout prob.

Output Layer, 1 output node

3. Creation of the Training Set & Training Process

To capture good driving behavior, I used the sample training data which recorded five 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 the expected steering from off-center positions of the road. These images show what a recovery looks like:

I also flipped images horizontally and negated angles, to augment the dataset and help avoid overfitting.

After the collection process, I had 8,038 number of data points. I then preprocessed this data by scaling all the images values to have a range between -0.5 and 0.5, to avoid numerical instability. Preprocessing also including cropping the top 70 and bottom 20 pixels to limit the portions of the image depicting the road.

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 the loss on both the training and validation sets decreasing monotonically with each subsequent epoch. I used an Adam Optimizer so that manually training the learning rate wasn't necessary.