Behavioral Cloning Writeup

Student Name: Ying Li
Email Address: liying2905@gmail.com

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build a convolutional 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.pdf summarizing the results
- video.mp4 recording of the car driving autonomously one lap around the track.

2. Submission includes functional code

Using the Udacity provided simulator and the 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 convolutional 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 2 conv layers with 3x3 filter size , followed by 2 conv layers with 5x5 filter size, followed by 4 fully-connected dense layers. The details of parameters in each layer is listed in the table in the next big session.

In the first layer, the image data is cropped by removing the top and bottom area using a Keras Cropping2D layer (model.py lines 85).

In the second layer, the data is also normalized in the model using a Keras lambda layer (model.py lines 86).

2. Attempts to reduce overfitting in the model

In order to reduce overfitting, each convolutional layer has a dropout rate of 0.5 (model.py lines 90, 95, 100, 105).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 71-75). 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 119).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, reverse lane driving, and recovery driving (from the side of the road drive to the center of the road)

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

Architecture and Training Documentation

1. Solution Design Approach

The overall strategy for driving a model architecture was to incrementally increase the performance of the architecture.

My first step was to use the LeNet architecture. Because I thought this is a good starting point.

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 has both a low mean squared error on the training set and the validation set. So I thought the architecture may be underfitting.

To combat the underfitting, I added more layers to the model: 2 additional conv layers with 5x5 filters and 2 additional fully connected layers. I also increased the depth of each conv layer to make them grow deeper layer by layer. I set the number of nodes in fully connected layers to decrease layer by layer, in order to get a smooth transformation from a lot of nodes flattened from the last conv layer to the output layer with only one node.

With this new architecture, I found it has a very 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 added dropout layers with 0.5 keep rate to each conv layer. This architecture performed much better than the last one. But the validation error is still a little higher than the training error.

I decided 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 and decrease the validation error, I decided to drive 2 more laps and more recovery scenes from road side to center.

With these additional data, the validation error was significantly lower. I know the overfitting problem is further relieved with more data.

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 83-117) consists of a convolution neural network with the following layers and layer sizes

Layer	Description	Input	Output
Input	160x320x3 RGB image		
Cropping	Crop the top sky (50px) and bottom (20px) to leave only road part in the image	160x320x3	90x320x3
Normalization	Normalize input to value: -0.5~0.5		
Convolution 3x3	1x1 stride, VALID padding	90x320x3	88x318x6
Max pooling	2x2 kernel, 2x2 stride	88x318x6	44x159x6
RELU			
Convolution 3x3	1x1 stride, VALID padding	44x159x6	42x157x12
Max pooling	2x2 kernel, 2x2 stride	42x157x12	21x78x12
RELU			
Convolution 5x5	1x1 stride, VALID padding	21x78x12	17x74x32
Max pooling	2x2 kernel, 2x2 stride	17x74x32	9x37x32
RELU			
Convolution 5x5	1x1 stride, VALID padding	9x37x32	5x33x64
Max pooling	2x2 kernel, 2x2 stride	5x33x64	3x17x64
RELU			
Fully connected		3x17x64=3264	256
Fully connected		256	64
Fully connected		64	8

Fully connected		8	1
Output logits	a single float representing steering angle		

3. Creation of the Training Set & Training Process

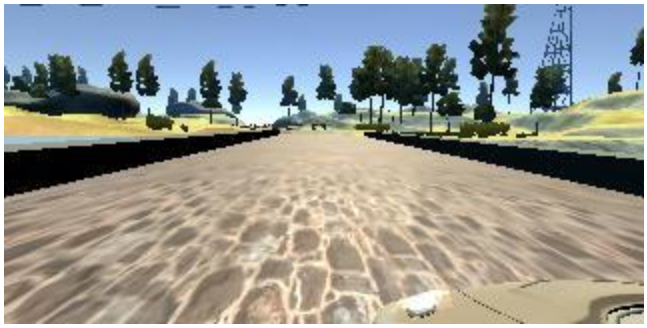
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded the vehicle driving reversely along the loop to connect more samples for it to learn turn right.

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

To augment the data set, I also take advantage of the left image and right image taken from side cameras, so the vehicle learns to drive in different positions. These images are also fed into the training set with the steering angle label adjusted to a small degree. From the perspective of the left camera, the steering angle should be less than the center steering angle. From the perspective of the right camera, the steering angle should be more than the center steering angle. I used a tuning parameter to adjust the correction of angles. These image show the corresponding left image and right image of a center image.

center image



left image

right image

After the collection process, I had 13852 data points. With augmentation of the left and right images, the number increased to 41556 data points. 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 10 as the validation error doesn't decrease anymore. I used an adam optimizer so that manually training the learning rate wasn't necessary.

Simulation

The simulation result is recorded in video.mp4. It shows that no tire leaves the drivable portion of the track surface. The car does not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).