Behavioral Cloning

Writeup Template

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

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 3x3 filter sizes and depths between 32 and 128 (model.py lines 83, 85)

The model includes RELU layers to introduce nonlinearity (code line 78), and the data is normalized in the model using a Keras lambda layer (code line 74).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 87).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 16-17, 58-59). 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 97).

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 by adding a correction of 0.2.

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 passes through five convolutional layers and four fully-connected-layers.

My first step was to use a convolution neural network model similar to the LeNet. I thought this model might be appropriate because it was relatively easy to construct an architecture, and a better performance model could be created by adding more layers.

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 I could see that the loss decreased as the epoch value increased by adding dropout.

Then I add fully connected layers and ReLU activation layers.

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 first, cars left the lane in two sections. In particular, after passing the curve, it was not well positioned at the center of the lane. To improve the driving behavior in these cases, I changed the image file from BGR to RGB and processed it into a suitable image format using cv2.cvtColor(image, cv2.COLOR_BGR2RGB).

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

2. Final Model Architecture

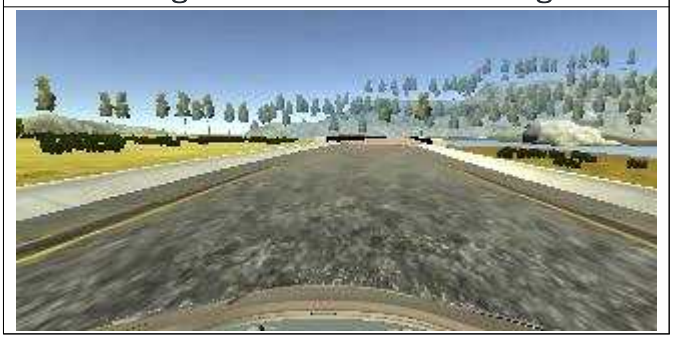
The final model architecture (model.py lines 71-96) consisted of a convolution neural network with the following layers and layer sizes.

Here is a visualization of the architecture.

Layer		Description
	Input	160x320x3 RGB image
	Cropping	90x320x3 RGB image
	Lambda	Lambda(lambda x: (x / 255.0) - 0.5)
Convolution Layer1	Convolution	2x2 stride, filter=24 , row=5 , col=5
	RELU	
Convolution Layer2	Convolution	2x2 stride, filter=36 , row=5 , col=5
	RELU	
Convolution Layer3	Convolution	2x2 stride, filter=48 , row=5 , col=5
	RELU	
Convolution Layer4	Convolution	filter=64 , row=3 , col=3
	RELU	
Convolution Layer5	Convolution	filter=64 , row= , col=3
	RELU	
	Dropout	keep_prob = 0.5
	Flatten	Input = 5x5x16. Output = 400
Fully connected Layer1	Fully connected	Output = 100
	RELU	
Fully connected Layer2	Fully connected	Input = 100, Output = 50
	RELU	
Fully connected Layer3	Fully connected	Input = 50, Output = 10
	RELU	
Fully connected Layer4	Fully connected	Input = 10, Output = 1
	RELU	

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:

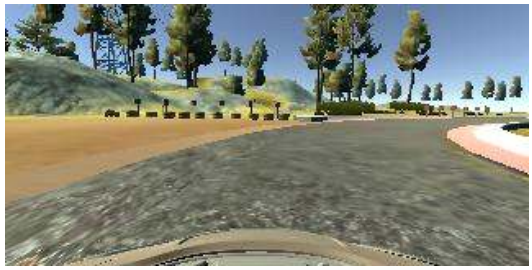
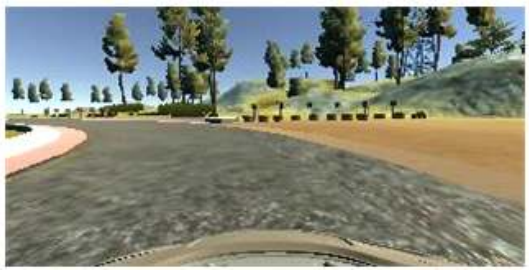
image of center lane driving


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 go back to its place, when it try to get out of the lane. These images show what a recovery looks like starting from left lane to center position :

left side position	recovering position	final recovery position
		

Then I repeated this process on track two in order to get more data points.

To augment the data sat, I also flipped images and angles thinking that this would increases the number of data. For example, here is an image that has then been flipped:

original image	flipped image
	

After the collection process, I had 7800 number of data points. I then preprocessed this data by using the generator function, a function that adds each of the processed images and angular values to the list.

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. When the epoch value exceeded 7, the ideal number of times proved by the constant or rising loss was 7. I used an adam optimizer so that manually training the learning rate wasn't necessary.