Behavioral Cloning

Writeup

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

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.md or 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

I used NVIDIA's self-driving car model. My model consists of 5 convolutional neural networks and 4 fully connected layers.

The model includes RELU layers to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer.

2. Attempts to reduce overfitting in the model

The model contains a single dropout layer after the convolutional layer and before the dense layers in order to reduce overfitting. The keep probability used is 0.8.

The model was trained and validated on different data sets to ensure that the model was not overfitting. The validation data set is 20% of the original dataset. 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.

4. Appropriate training data

The default training data provided in the GitHub repository was used for training purpose.

Model Architecture and Training Strategy

1. Solution Design Approach

I initially used basic LeNet architecture to train the model. The model seemed to work fine but would leave the track and also swiveled a lot.

I then switched to Nvidia's model and the car stayed on track for most of the time except for when the egde of road markers (re-white stripes and yellow solid lines) were missing. It is then I realized the data had to be augmented.

Based on the Domain Randomization idea, I augmented the training data by modifying the brightness of the images. The car seemed to follow the track and would not leave the road when no markers are present to indicate the edges of the road.

To combat the overfitting, I modified the model by adding a dropout layer with a keep_prob of 0.5. After noticing the huge loss I increased the keep_prob to 0.8 (drop probability to 0.2) and the model did not overfit the 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 consisted of a convolution neural network with the following layers and layer sizes

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 90, 320, 3)	0
conv2d_1 (Conv2D)	(None, 43, 158, 24)	1824
conv2d_2 (Conv2D)	(None, 20, 77, 36)	21636
conv2d_3 (Conv2D)	(None, 8, 37, 48)	43248
conv2d_4 (Conv2D)	(None, 6, 35, 64)	27712
conv2d_5 (Conv2D)	(None, 4, 33, 64)	36928
dropout_1 (Dropout)	(None, 4, 33, 64)	0

flatten_1 (Flatten)	(None, 8448)	0
dense_1 (Dense)	(None, 100)	844900
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11

==========

Total params: 981,819

Trainable params: 981,819

Non-trainable params: 0

I used a lambda layer to normalize the data and also cropped the input images to reduce the size of input.

The original dataset had 24111 images.

After modifying the brightness and flipping images if the steering angle is >0.5 or <-0.5 the dataset had a total of 50036 images.

I 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.

I used 2 epochs for training. I used an adam optimizer so that manually training the learning rate wasn't necessary.

Summary of data augmentation

Original image

flipped image

augmented image