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.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 is based on NVIDIA architecture which consists of 5 convolution neural networks with 5x5(3) and 3x3(2) filter sizes and depths between 24 and 64 (model.py lines 59-87)

The model includes LeakyReLU layers to introduce nonlinearity, the data is normalized in the model using a Keras lambda layer (code line 61) and cropped using Keras cropping2d layer(code line 63).

2. Attempts to reduce overfitting in the model

The model contains a dropout layer(model.py line 77) and 3 max-pooling layers in order to reduce overfitting.

The model was trained and validated on different data sets to ensure that the model was not overfitting. 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 86).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used the sample data provided for the project, and effectively used the 3 camera images. Also, I did augmentation to the data set by flipping the images and corresponding angles.

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 reduce the validation loss and to check if the car is able to drive on the road successfully.

My first step was to use a convolution neural network model similar to the NVIDIA. I thought this model might be appropriate because I tried on LeNET as well, but NVIDIA architecture provided the less loss output, and car was less wobbling on the road.

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 introduced a dropout layer into the model.

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, it was having left bias and to improve the driving behavior in these cases, I used the left and right camera images and provided correction to the corresponding angles, also I augmented the data-set with flipped images.

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 59-87) consists of a convolution neural network with the following layers and layer sizes:

Here is a visualization of the architecture:

Layer	Input	Description	Output
Lambda Layer	Images(160,320,3)	Normalization for 0 mean and low standard deviation	Normalized Images(160,320,3)
Cropping Layer	Normalized Images(160,320,3)	For getting the relevant region of the image	Normalized Images(68,320,3)
Convolution Layer	Normalized Images(68,320,3)	filter size = 5x5 ; depth = 24	64x316x24
Leaky ReLU Layer	64x316x24	for activation	64x316x24
Max Pooling Layer	64x316x24	k-size=2x2	32x158x24
Convolution Layer	32x158x24	filter size = 5x5 ; depth = 36	28x154x36
Leaky ReLU Layer	28x154x36	for activation	28x154x36
Max Pooling Layer	28x154x36	k-size=2x2	14x77x36
Convolution Layer	14x77x36	filter size = 5x5 ; depth = 48	10x73x48
Leaky ReLU Layer	10x73x48	for activation	10x73x48
Max Pooling Layer	10x73x48	k-size=2x2	5x36x48
Convolution Layer	5x36x48	filter size = 3x3 ; depth = 64	3x34x64
Leaky ReLU Layer	3x34x64	for activation	3x34x64
Convolution Layer	3x34x64	filter size = 3x3 ; depth = 64	1x32x64
Leaky ReLU Layer	1x32x64	for activation	1x32x64
Dropout Layer	1x32x64	keep_prob=0.5	1x32x64
Flatten Layer	1x32x64	for flattening into 1-D	2048
Dense Layer	2048	Fully Connected Layer	100
Leaky ReLU Layer	100	for activation	100
Dense Layer	100	Fully Connected Layer	50
Leaky ReLU Layer	50	for activation	50
Dense Layer	50	Fully Connected Layer	10
Leaky ReLU Layer	10	for activation	10
Dense Layer	10	Fully Connected Layer	1

3. Creation of the Training Set & Training Process

I used the sample data-set provided for this project.

To augment the data-set, I used the left and right camera images and flipped the images and angles thinking that this would enrich my training data set and would make the model more generalised.

After the collection process, I preprocessed this data by cropping it and normalizing it to 0 mean and low standard deviation.

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 number of epochs I used is 10 after checking the output for many different epochs. I used an adam optimizer so that manually training the learning rate wasn't necessary. I have attached the video as well to show how my model ran.