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.md and writeup.pdf summarizing the results
- video.mp4 that is a video recording of the simulator running in Autonomous mode. Link to the video

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

The model includes RELU layers to introduce nonlinearity (code line 65-67,70-71), and the data is normalized in the model using a Keras lambda layer (code line 62).

My model uses the network architecture provided by NVIDIA for their self driving car training.
The complete architecture is being listed below:

Layers	Remarks
Lambda Layer	Used in order to normalize the data
Cropping Layer	Used in order to crop the top portion of the images as well as the bottom part where the car bumper comes in
Convolutional Layer	Has got ‘relu’ activation, with a shape of 24x5x5 and subsampling of 2x2
Convolutional Layer	Has got ‘relu’ activation, with a shape of 36x5x5 and subsampling of 2x2
Convolutional Layer	Has got ‘relu’ activation, with a shape of 48x5x5 and subsampling of 2x2
Dropout Layer	A dropout having a rate of 0.2

Convolutional Layer	Has got ‘relu’ activation, with a shape of 64x3x3
Convolutional Layer	Has got ‘relu’ activation, with a shape of 64x3x3
Flatten Layer	In order to flatten the network and make the transition to a fully connected network
Fully connected Layer	Having 100 nodes
Fully connected Layer	Having 50 nodes
Fully connected Layer	Having 10 nodes
Fully connected Layer	Having 1 output node

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 69).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 81). 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 79).

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, using the left and right images with correction factors as well as flipping the images in order to train the model on a more generalised dataset.

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 start with a ver basic structure and mve forward adding into the network makin it more and more robust with each iteration.

My first step towards making model was to adding a Lambda Layer for normalizing the data and flattening it to output it from a layer having 1 node.

This made the car run for a short distance but didn’t make it run for a longer distance without going off the track.

My next step was to feed in the same network used above with images that were flipped horizontally. This added the amount of data that was being fed into the network.

This did improve the performance of the model on straight stretches, but it still was failing at the curves.

The next step was to change the network architecture that was being used. I used LeNet architecture. But, instead of making it a network for classification, it was used to perform regression.

This model worked well but was still struggling at center regions of the track. Trying to compensate this, I added dropout to the model.

This improved the situation, but not by much.

Finally, the model was fed with data from left and right cameras.

The final step involved using the network architecture provided by NVIDIA. The architecture along with the big dataset enabled the model to keep the car on the road for the complete lap around the track

2. Final Model Architecture

The final model architecture (model.py lines 60-76) consisted of a

convolution neural network with the following layers and layer sizes:

Layers	Remarks
Lambda Layer	Used in order to normalize the data
Cropping Layer	Used in order to crop the top portion of the images as well as the bottom part where the car bumper comes in
Convolutional Layer	Has got ‘relu’ activation, with a shape of 24x5x5 and subsampling of 2x2
Convolutional Layer	Has got ‘relu’ activation, with a shape of 36x5x5 and subsampling of 2x2
Convolutional Layer	Has got ‘relu’ activation, with a shape of 48x5x5 and subsampling of 2x2
Dropout Layer	A dropout having a rate of 0.2
Convolutional Layer	Has got ‘relu’ activation, with a shape of 64x3x3
Convolutional Layer	Has got ‘relu’ activation, with a shape of 64x3x3
Flatten Layer	In order to flatten the network and make the transition to a fully connected network
Fully connected Layer	Having 100 nodes
Fully connected Layer	Having 50 nodes

Fully connected Layer	Having 10 nodes
Fully connected Layer	Having 1 output node

3. Creation of the Training Set & Training Process

In order to compensate for the conflict between the colorspaces in model.py and drive.py, I've used **matplotlib.image as mpimg** to read the images instead of **cv2**. This read the images in teh RGB colorspace for model.py.

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 recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover from the edges of the track

To augment the data sat, I also flipped images and angles thinking that this would make the model train on a more generalised dataset. Furthermore, I also took into account the left and right camera images.

After the collection process, I had 22476 number of data points. I then preprocessed this data by normalizing it through the Lambda layer in my network.

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 3 as evidenced by multiple iterations and watching the Validation loss. I used an adam optimizer so that manually training the learning rate wasn’t necessary.