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](https://review.udacity.com/#!/rubrics/432/view) 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 consists of a convolution neural network with 3x3 and 5x5 filter sizes and depths between 24 and 64 (model.py lines 49-53). The data were then flattened, and dense layers were applied.

The model includes RELU activation to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer (code line 18).

Additionally, the images were cropped to exclude unnecessary imagery like the sky and the car hood.

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py line 16-36). 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 62).

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, driving slowly around curves, driving the track in the opposite direction, and driving on the other map.

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 simple and add layers until the requirements were met.

My first step was to use a convolution neural network model similar to the LeNet. I thought this model might be appropriate, because it is powerful for processing images.

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 relatively high mean squared error. So I added complexity to the model by preprocessing the images and adding 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... to improve the driving behavior in these cases, I diversified the data set. I flipped all the images and cropped the images accordingly so that more good data was included in the data set.

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 46-58) consisted of a convolution neural network with the following layers and layer sizes:

- 320x160x3 input
- 24x5x5 convolution
- 36x5x5 convolution
- 48x5x5 convolution
- 64x3x3 convolution
- 64x3x3 convolution
- 576x1 flat
- 100x1 dense
- 50x1 dense
- 10x1 dense
- 1x1 dense

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded three 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 .... These images show what a recovery looks like starting from the right side of the lane :

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 generalize the dataset. I used the left and right camera images as well, and I implemented a +/- 0.2 steering correction for each of the left/right images.

After the collection process, I had X number of data points. I then preprocessed this data by cropping a portion of the top and bottom of the image to remove unnecessary pixels.

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 trial and error. I used an adam optimizer so that manually training the learning rate wasn't necessary.