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

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 5*5 filter sizes and depths between 64 and 128 (model.py lines 133-141)

The model includes RELU layers to introduce nonlinearity (model.py lines 133-141), and the data is normalized in the model using a Keras lambda layer (code line 134).

A single neuron is then used in a Dense layer to make a single prediction on the steering angle after the data is flattened.

*2***.** *Attempts to reduce overfitting in the model*

The model contains an on-epoch-end class to reduce overfitting (code lines 152 - 157).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code lines 48-106). 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 161).

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, and driving on opposite sides of the track.

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 gradually increase the model's width and depth until I reached a point at which the model was either overfitting too much or drove at a desired accuracy.

My first step was to use a convolution neural network model similar to the NVIDIA model. I thought this model might be appropriate because it is used by the team at NVIDIA to drive actual vehicles autonomously.

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 that at the end of every epoch, a class would run an detect whether the validation loss was below a desired accuracy. If that was the case, I would stop training.

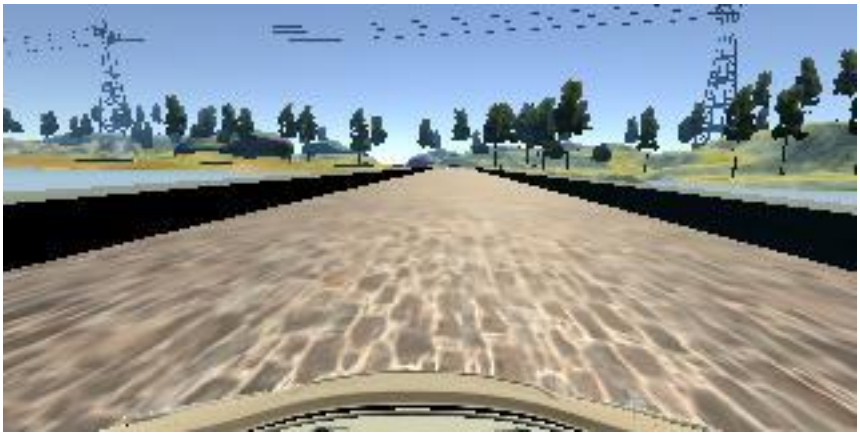
The reason I decided not to include Dropout layers was that the model was achieving very low validation losses early on during training therefore I found them unnecessary. However, if the model was more complex or if the track was more difficult to navigate, I would include Dropout layers to further combat overfitting.

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 collected more data specific to these points and retrained the model to learn on my driving behavior.

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

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:



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 should it leave the center of the road. These images show what a recovery looks like starting from left to center:




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 increase my data while also augmenting it.

After the collection process, I had 77, 432 data points. I then preprocessed this data by copping the images and mean centering the data to 0.

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 1 or 2 as evidenced by the validation accuracy. I used an adam optimizer so that manually training the learning rate wasn't necessary.