Behavioral Cloning

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.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.

Training Strategy (Lines 47 to 155) and Model Architecture (Lines 171 to 188)

1. An appropriate model architecture has been employed

My model consists of a convolution neural network developed by NVIDIA. I chose this architecture after first trying a LeNet architecture to see if my code function correctly and also after visiting the Slack channels to see how other students had approached this problem in the past.

Input: The input to the model is a 160x320 pixel image that is normalized with a center of 0. The image is then cropped by 70 pixels from the top, 20 pixels from the bottom and 20 pixels from each side.

Architecture:

Layer 1: 24 Layer, 5x5 kernel

Layer 2: 36 Layer, 5x5 kernel

Layer 3: 48 Layer, 5x5 kernel

Layer 4: 64 Layer, 3x3 kernel

Layer 5: 64 Layer, 3x3 kernel

Layer 6: Fully connected 100 element

Layer 7: Fully connected 50 element

Layer 8: Fully Connected 10 element

Each layer uses a Relu activation function to introduce nonlinearities.

Loss was computed by mean-square error since we are regressively picking a steering angle and the 'adam' optimizer was used.

A generator was also used to help with memory management. The generator function is defined in lines 9-42. Within the generator function, code for data augmentation is also built in. While there is provision built in for using the left, center, and right images, while also flipping the center image to help generalize the model, only the left, and right images were added to the data set with an aggressive steering angle augmentation (+/- 0.45) to help center the car. I used 3 Epochs after observing the validation score of the model decreasing monotonically through about 3 or 4 Epochs and plateauing at 5.

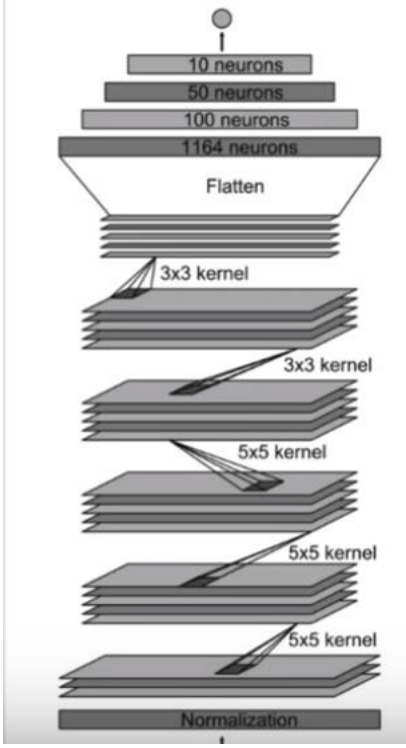

Figure 1. NVIDIA network borrowed for this project

2. Attempts to reduce overfitting in the model

I tried to experiment with Dropouts to reduce overfitting but found no noticeable improvement for model performance.

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 47-155). Early in model development and training I had found little success in driving smoothly and found the model to 'lock up' (i.e. frozen steering angle). To attempt to debug I drove the vehicle using sinusoids, purposely driving from edge to edge on the track to see if the model would respond. To my surprise, with just a couple of laps of training data, the car almost made it all the way around albeit with very erractic, sinusoidal driving. Encouraged that the model was working, I added a sinusoidal style driving, with less amplitude and with just over 7000 images, was able to drive around the track, almost centered the entire way around. However, the intent was to train the model using good driving examples, so I kept adding more training sets until I could train without the sinusoidal input.

What I finally settled on was utilizing smooth driving styles and adding recovery driving as needed until the model performance was acceptable. I found that adding the left and right camera angles with steering augmentation of +/- 0.45 steering, allowed me to stop using the sinusoidal data and gave the model enough recovery training to stay on course.

To help generalize the model, I also added a full lap of the mountain driving. While the model cannot drive completely around the mountain course, it make is about ¾ of the way at about 5-8 mph, though somewhat erratically.

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. It does not drive over the curbs and although the video shows the model driving briefly on the red rumble strip, when not recording the video, the model only gets close to the rumble strip rather than driving on it. There could be some lag the steering angle due to the increased resource load during the recording but I have not verified that completely. Rather, I just noticed the difference in model behavior. The difference in performance is somewhat mitigated by making the display size smaller during recording

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 190).

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 as suggested in the class videos. As I mentioned before, this initially had not worked for me in the beginning and used sinusoidal inputs to see if the model was working. I eventually made my way back to smooth input driving and recovery driving.

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 first try the LeNet architecture since I was familiar with it. Then I implemented the NVIDIA architecture since it had already been optimized by NVIDIA. I figured I would stick with something that works while I learned what training strategies would work and which ones would not.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set (line 161). I also shuffled the data to prevent the model from memorizing the track.

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. I initially used 4 to 5 Epochs but found the overfitting to be lessened if I reduced the number of Epochs to 1.

I also noticed that the model had a tendency to over fit when only the smooth driving style was in the training set. What I also noticed while watching the output was that if the model did not see any high steering angle during training, it would not know to correct for.

After that I started to add recovery driving which helped to increase the loss (less overfitting). Adding left and right images also helped with this trend, and adding one complete lap of slow driving on the mountain course further generalized the model. By this time, I could increase the number of Epochs to 3 and the training and validation losses had become more equal, hovering around 0.11 to 0.12

The final step was to run the simulator to see how well the car was driving around track one. I noticed that training with the same data set enabled, but re-training the model yielded slightly different results. Adding additional training data to help smooth out the driving or improve turning performance around a certain corner would take 2 or 3 trials of training to see if the data set really helped or not.

The other difference I noticed was with different driving speeds. During training, I tended to drive quickly (30mph) since I was focused more on steering. However, for most of the models that could successfully drive around the track, a speed of 10-15mph was best. Faster speeds would induce oscillations in the steering angle, likely due to the aggressive left and right camera steering angle correction

While the video shows a successful completion of 2 laps of driving on track one, the model also can drive around a significant portion of track 2, albeit at speeds of only 5-8 mph. The PI gains on the accelerator pedal also have to be adjusted due to the increased grades seen on Track 2.

At the end of the process, the vehicle is able to drive autonomously around Track 1 without leaving the road.

2. Final Model Architecture

The final model architecture was unchanged from the description shown above. Most of the focus in developing the model was in taking additional data to see how the network would react to additional data

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded one lap on track one using center lane driving. Here is an example image of center lane driving:

Figure 2. Smooth driving, Track 1

Here is the steering angle for that data set

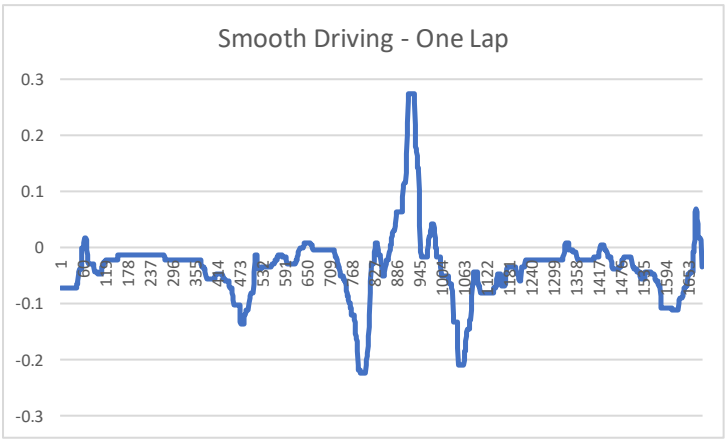

Figure 3. Steering Angle

I also added a reverse driving data set

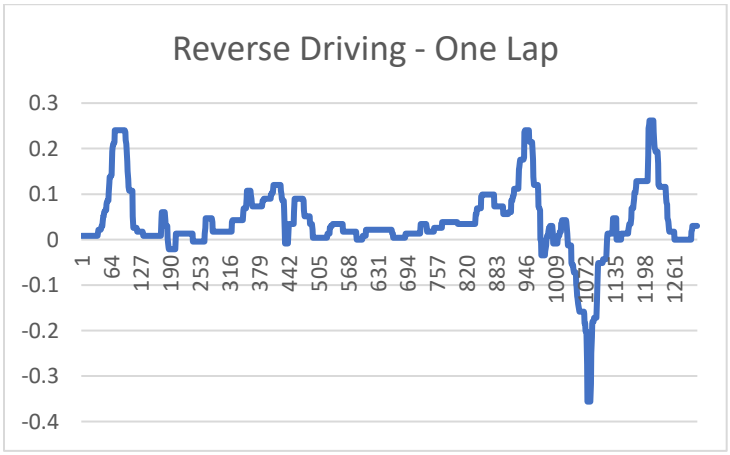

Figure 4. Steering Angle, Reverse, Track 1

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 with larger steering angles. These images show what a recovery looks like

Figure 5. Sequence of images during recovery driving from the right edge

I also added several data sets with recovery driving. To make the model try to return more quickly, I deleted lines in the driving_log.csv file that had 0 steer angle. The following graph shows an example of the steering angles for the remaining files used for training

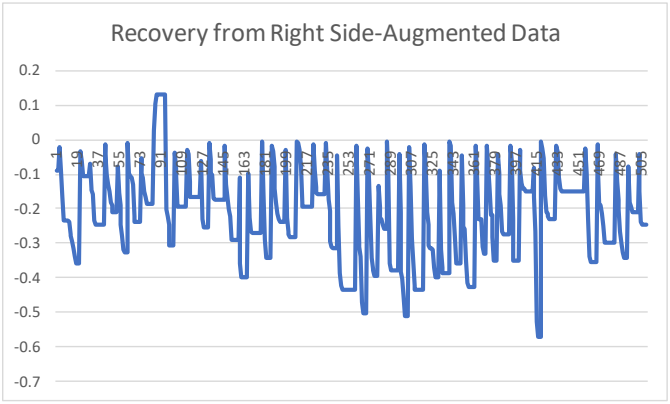

Figure 6. Example of data set with driving for recovering from the right side

As I observed the performance of the model, I added additional

To augment the data sat, I also added the left and right camera angles with an additional steer angle of 0.45. I had also tried to include a flipped version of the images but I never noticed an improvement in training from using the flipped images. Here is an example of the left side view of a smooth driving training lap

Figure 7. Right side camera angle during centered, smooth training lap

The complete set of data are shown in the code from lines 47 to 155 with data sets used . The total number of images for training was 39648 which comprise 80% of the data. The remaining 20% of the data were used for validation.

When I run model.py the following training loss and validation loss for the 3 Epochs are as shown. Subsequent Epochs did not have significant reduction in loss and validation loss when I ran up to 5 Epochs

Epoch	Loss	Val_loss
1	0.1221	0.3153
2	0.1131	0.1886
3	0.1058	0.1603

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

With this set up I was able to complete 2 trips around Track 1 but only made it around about ¾ of Track 2.