Behavioral Cloning - Project 3
William Keller
March 25, 2017

Project

- Use the simulator to collect data of good driving behavior
- Build a convolution neural neural network (CNN), using Keras, that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test the model to ensure it can successfully drive around the track without leaving the road
- Written summary, which you are reading

Rubric Points

Files Submitted and Code Quality

1. My project includes the following files:
    a. model.py, which contains the script used to create and train the model
    b. drive.py (I used the version included with the project resources with no changes)
    c. model.h5, which is the trained neural network generated by model.py listed above
    d. writeup_report.pdf summarizing the project results, which you are currently reading
2. Functional Code
    a. This submission includes functional code which uses the Udacity driving simulator, drive.py, and model.h5. The simulator can drive autonomously by executing *python drive.py model.h5*
3. Usable and Readable Code
    a. Model.py can be (and was) used to train and save the CNN and included the pipeline I used to train and validate the model along with comments to explain the code where necessary

Model Architecture

1. Appropriate architecture employed
    a. My model consists of the NVIDIA framework with five convolutional layers (lines 68-74) and four direct layers (lines 76-79). It uses RELU activation in the convolutional layers, and a lambda layer to normalize (line 66).
2. I tried adding dropout layers in other versions of my project in an effort to reduce overfitting, but they proved unsuccessful. I also tested with as many as ten epochs and as few as one in different versions of the code. I found, and I am not certain I understand this, that when I reduced the number of epochs to where the validation loss

looked the lowest, the validation loss would end up higher than expected. In other words, validation loss at epoch #5 when running five epochs was very close to the validation loss at epoch #2 when running two epochs. I did split the data set into training and validation subsets, with 20% of the data binned to validation.
3. The model uses an adam optimizer.
4. Training data consisted of center lane driving only. Attempts to record recovery driving only seemed to make the model more erratic when in autonomous mode - I may not have recorded the recovery laps as well as possible.

Training Strategy

1. I employed an iterative strategy to develop the model. The first steps were to get the simulator working and learn how to drive it. I also tried running the first very simple models (following along with the lessons) on my laptop, which was slow, but serviceable. Once I added some complexity, however, it became abundantly clear a GPU would be necessary. So, I had to take a break from the main project to learn how to connect my Windows laptop to AWS.

   Once I had the various parts up and running (driving simulator, AWS via PuTTy, FileZilla), I was able to work with more elaborate architectures. I began with a LeNet architecture, which was able to successfully drive the car right into a lake. I added various image processing steps (detailed later in this document), which helped increase the length of time the simulated vehicle could stay on the road. Eventually, however, I replaced the LeNet architecture with the larger NVIDIA architecture, which was a huge improvement. Not perfect yet (that required more image processing), but that ended up being the final architecture.

2. The final model architecture (model.py lines 65-79) consists of the NVIDIA convolution neural network with five convolution layers followed by four direct layers.

3. To create the data set for training and validation, I drove around the track once forward and once backward. (I think it was once each way; it may have been twice.) All laps were intended to demonstrate center lane driving. Within the model, I augmented the data considerably. All images from the center camera are flipped (along with the steering angles) to double the number of samples. Then for every data item with a steering angle other than zero the model adds the left or right camera image, and those are then flipped too. This not only adds to the absolute number of examples for the model to process, it also helps to correct for the bias toward straight-ahead driving. I don't remember if I saw this suggested on the forums or elsewhere, but it was the key decision to get the model to work.

   To preprocess the images, I cropped the tops and bottoms of the images, and also normalized the pixel values to a range of -0.5 to +0.5.

20% of the data set is binned to validation in the model, with the balance serving as the training data.

I tried adding dropouts to the model and also varied the number of epochs from 1-10 (I used 1, 2, 3, 5, and 10 epochs at various points). I knew ten was going to be too many, but I wanted to see how the validation loss varied as the number of epochs extended.