Behavioral Cloning

Files:

- model.py – script to create and train the model
- helper.py – helper methods to augment data and read images
- model.h5 – trained CNN
- video.py – script to play video made up of the images in the output directory
- run1.mp4 – video of the car driving autonomously around the track

Final Model Architecture

- A CNN based off of the commaai model seen here.
    - lambda layer – normalize the color values between -1 and 1
    - Cropping2D – crop the top and bottom of images to remove sky and hood of car. These parts of the image introduce unnecessary noise.
    - 3 Convolutional layers
        - 8x8 kernel, depth of 16, stride of 4, relu activation
        - 5x5 kernel, depth of 32, stride of 2, relu activation
        - 5x5 kernel, depth of 64, stride of 2, relu activation
    - Flatten
    - Dropout layer – keep probability .2
    - Fully connected layer – size of 512
    - Dropout layer – keep probability .5
- Adam Optimizer, and MSE loss function
- Python generator. I trained this model locally on a MacPro and could not keep all of the data in memory. The solution was to use the "fit_generator" function provided by Keras. This fed the model batch_size=128 images at a time. There are two generators, one for training, and one for validation:
    - Training generator: Includes left/right images, and randomly flipped images to avoid a "left-turn" bias of the track
    - Validation generator: only center images, no random flipping. I wanted the validation set to be similar to what the model would see in testing.
- Dropout layers help prevent overfitting
- 4 Epochs
- Validation set was 30% of the center images, or 10% of total images

Architecture Discussion

- I went through several iterations of model architectures starting with the Nvidia model discussed here. Ultimately, I felt this model may be too powerful (too many parameters) and was overfitting the dataset. I tried gathering a lot of data for this model to prevent overfitting (see data discussion below) but testing performance was poor. I also tried "scaling down" the model by adding MaxPooling layers, removing convolutional layers, increasing kernel size, removing fully connected layers, and adding dropout. These modifications resulted in better performance, but the model was consistently fooled by the dirt patch after the bridge, and would always drive off the road
- The commaai model seemed to provide the right balance of convolutions for my dataset and performed well.

Data Collection / Augmentation

- Data collection was the most difficult / time consuming part of this project. Ultimately, I used a fairly simple data set consisting of 3 laps driving in the center of the track + extra data collection on the left-hand turn after the bridge. I collected ~9k * 2 (left/right) = ~27k images.
- As mentioned in the architecture discussion, I collected a lot of data for the Nvidia model. Here is a discussion of that data collection process (Note: this is not what my final architecture was trained on):
    - ~7 laps around the track driving in the center
    - A large amount of recovery data – upwards of 50% of the total dataset. Recovery data helps teach the model how to move back to the center of the track once it's wandered towards the shoulder. I toggled recording to avoid recording driving towards the side
    - Using Left / Right camera images - another way to provide recovery data. I use these images with a correction of +/-.25 on the center image steering angle
    - Flipping images / steering angles – helps temper left turn bias
    - Driving around the track backwards – help temper left turn bias

Example Training Data

Center Camera – Steering Angle -0.04333726

Right Camera – Steering Angle -0.04333726 - .25 = 0.29333726

Left Camera – Steering Angle -0.04333726 + .25 = 0.20666724