Behavioral Cloning

Bernard Lange

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Files Submitted & Code Quality:

1. The submission includes all required files and can be used to run the simulator in autonomous mode:
    a. Model.py containing the script to create and train the model
    b. Drive.py for driving the car in autonomous mode
    c. Model.h5 containing a trained convolution neural network
    d. Writeup_report.pdf summarizing the results
    e. Video.mp4 with a recording of vehicle autonomously driving around the track.
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 replicates the network created by the Self-Driving Cars team at Nvidia (more information available here: https://devblogs.nvidia.com/deep-learning-self-driving-cars/). First data was normalized (Lambda), then images were cropped (70 pixels from the top and 25 from the bottom) so factors like the sky, environment and bonnet were not taken into consideration. Subsequently, three 2x2 strided convolution layers were implemented with depths 24-48 and kernel size of 5x5. Then two non-strided convolutional layers with depths of 64 were used. Each convolutional layers used ReLu activation function which introduces non-linearities.
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. Validation samples accounted for 0.2 of total gathered data. The rest was used in training.
3. Model Parameter tuning
   The parameters were tuned using adam optimizer and loss was calculated using means squared error.
4. Appropriate training data
   The training data kept the vehicle on the road. It involved centre lane driving, recovering from difficult turns, like the one with sand on the left side. The track was driven and recorded in clockwise and anti-clockwise. In more difficult patterns the speed was decreased to record more frames. If the trained model encountered difficulties in some parts of the track, the car was positioned in a respective part of the track with correct wheel angle and was left there (speed = 0). The model was trained by using images and appropriate steering angles. Speed wasn’t involved. The vehicle’s speed in autonomous mode was controlled by Proportional-Integral Controller available in drive.py. Moreover, as the vehicle was trained using BGR color

space, the drive.py file had to be edited to acquire photos in the same color space (it was RGB before).

Model Architecture and Training Strategy

1.Solution Design Approach

The main strategy used in my model was to record the vehicle driving around the track and use the images to train the network. Each frame was saved and described in csv with a destination of each frame, velocity, steering angle and braking. For each position, three frames were saved. One in the centre, second with an offset a bit to the left and third a bit to the right. During the training mode, appropriate steering offset was introduced for off-centre frames. Moreover, flipped images were introduced to the model as well. This enabled to significantly reduce overfitting of the model. Only steering angle was used in training the model. As the size of data exceeded the capabilities of my laptop, the generator was used with a batch size of 512. Using keras, the convolutional neural network was implemented (described above) which is used by Nvidia for self-driving cars. To gauge how well the model performs, the dataset was split into the training set (0.8) and validation set (0.2). The model was saved in model.h5 file. Subsequently, the drive.py file was run which fed the model to the Udacity Simulator in autonomous mode. The biggest challenge was the left-turn near the sand part where the vehicle could easily leave the track. In turned out that the reason for it was different color space used in drive.py and model.py. After it was corrected, the model behaved as expected. The final model is capable of safely driving around the track as it is documented in the video file attached to this submission.

2. Final Model Architecture

The final model architecture consisted of a convolutional neural network with following layers, layers sizes, kernel sizes, strides and activation functions (where required):

Layer	Size	Kernel Size	Strides	Activation Fcn
Convolution2D	24	5x5	(2,2)	ReLu
Convolution2D	36	5x5	(2,2)	ReLu
Convolution2D	48	5x5	(2,2)	ReLu
Convolution2D	64	3x3	Non	ReLu
Convolution2D	64	3x3	Non	ReLu
Flatten	-	-	-	-
Fully Connected	100	-	-	-
Fully Connected	50	-	-	-
Fully Connected	10	-	-	-

4. Creation of the Training Set & Training Process.
Numerous laps were recorded both in clockwise and anti-clockwise direction. The centre and off-centre images were saved. The example of centre and off-centre images are shown below:

The images were cropped and flipped inside the main model as described above. The information about images was recorded in a driving_log.csv file. It recorded 53789 frames. This number doesn't include the off-centre images which were also flipped so it should be multiplied by 6. The data was shuffled and 80% was used for training (the rest for validation). The number of epochs was 10 and adam optimizer was used.