This is the write-up report for project3 Behavioral Cloning.

I. Running the Code

Coding environment

There were two different environments I was working under:

1. Local machine - Python 3.6.3 (Anaconda), tensorflow 1.5.0, keras 2.1.3
2. AWS EC2 (carnd-term1) - Python 3.5.2, tensorflow 0.12.1, keras 1.2.1

The code submitted contains both codes with code for one version commented out.
Make sure to switch code if there is an error.

Also, there are minor changes in path coding due to Windows OS / data location.
Try to modify inside of apostrophe (‘ ‘) in case of error. ‘./IMG/’ need to be changed to ‘./’ depending on the different setup.

```
# split('\\') for windows, otherwise might want to change to split('/')
f_center = './IMG/'+row[0].split('\\')[-1]
f_left = './IMG/'+row[1].split('\\')[-1]
f_right = './IMG/'+row[2].split('\\')[-1]
```

Change in drive.py

The only thing changed in drive.py is the speed (from 9mph to 15mph)

II. Model Architecture and Training Strategy

Model architecture & Prevention of overfitting

The base model architecture used is NVIDIA's architecture that introduced during the lessons. There were several methods including data augmentation (image flipping, adding clockwise data), normalization and limiting the number of epochs, dropout, etc.

The hardest part of preventing overfitting was sample size adjustment. This was the first thing I tried differently. Following are the list of data samples I created/used (with sample size):

1) Original Udacity data
2) 1 normal lap (3627 samples)
3) 2 normal laps (7296)
4) 3 mix laps: 2 clockwise, 1 counter-clockwise (8781)
5) 5 clockwise laps (17325)
6) 1 recovery lap: going offroad (5892)
7) 2 zig-zag laps (5919)
8) 5 stable laps: slow, careful laps focusing on stay centered as much as possible (48786)
9) 2 track2 laps (14268)
10) 20 normal laps (63702)
11) Random recovery laps (18258)

However, I realized this was mistake toward the end of project in terms of efficiency, because all rest of parameters were affected by the size of samples that made parameter optimizing extremely difficult.

Anyways, various different combinations (with different pre-processings) were tried, and it ended up final solution was using combination of #5 and #8, 10 laps total without any additional recovery lap other than using 3 cameras.

Model parameter & Data selection

A number of parameters were optimized by trial and error.
Following is the list of parameters (with final values chosen):

- Zero angle filtering ratio (1/40th, see the explanation in later section)
- Angle correction for camera position difference (0.16)
- Number of epoch (3 with standard learning rate)
- Image crop range
- Model parameters

Also, MSE and Adam optimizer were used as loss evaluation and optimizer as suggested during the lesson.

III. Architecture and Training Documentation

Solution Design

For solution architecture, pretty much everything introduced during the lessons was used including lamda, layer, image flipping/cropping, centered driving, usage of 3 cameras and the generator.

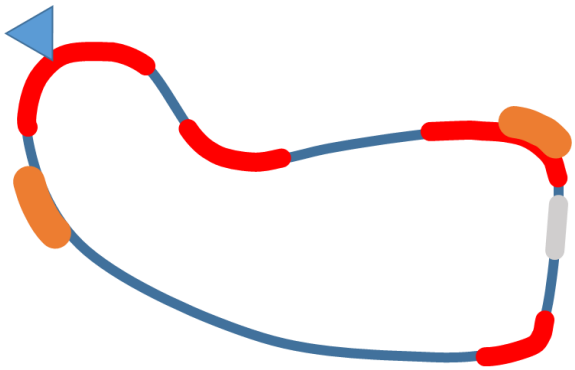
However, the model fail to follow the lane during the sharp turn. One video from one of forum discussions suggests there is imbalance of data between straight roads and corners. So, I began by analyzing track info.

Track analysis

Dimension of the track is not given, so I decided to begin with a quick analysis on the track using the data recording of 1 lap. I came up with following numbers base on simple counting the number of images corresponding to each part of track. Also, I kept maximum speed (30.2mph) consistent as much as possible (except on a couple of steep curves) during the lap so the measurement is reliable.

Following is the sequence of images counted for respective section of the track:
150 curve- 20 normal- 40 ground- 370 normal- 90 curve- 110 normal- 85 bridge- 55 normal- 75 curve(incl. 25 ground)- 90 normal- 55 curve- 60 normal (1200 frames total)

This could be translated into the map below. Also, the time difference between the first and last frame during one lap was approx. 105s. Assuming the speed was constant 30.2mph, we can guess the length of the track is about 0.88 mile.
30.2mile/h * 1h/3600s * 105s = 0.8803mile



Rough shape of the track. (not drawn to scale)

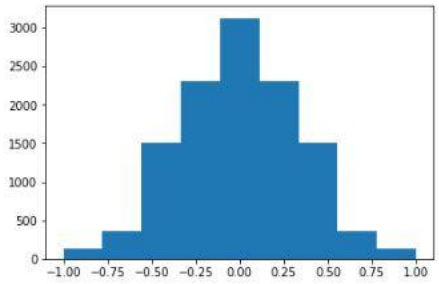
As you may realize, red lines are curves, orange is the empty ground and grey is the bridge.

Steering angles normalization

Then, I came up with following code to normalize zero angles by filtering out zero angles randomly, took only 1/40th of zero angle samples. This 1/40th ratio depends on the type and size of input samples and was optimized by trial and error base on sample size of final working solution – consists of 66111 images (22037 images per each camera).

```
shuffle(lines)
new_lines=[]
angles=[]
for line in lines:
    if float(line[3])!=0 or random.randint(1, 40)==1:
        new_lines.append(line)
        angles.append(float(line[3]))
        angles.append(-float(line[3]))

plt.hist(np.asarray(angles, dtype='float'), bins=9)
plt.show()
```



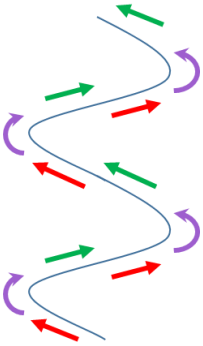
Creation of the training dataset and Training process

One approach I think I tried quite unique is the data filtering as a pre-processing.
I understood the purpose of using 3 cameras and collecting data from recovery laps, but I wanted more systematic way for the vehicle to be centered.

The approach is like following.
More or less, the motion of vehicle is basically like a wave.
We steer toward the another direction when we realize car is gearing toward one direction and repeating. Of course, we normally consider steady long wave with lower number of pitch (one on the right) as a better driving than more number of pitch with zig-zag behavior (one on the left).

One reason I came up with this approach was the streeing was really sensitive and difficult to control. Actually, this is the characteristic of racing games unlike the real world driving because typically the translation rate of turning the front wheels to the steering wheel of real cars is much smaller than the simulator either through keyboard or mouse input.

Anyways, we can break down this wave into 3 parts like below, assuming the driving behavior as a wave form.



Base on an virtual center line, there are lines that going **outward (Red)**, lines **turning (Purple)** and lines going **inward (Green).** What I tried was to eliminate data corresponding to going outward from training because I think this type of data is basically training vehicles to deviate from center line.

On the opposite, green and purple lines both help the training to stay in center.

Examples of each type

C:\Users\j C:\Users\j C:\Users\j	-0.65	0.511496	0	10.18368
C:\Users\j C:\Users\j C:\Users\j	-0.8	0.688075	0	10.49863
C:\Users\j C:\Users\j C:\Users\j	-0.88461	0.512611	0	10.91384
C:\Users\j C:\Users\j C:\Users\j	-0.65828	0.286274	0	11.2178
C:\Users\j C:\Users\j C:\Users\j	-0.41845	0.046451	0	11.3178
C:\Users\j C:\Users\j C:\Users\j	-0.18951	0	0	11.25044
C:\Users\j C:\Users\j C:\Users\j	0	0	0	11.14593
C:\Users\j C:\Users\j C:\Users\j	0	0	0	11.05509
C:\Users\j C:\Users\j C:\Users\j	0	0	0	10.9698
C:\Users\j C:\Users\j C:\Users\j	0	0	0	10.88581
C:\Users\j C:\Users\j C:\Users\j	0	0	0	10.80258
C:\Users\j C:\Users\j C:\Users\j	0	0	0	10.72005
C:\Users\j C:\Users\j C:\Users\j	0	0	0	10.65859
C:\Users\j C:\Users\j C:\Users\j	0.2	0	0	10.56372
C:\Users\j C:\Users\j C:\Users\j	0.4	0	0	10.43797
C:\Users\j C:\Users\j C:\Users\j	0.174662	0	0	10.35393
C:\Users\j C:\Users\j C:\Users\j	0	0	0	10.26822
C:\Users\j C:\Users\j C:\Users\j	0	0	0	10.20587

In the actual data, for example, we can see any non-zero steering values are turning; negative values turing left and positive values turing right. Green and Red values are zero values in between.

Assuming the overall zig-zag wavey behavior is statistically symmetrical, we can say first half of zero values are Green and later half of zero values are Red. So, I took later half of zero samples filtered out of training with following code. (uncommented part)

```
i=0
j=0
while j < len(lines):
    if float(lines[j][3])==0 and float(lines[j-1][3])==0:
        i+=1
    elif float(lines[j][3]) != 0:
        del lines[j-i//2:j]
        j-=i//2
        i=0
    # Comment out this condition if no manual marking is used
#     if float(lines[j][7]) == 1:
#         del lines[j]
#         j-=1
    j+=1
```

Remember, this **DOES NOT** remove any sample image nor data on excel "driving_log.csv" file.

This approach does not solve every problem, but including this tends to yield better result at least and helps a little on data balancing.

In addition, I also included manual filtering (commented out part of code above) which is simply filtering samples out by manually marking on a column in "driving_log.csv." In this way, we can decide whether to use data for training or not without searching for images in a directory one by one.

Model Architecture

NVIDIA'S ARCHITECTURE	DESCRIPTION
Input 160x320x3	
AveragePooling 2x2	Stride of (2,2) to cut the size of input in half, images resized to 80x160x3
Cropping ((11,5),(0,0))	Crop 11 pixels from top, 5 pixels from bottom, image shape 64x160x3
Conv 5x5x24	Stride of (2,2) for first three Conv layer, just like the standard NVIDA architecture. The only difference is there is a dropout added between Conv layers and flatten layer.
ReLu	
Conv 5x5x36	
ReLu	
Conv 5x5x48	
ReLu	
Conv 5x5x64	
ReLu	
Conv 5x5x64	
ReLu	
Dropout (0.2)	
Fully Connected (FC) 400 -> 100	
FC 100 -> 50	
FC 50 -> 10	
FC 10 -> 1	

Model parameters were chosen base on trial and error. Also, dropout was also tried with different values, but commented out because seem to have adverse affect on the result somehow.

Result

Two complete laps are recorded in video.mp4. I'm aware there are some sections where the vehicle gears toward one side, like at the beginning of the bridge, and there is a room for further improvement. It will be easily fixed if I add some recovery samples for those particular sections. However, I didn't want to manually alter, and the model actually passed the test running 5 continuous laps.

Conclusion/Thoughts

I actually started this project quite early, almost 3 weeks ago, but probably ran a couple hundreds trainings until the model finally makes one complete lap. I underestimated the importance of getting a result quickly. I focused more on finding solution than making it run faster. I initially worked on my laptop which does not have any GPU, trying to save AWS credit as much as possible.

At the beginning I got the code running relatively easily but implemented with wrong setup, which ran a couple hours per epoch even on AWS GPU instances. This made me run only one or two trainings per day and took me almost a full week to figure out there was something wrong. And for the rest of weeks I was struggling hard tried to figure out what was wrong.

Now my final model runs about 3~40sec per epoch so I can run almost a hundred trainings a day, and I ran more training last couple of days than rest of weeks combined. The biggest lesson I learned from this project is this; quicker feedback allows better productivity.

Reference

Some sources I looked into were:
Several forum discussion pages under "Project: Behavioral Cloning"
Lecture notes/videos
https://stackoverflow.com/questions/2632205/how-to-count-the-number-of-files-in-a-directory-using-python (for image count of an assertion)