Behavioral Cloning

Yuda Wang

Self-driving Car Nanodegree

Udacity

July 2017

Contents

Chapter 1

Introduction

This is the third project of self-driving car nanodegree, term 1.

There are several potential problems to overcome during the project:

1. Car don't turn big loop at the unsafe place.

Solution: Don't crop the lower side of training image, use relatively higher proportion of training set that turns big loop.

2. Cars often stuck at the side of the driving lane.

Solution: Reduce the proportion of training set that goes straight.

3. The offset of turning to one side.

Solution: Add the flipped version of training data.

4. The amounts of training data samples are less smaller than 40k as Paul Heraty said.

Solution: Add left and right camera data.

5. No analog joystick as Paul Heraty said.

Solution: Become an experienced driver.

Here is my training procedure:

1. Drive 10min clockwise, 10min counter-clockwise with fastest and smallest graphic solution.
2. Add flipped the data set.
3. Add left and right camera data set, with 0.2 degree of offset.
4. Make each direction of training data set is almost equal.
5. Push the data set into generator and train the data.

Chapter 2

Training Code

2.1 Data Prepossessing

2.1.1 Collecting Data

Drive 10min clockwise, 10min counter-clockwise in different folders with fastest and smallest graphic solution.

Merge them together.

```
samples = []

def read_log(file):
    global samples
    with open(file) as csvfile:
        reader = csv.reader(csvfile)
        for line in reader:
            samples.append(line)
```

```
file_0 = 'try0/driving_log_0.csv'
file_1 = 'try0/driving_log_1.csv'

read_log(file_0)
read_log(file_1)
```

2.1.2 Add Flipped Data

To add flipped data, I use cv2.

```
##add_flipped_data


def add_flip_data():
    global samples
    add_flip_samples=[]
    for sample in samples:
        #uncomment here for the first run
        #image_center = cv2.imread('./try0/IMG/'+sample[0].split('\\')[-1])
        #image_left = cv2.imread('./try0/IMG/'+sample[1].split('\\')[-1])
        #image_right = cv2.imread('./try0/IMG/'+sample[2].split('\\')[-1])
        #cv2.imwrite('./try0/IMG/'+'flip_'+sample[0].split('\\')[-1], cv2.fli
        #cv2.imwrite('./try0/IMG/'+'flip_'+sample[1].split('\\')[-1], cv2.fli
        #cv2.imwrite('./try0/IMG/'+'flip_'+sample[2].split('\\')[-1], cv2.fli
        add_flip_samples.append(["\\"+'flip_'+sample[0].split('\\')[-1], "\\"
    samples = samples + add_flip_samples

add_flip_data()
```

2.1.3 Add Left and Right Data

The bias I use is 0.2

```
##add left and right data


def add_lr_data():
    global samples
    add_lr_samples=[]
    for sample in samples:
        add_lr_samples.append(["\\"+sample[1].split('\\')[-1],
        add_lr_samples.append(["\\"+sample[2].split('\\')[-1],
    samples = samples + add_lr_samples
```

```
add_lr_data()
```

2.1.4 Image Summary

Center image from front camera.

Left and right images from side cameras.

Flipped image from front camera.

Flipped images from side cameras.

2.1.5 Balance the Data

The very common way from the Internet, divide the different directions into different sections of range. Make the amount of data in different sections are roughly equal.

An important thing to notice is that this should be done after all the data augmentation finished. If we balance at the very beginning, then the data augmentation would provide no help.

```
##balance data

##nbins and max_examples are hyperparameters

def balance_data(nbins = 2000, max_examples = 200):
    global samples
    samples = np.array(samples)
    balanced = np.empty([0, samples.shape[1]], dtype=samples.dtype)
    histo = []
    for i in range((-1)*nbins, nbins):
        begin = 1.0*i/nbins
        end = begin + 1.0 / nbins
        extracted = samples[(samples[:,3].astype(float) >= begin) & (samples[:,3].astype(float) < end)]
        np.random.shuffle(extracted)
        extracted = extracted[0:max_examples, :]
        histo.append(len(extracted))
        balanced = np.concatenate((balanced, extracted), axis=0)
    print(histo)
    return balanced
```

```
input_data = balance_data()
```

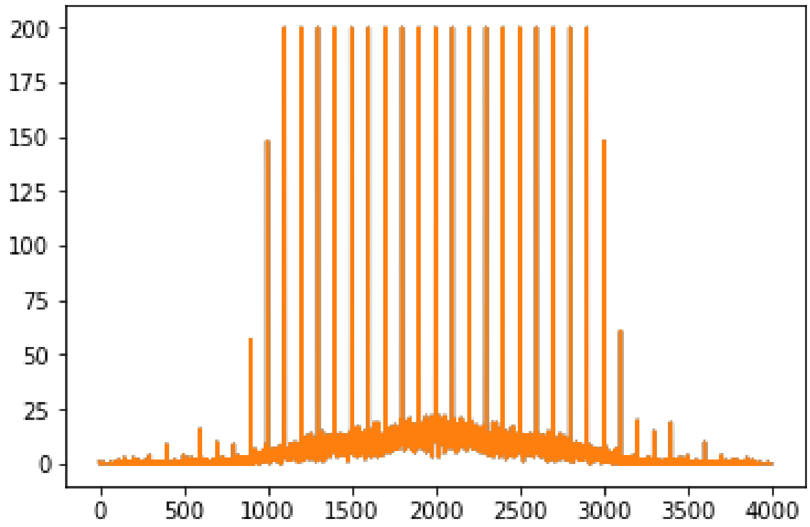
After balance the data

```
input_data = balance_data()

plt.plot(range(len(histo)), histo)
```

```
[<matplotlib.lines.Line2D at 0x7c94fd0>]
```

```
plt.show()
```



2.2 Train the Neural Networks

2.2.1 Generator

Because the training set is large, I use generator with fit generator in keras.

```
from sklearn.model_selection import train_test_split
train_samples, validation_samples = train_test_split(input_data, test_size=0.2)


def generator(samples, batch_size=32):
    num_samples = len(samples)
    while 1: # Loop forever so the generator never terminates
        shuffle(samples)
        for offset in range(0, num_samples, batch_size):
            batch_samples = samples[offset:offset+batch_size]
            images = []
            angles = []
            for batch_sample in batch_samples:
                name = './try0/IMG/'+batch_sample[0].split('\\')[-1]
                image = cv2.imread(name)
                angle = batch_sample[3]
                images.append(image)
                angles.append(angle)



                # trim image to only see section with road
            X_train = np.array(images)
            y_train = np.array(angles)
            yield shuffle(X_train, y_train)
```

2.2.2 Model Architecture

Large dropout probability will cause random variation in the real driving. So I just set the dropout to be 0.1.

```
model = Sequential()
model.add(Lambda(lambda x: x/255.0-0.5, input_shape=(160,320,3)))
model.add(Cropping2D(cropping=((60,0), (0,0))))
model.add(Convolution2D(8, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))
model.add(Convolution2D(16, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(20, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(1))

print("done")
```

Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lambda_1 (Lambda)            (None, 160, 320, 3)       0         
_________________________________________________________________
cropping2d_1 (Cropping2D)    (None, 100, 320, 3)       0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 98, 318, 8)        224       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 49, 159, 8)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 49, 159, 8)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 47, 157, 16)       1168      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 78, 16)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 23, 78, 16)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 76, 32)        4640      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 38, 32)        0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 10, 38, 32)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 12160)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 500)               6080500   
_________________________________________________________________
dropout_4 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 100)               50100     
_________________________________________________________________
dropout_5 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 20)                2020      
_________________________________________________________________
dropout_6 (Dropout)          (None, 20)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 21        
=================================================================
Total params: 6,138,673
Trainable params: 6,138,673
Non-trainable params: 0
_________________________________________________________________
done
```

The model architecture is from the nvidia end-to-end self driving paper. The lower side of the frame isn't cropped considering that left and right image still provides information for the driving lane.

2.2.3 Fit Generator

```
model.compile(loss='mse', optimizer='adam')
best_model = ModelCheckpoint('model_best1.h5', verbose=2, save_best_only=True)
model.fit_generator(train_generator, samples_per_epoch=len(train_samples), validation_
```

The smaller final validation loss doesn't mean better performance. For example, if the data is unbalanced, the 0.016 loss will show worse performance than the 0.02 loss with balanced data.