project4. Behavioral Cloning

February 28, 2019

1 Project4. Behavioral Cloning

1.1 Object of this project is training a car to automatically drive by using video frames I drove

1.2 SEQUENCE

1.2.1 1. FIRST ARCHITECTURE : JUST 1 PERCEPTRON & LAMBDA NORMALIZATION

1.2.2 2. SECOND ARCHITECTURE : LENET

1.2.3 3. ADDING DATA FROM LEFT, RIGHT CAMERA IMAGES

1.2.4 4. ADDING CROPPING

1.2.5 5. THIRD ARCHITECTURE : NVIDIA CNN ARCHITECTURE

1.2.6 6. USING GENERATOR

1.2.7 7. VISUALIZING LOSS

1.2.8 8. DISCUSSION

1.3 import libs I need

In [ ]:
```
import cv2
import csv
import numpy as np
from scipy import ndimage
# This is to visualize loss
import matplotlib.pyplot as plt
```

In [ ]:
```
lines = []

with open('/opt/carnd_p3/data/driving_log.csv') as csvfile:
    reader = csv.reader(csvfile)
    for line in reader:
        lines.append(line)

# This is because first row of csv file is data name (image, steering angle, throttle.
del lines[0]
```

```
images = []
measurements = []
```

1.4 This code is when I used center camera images only

```
In [ ]: for line in lines:
            source_path = line[0]
            file_name = source_path.split('/')[-1]
            current_path = '/opt/carnd_p3/data/IMG/' + file_name
            image = ndimage.imread(current_path)

            images.append(image)
            measurement = float(line[3])
            measurements.append(measurement)

        X_train = np.array(images)
        Y_train = np.array(measurements)
```

1.5 import keras libs I need

```
In [ ]: from keras.models import Sequential
        from keras.layers import Flatten, Dense, Lambda, MaxPooling2D, Conv2D, Cropping2D
```

1.6 1. This is first model just using 1 perceptron and Lambda normalization

I did not expect the result
I thought car trained by this model will just stagger left and right
But unexpectively it somehow drives well
Of course it needs many reinforcement
Its training data loss was 1.963 just after 1 epoch

```
In [ ]: model = Sequential()
        model.add(Lambda(lambda x: x/255-0.5, input_shape=(160,320,3)))
        model.add(Flatten())
        model.add(Dense(1))
        model.compile(loss='mse', optimizer='adam')
        model.fit(X_train, Y_train, validation_split=0.2, shuffle=True, nb_epoch=1)

        model.save('model.h5')
```

1.7 2. This is second model using LeNet architecture

In fact, I expected this result a little, because its LeNet!!
And as a result, it drives somehow well
It could drive through curve but at the bridge, it drove to bridge wall and could not escape
Its training loss was 0.0156 after 5 epochs

```
In [ ]: model = Sequential()
        model.add(Lambda(lambda x: x/255-0.5, input_shape=(160,320,3)))
        model.add(Conv2D(10,(5,5), activation = 'relu'))
        model.add(MaxPooling2D())
        model.add(Conv2D(15,(5,5), activation = 'relu'))
        model.add(MaxPooling2D())
        model.add(Flatten())
        model.add(Dense(120))
        model.add(Dense(84))
        model.add(Dense(1))
        model.compile(loss='mse', optimizer='adam')
        model.fit(X_train, Y_train, validation_split=0.2, shuffle=True, nb_epoch=5)

        model.save('model.h5')
```

1.8 3. So I added left,right camera images to LeNet

This code have at least 2 benefits 1. We can get 3 times more data - Almost 8000 images 24000 images 2. By adding images leaning toward left, right we can train car returning to center - Use correction_factor plus minus steering angle

And surprisingly, by just adding left,right camera images made car drive through bridge well! I thought why

Without left, right camera images, car cannot espcape when it get out of center or road or course

Then by adding left,right camera images, it trained how to escape by turning steering wheel

Assigning larger correction_factor may makes car turn more dinamic

As a result, training loss was 0.0097 after 3 epochs

```
In [ ]: for line in lines:
            for i in range(3):
                correction_factor = 0.2
                source_path = line[i]
                file_name = source_path.split('/')[-1]
                current_path = '/opt/carnd_p3/data/IMG/' + file_name
                image = ndimage.imread(current_path)

                images.append(image)

                if i==0:
                    measurement = float(line[3])
                    measurements.append(measurement)
                elif i==1:
                    measurement = float(line[3]) + correction_factor
                    measurements.append(measurement)
                elif i==2:
                    measurement = float(line[3]) - correction_factor
                    measurements.append(measurement)
```

```
X_train = np.array(images)
Y_train = np.array(measurements)
```

1.9 4. I added Cropping2D to LeNet

Just adding Cropping made car drive well!
I thought why
Cropped background and hood image is more simple than full image
So when car drives automatically and decides which image is equal to present condition, it is maybe more easy and precise
But this car cannot turn left at sharp curve
As a result, training loss was 0.016 after 3 epochs which is higher than before
But car drove better so I concluded getting smaller loss doesn't make sure driving better

```
In [ ]: model = Sequential()
        model.add(Lambda(lambda x: x/255-0.5, input_shape=(160,320,3)))
        model.add(Cropping2D(cropping=((70,25),(0,0))))
        model.add(Conv2D(10,(5,5), activation = 'relu'))
        model.add(MaxPooling2D())
        model.add(Conv2D(15,(5,5), activation = 'relu'))
        model.add(MaxPooling2D())
        model.add(Flatten())
        model.add(Dense(120))
        model.add(Dense(84))
        model.add(Dense(1))
        model.compile(loss='mse', optimizer='adam')
        model.fit(X_train, Y_train, validation_split=0.2, shuffle=True, nb_epoch=3)

        model.save('model.h5')
```

1.10 5. This is third model using NVIDIA CNN architecture

I just used architecture published in NVIDIA homapace
And not a surprisingly, it drove so well
It finished one lap without escaping even center line
As a result, training loss was 0.014 after 3 epochs

```
In [ ]: model = Sequential()
        model.add(Lambda(lambda x: x/255-0.5, input_shape=(160,320,3)))
        model.add(Cropping2D(cropping=((70,25),(0,0))))
        model.add(Conv2D(24,(5,5), strides = (2,2), padding = 'valid', activation = 'relu'))
        model.add(Conv2D(36,(5,5), strides = (2,2), padding = 'valid', activation = 'relu'))
        model.add(Conv2D(48,(5,5), strides = (2,2), padding = 'valid', activation = 'relu'))
        model.add(Conv2D(64,(3,3), strides = (1,1), padding = 'valid', activation = 'relu'))
        model.add(Conv2D(64,(3,3), strides = (1,1), padding = 'valid', activation = 'relu'))
        model.add(Flatten())
        model.add(Dense(100))
        model.add(Dense(50))
```

```
model.add(Dense(10))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
model.fit(X_train, Y_train, validation_split=0.2, shuffle=True, nb_epoch=3)

model.save('model.h5')
```

1.11 6. Make a generator

Fidding all data at once is borne to GPU, so devide data by fit_generator
I devided by 32 data at one batch but it makes making result more slow
So I decided not to use ganerator

```
In [ ]: import os
        import sklearn
        from sklearn.model_selection import train_test_split
        train_samples, validation_samples = train_test_split(lines, 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:
                        for i in range(3):
                            source_path = batch_sample[i]
                            file_name = source_path.split('/')[-1]
                            current_path = '/opt/carnd_p3/data/IMG/' + file_name
                            image = ndimage.imread(current_path)
                            images.append(image)

                            correction_factor = 0.2   # For using left&right camera image
                            if i==0: # Center camera
                                angle = float(batch_sample[3])
                                angles.append(angle)
                            elif i==1: # Left camera
                                angle = float(batch_sample[3]) + correction_factor
                                angles.append(angle)
                            elif i==2: # Right camera
                                angle = float(batch_sample[3]) - correction_factor
                                angles.append(angle)
```

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

    # compile and train the model using the generator function
    train_generator = generator(train_samples, batch_size=32)
    validation_generator = generator(validation_samples, batch_size=32)

    model = Sequential()
    model.add(Lambda(lambda x: x/255-0.5, input_shape=(160,320,3)))
    model.add(Cropping2D(cropping=((70,25),(0,0))))
    model.add(Conv2D(24,(5,5), strides = (2,2), padding = 'valid', activation = 'relu'))
    model.add(Conv2D(36,(5,5), strides = (2,2), padding = 'valid', activation = 'relu'))
    model.add(Conv2D(48,(5,5), strides = (2,2), padding = 'valid', activation = 'relu'))
    model.add(Conv2D(64,(3,3), strides = (1,1), padding = 'valid', activation = 'relu'))
    model.add(Conv2D(64,(3,3), strides = (1,1), padding = 'valid', activation = 'relu'))
    model.add(Flatten())
    model.add(Dense(100))
    model.add(Dense(50))
    model.add(Dense(10))
    model.add(Dense(1))

    model.compile(loss='mse', optimizer='adam')

    model.fit_generator(train_generator,
                        steps_per_epoch = len(train_samples),
                        validation_data = validation_generator,
                        validation_steps = len(validation_samples), nb_epoch=3)

    # model.fit(X_train, Y_train, validation_split=0.2, shuffle=True, nb_epoch=3)

    model.save('model_generator.h5')
```

1.12 7. Visualizing loss

I used history object of model

And it did not represent error but did not plot anything

I think there must be some problems using plt lib in this workspace

```
In [ ]: model = Sequential()
        model.add(Lambda(lambda x: x/255-0.5, input_shape=(160,320,3)))
        model.add(Cropping2D(cropping=((70,25),(0,0))))
        model.add(Conv2D(24,(5,5), strides = (2,2), padding = 'valid', activation = 'relu'))
        model.add(Conv2D(36,(5,5), strides = (2,2), padding = 'valid', activation = 'relu'))
        model.add(Conv2D(48,(5,5), strides = (2,2), padding = 'valid', activation = 'relu'))
        model.add(Conv2D(64,(3,3), strides = (1,1), padding = 'valid', activation = 'relu'))
        model.add(Conv2D(64,(3,3), strides = (1,1), padding = 'valid', activation = 'relu'))
```

```
model.add(Flatten())
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')

history = model.fit(X_train, Y_train, validation_split=0.2, shuffle=True, epochs=3)

print(history.history.keys())
### plot the training and validation loss for each epoch
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model mean squared error loss')
plt.ylabel('mean squared error loss')
plt.xlabel('epoch')
plt.legend(['training set', 'validation set'], loc='upper right')
plt.show()

model.save('model_loss_visualization.h5')
```

2 8. discussion

1. At first, I wanted to gather data from inverse track and escaping from out of line and second advanced track

I don't know why exactly, but simulation was so slowly and did not work smoothly so that I cannot drive better than sample data

And surprisingly just only using sample data (center,left,right image) and NVIDIA CNN architecture, the car drove so well

I don't need more technics like drop out...

Especially this project was so interesting

And it let me think of how potential deep learning has even though it is not related to just self driving car

Also makes me more interesting out of deep learning, like reinforcement learning...

This project must need result data, like at this image, I turn steering at 3degree

But almost every data has no result (y) data

So I am very looking forward to learning unsupervised learning

2. All of this code is not written by my own hand

I searched google a lot, of course I did not just copying but changed it as my style, and also know that almost everyone

is same as this situation

But I am doubt about my real coding skill without searching

Can I code all of this level without searching?

? NVIDIA drop out