README

April 18, 2017

1 Project 3: Behavioural Cloning

The objective of this project is to train a model that teaches a car to drive around a track in Udacity's simulator.

```
In [1]: import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        %matplotlib inline
```

This Note book contains info on

1. Files in this repo
2. Dataset characteristics
    - Data Generation
    - How data was captured
    - How the model was trained
3. Solution Design
    - Problem objective
    - Pre-processing of input data
    - Approach taken for designing model architecture
4. Model architecture
5. Refernces

1.1 1. Files in this repo

- `model.py`: Python script to import data, train model and save model.
- `model.json`: Model architecture.
- `model.h5`: Model weights.
- `drive.py`: Python script that tells the car in the simulator how to drive
- `behavioural-cloning.ipynb`: Notebook with all modeles.

1.2 2. Dataset Characteristics

1.2.1 Data Generation:

The model was trained by driving a car in the Udacity's car-driving simulator. Each recorded datapoint comprised 7 attributes: * Image taken from the center camera on the car (320 x 160 pixel image with 3 colour channels) * Image taken from the left camera on the car * Image taken from the right camera on the car * Throttle * Speed * Brake * Steering angle (variable to predict, a float ranging from -1.0 to 1.0 inclusive)

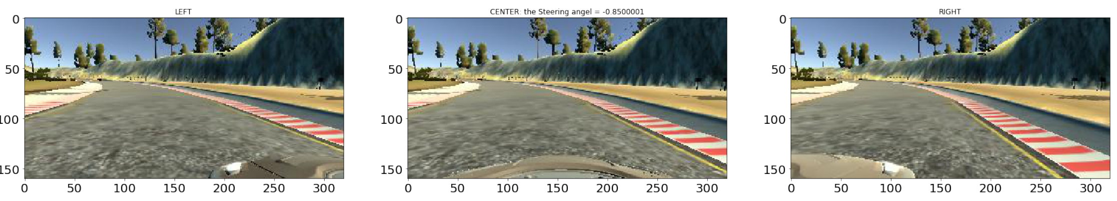
1.2.2 How data was captured

I recorded driving the car around the training track for two to three laps.

1.2.3 How the model was trained

The model was trained using the center left and right images as input (X) and the steering angle as the variable to predict (y). Around 9000 datapoints were used to train the model. I divided these examples into training and validation sets using `sklearn.model_selection.train_test_split` to reduce overfitting.

```
In [7]: # Visualize left, center and right angle camera at the same moment
        img_left = mpimg.imread('./examples/left_2017_04_16_14_40_49_707.jpg')
        img_center = mpimg.imread('./examples/center_2017_04_16_14_40_49_707.jpg')
        img_right = mpimg.imread('./examples/right_2017_04_16_14_40_49_707.jpg')
        angle = -0.8500001

        # Keeping the 640 x 480 perspective
        plt.rcParams["figure.figsize"] = [32, 24]
        plt.tick_params(axis='x', labelsize=25)

        plt.subplot(1, 3, 1)
        plt.imshow(img_left)
        plt.title("LEFT")
        plt.tick_params(axis='x', labelsize=20)
        plt.tick_params(axis='y', labelsize=20)

        plt.subplot(1, 3, 2)
        plt.imshow(img_center)
        plt.title("CENTER: the Steering angel = -0.8500001")
        plt.tick_params(axis='x', labelsize=20)
        plt.tick_params(axis='y', labelsize=20)

        plt.subplot(1, 3, 3)
        plt.imshow(img_right)
        plt.title("RIGHT")
        plt.tick_params(axis='x', labelsize=20)
        plt.tick_params(axis='y', labelsize=20)

        #plt.tight_layout()
        plt.show()
```



1.3 3. Solution Design

1.3.1 Objective

The target is for the car to drive within the lane lines, so the main features the model needs to recognise from the center image are the lane lines.

1.3.2 Pre-processing of input data

The X input fed into the model are the original unprocessed images taken by the camera. They are of size 160x320 pixels with 3 channels (RGB).

Pre-processing steps:

- Crop away parts of the test image that are irrelevant to determining the steering angle (parts that don't include the track).
  - i.e. crop away the top 70 rows of pixels and bottom 25 rows of pixels).
- Normalise image because normalised input data empirically produces more accurate models. The intuition may be to make the model more robust to changes in e.g. lighting.
- Flip the images to collect more traning dataw withoud drive the road in the opst direction

1.3.3 Approach taken for designing model architecture

1. Regression (without Convolutions)
2. CNN (LeNet)
3. End to End Learning for Self Driving Cars

1.3.4 Definitions

Convolutions

- Convolutions generally seem to perform well on images, so I tried adding a few convolution layers to my model.

Activations and Dropout

- Activation to introduce non-linearities into the model: I chose ReLU as my activation.
- ReLU: `h = max(0,a) where a = Wx + b`.
- I added dropout to prevent the network from overfitting (only in LeNet).

Fully connected layer

- I added a fully connected layer after the convolutions to allow the model to perform high-level reasoning on the features taken from the convolutions.

Final layer

- This is a regression and not a classification problem since the output (steering angle) is continuous, ranging from -1.0 to 1.0.
  - So instead of ending with a softmax layer, I used a 1-neuron fully connected layer as my final layer.

1.4 4. Model architecture

I tried many models the first one is the easiest. Network consists of I also tried (1) a scaled-down version of NVIDIA's pipeline and (2) various combinations of conv layers combined with fully connected layers.

The model is a Sequential model comprising three convolution layers and three fully-connected layers. The model weights used were those obtained after training for **20 epochs**.

The model code and specifications are below:

```
model = Sequential()

# Normalization layer
model.add(Lambda(lambda x: x/ 255.0 - 0.5, input_shape=(160,320,3)))
model.add(Cropping2D(cropping=((70,25),(0,0))))

# 5 x 5 Convolution filters + subsample which make the size of the images smaller
model.add(Convolution2D(24,5,5, subsample = (2,2), activation = 'relu', name='conv1'))
model.add(Convolution2D(36,5,5, subsample = (2,2), activation = 'relu', name='conv2'))
model.add(Convolution2D(48,5,5, subsample = (2,2), activation = 'relu', name='conv3'))
# 3 x 3 Convolution filters
model.add(Convolution2D(64,3,3,activation = 'relu', name='conv4'))
model.add(Convolution2D(64,3,3,activation = 'relu', name='conv5'))

model.add(Flatten())
model.add(Dropout(.8))
model.add(Dense(100))
model.add(Dropout(.8))
model.add(Dense(50))
model.add(Dense(10))
# one single output layer
model.add(Dense(1))



adam = Adam(lr=0.0001)

model.compile(optimizer=adam, loss="mse", metrics=['accuracy'])
```

1.5 5. Discussion

1.6 6. Refernces:

1. Self-driving car Nanodegree Program, http://udacity.com
2. End to End Learning for Self-Driving Cars - Nvidia
3. http://yann.lecun.com/exdb/lenet/
4. jessica yung, http://github.com