CarND-BehaviorCloneSubmission

1. Submission packages
model.py - The script used to create and train the model.

drive.py - The script to drive the car.

model.h5 - The saved model.

writeup_report - It explains the structure of the network and training approach.

2. Quick Start
2.1 Prerequisites

All software dependency installation should be based on the previous course material of Udacity self-driving cars. In addition, update the following package:

pip install python-socketio ==1.6.3

pip install keras==1.2.1

2.2 train the model

Run the following command to train the model:

python model.py

The model is trained based on training data stored in the specified directory. To modify the directory, please go to data.py file and change the definition for the variable train_data for the directory of training data. The train data structure should be the same as what the course material provided.

train_data = [("../data_given/data/driving_log.csv", "../data_given/data/IMG")]

If there exists trained weights, the program will auto load the weight as first choice. If not, the weights will be trained from “scratch”.

In default, the weights are stored in the directory “../model/model.h5”

2.3 Check the results of training model

After you run “python model.py”, you will see some training results printed on screen. These results include:

a) model loss to validation data set

b) mean-square-error to validation data set

c) change before and after training for those sampled trainable and non-trainable weights

2.4 Run on simulator by commanding steering from trained model

Run Udacity behavior clone project simulator; choose autonomous mode; select the test track. After the simulator is launched successfully, run

python drive.py

Note: Currently, the trained model works well for the first test track, but not on the second. It needs additional work to make it work on the second. Due to time constraints, this work is not done for submission

3. Code Structure

Including the files required for submissioin (model.py, drive.py), there are two additional ones (data.py, steering_prediction.py).

3.1 Data.py:

This module is for preprocessing data, which consists of four functions:

3.1.1 get_data():

a) This function outputs a dataframe containing the train image path and lable value

b) This function specifies the directory where the data log and images are saved

c) Image path and labled values are read into a dataframe

d) Only center camera image data (CenterImage) and steering (SteeringAngle) are kept for further processing

e) To improve the train image quality and increase number of train data samples, raw image data is processed by removing speed below 20mph and mirroring the data

3.1.2 vgg_processor(img_io, image_size):

a) This function outputs an numpy array for an image

b) read image from img_io, which could be image path or ByteIO

c) resize image to the target size

d) mirror image by using image array transformation

e) use imagnet function to preposs image (RGB->BGR, substact predefined mean)

3.1.3 x_reader(path, image_size):

a) This function outputs image array (4 dimension) for image batch with input of image batch path

3.1.4 data_generator (log, data_size, image_size=(80,80,3), batch_size = 256):

a) This function outputs image array (4 dimension) and labels (numpy array)

b) it is a infinite batch data generator

c) the generator is consumed by model.fit_generator and model.predict_generator in model.py

3.2 model.py

This function use 7 steps to define the model:

1) Retrieve training data:

the data log (path and lable values) is read from data.py; it constructs train data in a format of data generator by calling data generator function in data.py. The train data is further split as train and validation data set. Since all these data are from the first test track, the proogram does not distinguish additonal test data neither test against that.

2) Extract features from VGG model:

VGG model features are extracted up to the layer of "block5_conv3". The weights in those extracted layers are set non-trainiable such that we can reuse the weights of VGG model trained from imagenet.

3) Modify the last several layers:

The output of "block5_conv3" is processed through pooling, drop out, batch normalization. Then output is further processed through 3 layers with activation function "elu". The final output is a scalar as prediction of steering angle.

4) Train model:

Compile the model with loss function as mean square error, and optimizer as adam. Then train the model with input of data generator.

5) Check training results

Model prediction is checked agaist validation data set with mean square error in range of [0.013-0.016]. No check on test data set.

6) Save model:

Model is save in a format of .json to the specified directory.

7) Save weights:

Weights is save in a format of .h5 to the specified directory.

3.3 steering_prediction.py

This module is for predicting steering with input of current image

Input: image array (3 dimension)

Output: function self.predict to predict the steering given image input

3.4 drive.py

This module has minimum modification from what Udacity provided.

4 model architecture

The model in this submission adopts the VGG16 architecture with the last few layers modified for this purpose. Due to time constraint, no other model structures is tested. The author may explore other models in future. The model is summarized as follows:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 80, 80, 3)	0	
block1_conv1 (Convolution2D)	(None, 80, 80, 64)	1792	input_1[0][0]
block1_conv2 (Convolution2D)	(None, 80, 80, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 40, 40, 64)	0	block1_conv2[0][0]
block2_conv1 (Convolution2D)	(None, 40, 40, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 40, 40, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 20, 20, 128)	0	block2_conv2[0][0]
block3_conv1 (Convolution2D)	(None, 20, 20, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 20, 20, 256)	590080	block3_conv1[0][0]
block3_conv3 (Convolution2D)	(None, 20, 20, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 10, 10, 256)	0	block3_conv3[0][0]

block4_conv1 (Convolution2D)	(None, 10, 10, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 10, 10, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Convolution2D)	(None, 10, 10, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 5, 5, 512)	0	block4_conv3[0][0]
block5_conv1 (Convolution2D)	(None, 5, 5, 512)	2359808	block4_pool[0][0]
block5_conv2 (Convolution2D)	(None, 5, 5, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Convolution2D)	(None, 5, 5, 512)	2359808	block5_conv2[0][0]
averagepooling2d_1 (AveragePooli	(None, 2, 2, 512)	0	block5_conv3[0][0]
dropout_1 (Dropout)	(None, 2, 2, 512)	0	averagepooling2d_1[0][0]
batchnormalization_1 (BatchNorma	(None, 2, 2, 512)	2048	dropout_1[0][0]
dropout_2 (Dropout)	(None, 2, 2, 512)	0	batchnormalization_1[0][0]
flatten_1 (Flatten)	(None, 2048)	0	dropout_2[0][0]
dense_1 (Dense)	(None, 4096)	8392704	flatten_1[0][0]
dropout_3 (Dropout)	(None, 4096)	0	dense_1[0][0]
dense_2 (Dense)	(None, 2048)	8390656	dropout_3[0][0]
dense_3 (Dense)	(None, 2048)	4196352	dense_2[0][0]
dense_4 (Dense)	(None, 1)	2049	dense_3[0][0]

Trainable params: 25,702,401
Non-trainable params: 9,996,096

5 Miscellany

5.1 Train data acquisition

The model uses the train data provided by udacity. I believe these data are good quality. As i tried to use my own train data, the prediction does not work very well. One reason may be that the data obtained by myself have more steering noise.

5.2 data generator

With my current computer setting (one GTX1080 GPU + 32G RAM) and the amount of the data (downloaded from udacity), I did not see any big difference for the training performance using data generator or not. I started with normal way to train the model without data generator. Then I spent 3 more days (+10 hours) to figure out what's data generator, how to use, make the script, redebug my code, and integerate the data generator to my current models. I have no doubt it will perform better with more data due to less demanding to the memory. Suggest Udacity to include the use example of data generator in course material.

5.3 problems during train

1) Typically, the loss for train data and validation data gets smaller and smaller as training iteration increases. However,sometimes, due to unknow reason, the loss is reset to very large number and then becomes smaller again. The same problem does not shows up if the input data is normal data (not data generator)

2) The train data should have good quality. Otherwise, the vehicle may drive off the track during autonomous mode.

3) As being pointed out, the model does not predict very well on the second test track. It needs additional work to make it work on the second test track.