Behavioral Cloning

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- aggregator.py – takes a folder containing subfolders of driving training data and aggregates them into a single driving_log.csv with correct references to image files
- preprocess.py – cropping and normalization of image
- data.py – Referenced in the training process – the main objective is to read in training data and preprocess it; retrieve_generator() returns a generator of data that can be used for training
- model.py containing the script to create and train the model
- model_runner_7.py – a script used to store parameters for training, calls model.py and directs the location of output models
- drive.py for driving the car in autonomous mode
- final_model.h5 containing a trained convolution neural network

- writeup_report.md or writeup_report.pdf summarizing the results

2. Submission includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py final_model.h5
```

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with filter sizes [1x1, 3x3, 5x5]. The first layer is a 1x1 convolution with output depth = 3 in order to determine the best color map representation of the image, followed by depths ranging between 18-27. It can be found in nvidia_model() under model.py – the model was largely based off the Nvidia model described in the preparatory lesson, though changes were made to accept a smaller image input, and subsequently the depth of the convolution layers were changed to ensure that the model size remained within acceptable size parameters.

Additionally, elu activation functions were used in place of ReLU's which have been proven to be consistently outperformant in many research applications, and dropouts were added in the fully connected network at the end for reducing overfitting.

Data was normalized through functions in preprocess.py, and is referenced in data.py (for data generation) and drive.py.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 30 onwards).

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py:164 calls two separate generators that retrieve shuffled non-overlapping data). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

Early stopping of the model was also employed; the final model was trained up to 5 epochs, though it was found that the 1st epoch produced the best result and hence this was taken/used for submission.

3. Model parameter tuning

The model used an Adam optimizer, so the learning rate was not tuned manually during training, though the starting learning rate was set to 0.0005. It was found that the validation data scores were largely irrelevant despite this being part of the initial core strategy to reduce overfitting because to some extent driving around a track requires some amount of overfitting of the corners, preferred over a strategy that might be simpler such as driving straight most of the time.

Nadam was experimented with, though showed no noticeable improvement and was hence not used in latter stages of training/parameter tuning.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road, biased to a steering offset to the center of 0.15 of the total one-side steering range (i.e. 0.15 of 25 degrees).

The collected data had many examples that depicted driving straight; as such, the training data had to be biased upon generation towards examples that depicted larger turning angles.

For details about how I created the training data, see the next section.

#TODO NEXT SECTION

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to iteratively improve a model starting from a known model proven to work already with self driving cars.

In this case, that meant step 1 was to implement the Nvidia model introduced in the preparatory lessons. Then, using limited training data and minimal preprocessing, the model was trained to see if it was responsive to learning some of the turns of the track. Also, validation and training error was closely monitored as another quick-feedback method of gauging if the model as a starting point was sound.

A hyperparameter training search was then conducted across # epochs, the type and learning rate of the optimizer, and augmented with varying amounts of acquired data. Initially, regardless of how this search occurred, the model always seemed to be biased towards straight data – even with augmented with recovery data and left/right camera images with biased steering angles for recovery.

In order to explore more hyperparameters, image size was reduced and cropped to only process the road image section; this reduced training time, though resulted in few gained insights.

Finally, preprocessing was heavily reviewed, eventuating with a generator that was biased to output training data that had a large steering angle. Additionally, because validation MSE proved to be a non-insightful metric after much experimentation, a more robust metric was coded – it sampled images with high turning degrees and calculated the error in trying to predict steering angles for these turns. This metric, whilst coded, was not used as the preprocessing experimentation yielded a working model right away – though it would certainly be part of the next step of training, as the feedback time of understanding if the model was performant was taking too much time as a manual process of running the simulator.

Overfitting was somewhat combat between adding different probabilities of dropout and early stopping. Interestingly, initially when little preprocessing was done, training was conducted for an entire day on an AWS GPU instance and the resultant model led the car to drive predominantly straight. Overfitting was only really observed to the end once preprocessing was performed correctly and a model trained over a high number of epochs caused the car to drive ‘jittery’. Still, the car was able to drive around the track in this ‘overfit’ scenario.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 14-46) consisted of a convolution neural network with the following layers and layer sizes ...

Here is a is a summary of the architecture as printed out by the summary() method of the Keras model:

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
color_conv (Convolution2D)       (None, 18, 64, 3)     12          convolution2d_input_2[0][0]
____________________________________________________________________________________________________
convolution2d_6 (Convolution2D)  (None, 18, 64, 24)    1824        color_conv[0][0]
____________________________________________________________________________________________________
convolution2d_7 (Convolution2D)  (None, 18, 64, 36)    21636       convolution2d_6[0][0]
____________________________________________________________________________________________________
convolution2d_8 (Convolution2D)  (None, 14, 60, 48)    43248       convolution2d_7[0][0]
____________________________________________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 7, 30, 48)     0           convolution2d_8[0][0]
____________________________________________________________________________________________________
convolution2d_9 (Convolution2D)  (None, 5, 28, 64)     27712       maxpooling2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_10 (Convolution2D) (None, 3, 26, 64)     36928       convolution2d_9[0][0]
____________________________________________________________________________________________________
flatten_2 (Flatten)              (None, 4992)          0           convolution2d_10[0][0]
____________________________________________________________________________________________________
dropout_4 (Dropout)              (None, 4992)          0           flatten_2[0][0]
____________________________________________________________________________________________________
dense_5 (Dense)                  (None, 800)           3994400     dropout_4[0][0]
____________________________________________________________________________________________________
dropout_5 (Dropout)              (None, 800)           0           dense_5[0][0]
____________________________________________________________________________________________________
dense_6 (Dense)                  (None, 100)           80100       dropout_5[0][0]
____________________________________________________________________________________________________
dropout_6 (Dropout)              (None, 100)           0           dense_6[0][0]
____________________________________________________________________________________________________
dense_7 (Dense)                  (None, 50)            5050        dropout_6[0][0]
____________________________________________________________________________________________________
dense_8 (Dense)                  (None, 1)             51          dense_7[0][0]
====================================================================================================
Total params: 4,210,961
Trainable params: 4,210,961
Non-trainable params: 0
```

Number of layers = 15

Number of convolutional layers = 6

Number of fully connected layers = 4

Other layers include max-pooling, flattening and dropout.

3. Creation of the Training Set & Training Process

The process undertook for creating the training set data was very rigorous, but much of the process was discarded in the final model. The car was driven around the track over numerous laps, equivalent to driving at full speed for around 45 minutes. Then, many recovery scenarios were captured, starting from when the car was at the side of the road and orientating the steering wheel towards the center BEFORE recording was initiated.

Data was continuously augmented to observe the impact of the resultant model, and it was observed that so much data was being acquired that overfitting was a real possibility – training could iterate over 500,000+ examples. Hence, at this point, the

model was refactored to take as an input an epoch size parameter for training, at which the generator would yield a randomized set of data of this size before continuing with latter epochs.

After exhausting many hyper-parameter combinations for training, more attention was paid to preprocessing the data. This included cropping of the images to the most relevant sections of the road, resizing to a smaller image, biasing the data to learn curves over straight lines and brightness adjustments (randomized changes to the images).

It should be noted that the data was prepared with mirroring the steering angle and lateral image orientation in order to eliminate a non-zero steering bias.

Examples of the kinds of data captured in the data acquisition process are as follows:

- Well-behaved driving at the center of the road

- Turn recovery from the edges of the road

- Heavy iteration over acute angle corners