Behavioral Cloning: Train a CNN to drive

The goal of the behavioral cloning is to train a CNN to drive autonomously in a simulator. The first step is to collect data of good driving behavior data using the simulator, i.e. left, center, and right images captured by three on board cameras as well as corresponding steering angles. Due to the fact it's not easy to produce a "good" driving behavior data using keyboard, I used the provided training data. One CNN can be trained to take the images from the simulator and send the steering angles back to the simulator.

After I finished reading the suggestions from a previous student [1], I immediately wonder why we need to do such a big training using a 9-layer CNN architecture considering the driving road condition in the simulator isn't complex. We don't have so many features to train compared with NVIDIA's End to End Learning for Self-Driving Cars scenarios [2]. I understand a sophisticated CNN is needed considering there're plenty features in the real world driving, but don't feel it's necessary intuitively. I searched on-line and found a project did by Mengxi Wu [3], which is exactly the idea and got successfully achieved. I was amazed by how good performances his training results were by using only 2 layers with 63 parameters. I followed his philosophy and achieved satisfied training results.

Files Submitted & Code Quality

My project includes the following files:

- model_small.ipynb containing the script to create and train the model
- drive_small.py for driving the car in autonomous mode
- model_small.h5 containing a trained convolution neural network
- Behavioral Cloning.pdf summarizing the process

Using the Udacity provided simulator and my drive_small.py file, the car can be driven autonomously around the track by executing python drive_small.py model_small.h5. The changes of the provided drive.py are the resizing of the simulator input images and the PI controlled speed.

Model Architecture

As said, the CNN architecture was from [3] having 6 layers with only 63 parameters in total:

```
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 16, 32, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 30, 2)         20
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 7, 2)           0
_________________________________________________________________
dropout_1 (Dropout)          (None, 3, 7, 2)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 42)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 43
=================================================================
Total params: 63
Trainable params: 63
Non-trainable params: 0
```

- The first layer is normalization of the inputs between [-1, 1]
- The second layer is the 2D convolution with kernel size of $(3, 3)$, and followed by ReLU activation
- The third layer is Max pooling layer with kernel size of $(4, 4)$ and the valid padding
- The forth layer is Dropout to avoid overfitting
- The fifth layer is data Flatten.
- The sixth layer is Dense layer with 1 neuron to produce the output steering angles

Training Strategy

The training data compose of the provided left, center, and right images (features) as well as the corresponding steering angles (labels) in the driving_log.csv file.

The left and right images were used to teach the CNN to recover from the offsets. The steering angles were added by a constant 0.3 for the left images and subtracted a constant 0.3 for the right images.

In order to avoid one class dominating the CNN training, i.e. the training data has more turns to one side than to the other side, more images were generated by flipping the existing left, center, and right images.

The CNN was trained with a batch size = 128 and epoch = 10. The training was conducted by using Adam as the optimizer and mean squared error as the loss evaluations. Due to the small CNN architecture and the images got resized, all the preprocessed training data can be stored in memory all at once. Thus, using a generator to pull pieces of the data and process them on the fly when you need them is not necessary.

Reflection

It’s quite amazing the goal can be realized by a CNN with so simple architecture. Even it’s mainly because the very limited situations in the simulator. For the real world driving, the NVidia models [2] is necessary, not mention other tasks such as traffic sign, other cars, and pedestrian detections have to be conducted simultaneously.

However, the intuition I learned from the project can be quoted from Dima Yanchenko [4]: “It is more about the quality of your data—do your data fully (or with 99.6% confidence) represent the population of your environment—and the right model capacity to fit into the regularities in your task environment. Sometimes, putting more complexity into the model will push the model to learn unnecessary, irrelevant information, which will noise and bias your model decision making. And trying to generalize from that, using rough technics like dropout, weight regularization and etc. may become an intensive and less predictable task, with no confidence in robustness of results”.

I am still working on the project because the field self-driving is a much more complex task, and the data argumentation and batch generator skills are essential to solve practical problems. I found followed [5] and conducted a very comprehensive and effective data augmentations. However, the results were not satisfying due to NN black box nature. I will update my posting after I achieve a good results.

References

[1] Behavioral Cloning Cheatsheet by Paul Heraty (https://slack-files.com/T2HQV035L-F50B85JSX-7d8737aeeb)

[2] Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel LD, Monfort M, Muller U, Zhang J, Zhang X. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316. 2016 Apr 25.

[3] Self-driving car in a simulator with a tiny neural network (https://medium.com/@xslittlegrass/self-driving-car-in-a-simulator-with-a-tiny-neural-network-13d33b871234)

[4] Dima Yanchenko (https://medium.com/@DYanchenko/hi-mengxi-wu-620f6091dac0)

[5] An augmentation based deep neural network approach to learn human driving behavior (https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9)