Behavioral Cloning

Behavioral Cloning Project

I'll start stating that my original report (17 pages long) was accidentally deleted and unfortunately I will not go through the comparison and assessment of the results between the VGG16 pre-trained model and NVIDIA final results nor between CommaAI model and NVIDIA. This report, unfortunately, will go straight to the point without addressing the really valuable learning lessons I went through after 45 days on it...including almost 3 days of report writing.

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

In the following section 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:

- ***model.py*** containing the script to create and train the model
- ***utils.py*** containing additional functions used in "***model.py***"
- ***CNN_Architectures.py*** containing the 3 model architectures tested in this project
- ***drive.py*** for driving the car in autonomous mode
- ***model.h5*** containing a trained convolution neural network
- ***Behavioral Cloning report.pdf*** summarizing the results

- ***RunTrack1.mp4*** Successful lap around track 1
- ***RunTrack2.mp4*** Successful lap around track 2

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 model.h5
```

3. Submission code is usable and readable

The ***model.py*** file contains the code for modeling, training, validating and saving the convolution neural network and also comments to explain how the code works.

Model Architecture and Training Strategy

1. Introduction

Due to the learning nature of this process I decided to investigate 3 different (and relatively simple) architectures to:

1. Learn in more depth their architectural details, their different complexities, their pros & cons in tuning, memory usage, training time, etc...
2. Evaluate and compare their performances
3. Identify, if possible, their limitations.

The 3 models developed were:

1. NVIDIA Model (following the “End to End Learning for Self-Driving Cars” paper submitted on April 26, 2016
2. Comma AI model (from “https://github.com/commaai/research”)
3. VGG16 (from keras.applications.vgg16 library)

In this section I will describe each model in detail but before that I will discuss the “common” aspects of all models

2. Common factors

There are a few very important aspects in the development, training and testing of a CNN that are common to all model.

2.1. Data Pre-processing: Avoiding the "Garbage In – Garbage Out" problem

In my original document I went extensible with the unprecedented and totally underestimated importance of the preprocessing; specifically I addressed:

- Cropping: How useful is to determine what pixels are truly bringing valuable information and what are not..or even could damage the learning process. Example the sky, the sides, the hood. I also went into details on how I chose the cropping windows for each model.
- Color Vs gray: I went into the whole debate about how color should only be used when "color" has a key "conceptual" value in the learning process as it actually does in this case. I compared the difference between a CNN to identify and classify letters where color has no semantic value and how in other cases like trying to identify "dogs" for instance color might be a key factor since you are not likely to find a purple dog for example. In our case I performed training with gray images and proved that actually the cross over validation error was significantly higher and that despite the fact that the training was significantly faster since we use only a single channel rather than 3, the results were unacceptable proving the importance of color
- Random image brightness and horizontal/vertical shifts. I went long comparing how significant were the changes when using this technique and much the results went from total failure (off into the sand) Vs finishing the laps on both tracks.

2.2. The Generator's miracle

Another advice we got in our lectures, to the point of it practically being a requirement, was to use Python generators and in particular Keras ImageDataGenerator and the fit_generator function.

I started creating a very simple Generator (data_gen1) following and building upon Udacity's example. This one was just to test how it works. For this generator I used only Udacity image data.

I followed this development with "data_gen2". On this generator, I used all the data I collected and I build my own generator following Udacity's example.

"data_gen3" used exclusively Keras generators and I wasn't satisfied with it since I had to load all the images before feeding them to the "fit".

You'll miss the very long discussion describing all 3 and why I ended up using the second one. I can't help but tell you to just look at the code.

NVIDIA Model

This CNN model has been developed on "Build_NVDIA( )" function following the "End to End Learning for Self-Driving Cars" paper submitted on April 26, 2016 by NVIDIA.

Lots and lots missing here. Discussion regarding why I switched to eLu instead of staying with Relu..explaining the shortcomings of Relu (Dead activations for negative values... and only positive outputs when triggered are some of them) The other major discussion I wrote was about how I had to eliminate the DropOut layers that only made the response and the driving of the car flat/straight. I compared the results and proved that Dropout, even though is one of the primary methods used to avoid overfitting, didn't really work in this case. I later tried "SpatialDropout2D" after reading some other students' recommendations and, in an empirical study – like I always approached this assignment – I proved that did not help either by showing both cross-over validation errors. Unfortunately you'll never know how deep I went into this and how long it took me to discover the huge differences that this changes made. (I'm very frustrated and angry and you'll feel this through my writing)

I went into discussing why adding the cropping into the CNN instead of doing so as part of the pre-processing might have been an advantage. Why? Mainly because of the efficient use of memory and resources, especially if you are using GPU. Another amazing deep discussion you'll never read and most importantly, you'll never know how much I work put into all this.

Similarly as with the cropping, the normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing.

Lots of details are missing but nothing I'm willing to do at this point..too much effort too much time. For details look at CNN_Architecture.py

2. Attempts to reduce overfitting in the model

Deeper discussion and comparison regarding why I eliminated the DropOut layers.

The other mayor point was the splitting and shuffling of data shown in "Data_setup" function on "Model.py"

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually. I went on a long discussion about:

- At tried actually change the learning rate to 0.0001 instead of the default initial one 0.001 and had to stop training after 24 hrs.
- Played with padding and showed/compared the results. Last one included in the architecture is the successful one.
-

4. Appropriate training data

After 45 days generating different types (augmented) data and trying to understand what and why is "good" data I can summarize the more than 2 pages I dedicated to this as:

- Using left/right cameras was a double edge sword.
    - Created "good" data and helped me balance (same amount of straight, left turns and right turns driving) all the data I collected.
    - It introduced a new parameter: the Steering Correction. Which added complexity and difficulty tuning the system
    - It proved to make the car swerve more while driving and I explained my hypothesis of non continues prediction since we added a hard-coded value for the correction that not always transition smoothly between the real driving and these added/augmented data.
- Splitting
- Random pre-processing (Brightness and Horizontal/Vertical shifting)
-

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to use already designed models to learn from their strengths and weaknesses. To do so, a profound understanding of what each parameter and layer would do and to understand how sensitive the entire CNN output is to changes in each and every parameter.

My first step was to use a convolution neural network model similar to the NVIDIA, VGG16 and CommaAI. I thought these models, especially NVIDIA and CommaAI might be appropriate because they were used in the past for exactly the same purposes.

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track... to improve the driving behavior in these cases, I ...

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 consisted of a convolution neural network with the architecture you can find on CNN_architectures.py (Build_NVIDIA)

3. Creation of the Training Set & Training Process

To capture good driving behavior, I recorded:

- Track 1. Normal driving (8,976 frames)
- Track 1 Recovery: Different recovery scenarios (2,367 frames)
- Track 1 Recovery Reverse: Different Recovery scenarios while driving on the opposite direction on normal driving. (2,394 frames)
- Track 1 Reverse: Normal driving on the opposite direction. (9,252 frames)
- Track 1 Validation: Initially this was a normal driving data to be used for validation that later was included as training data (validation data was randomly selected from all the training data) (2,880 frames)
- Track 2 Normal: Normal driving on track 2. (19,272 frames)
- Track 2 Validation: As on track 1, this data was supposed to be used for validation but ended up being included on the training data for later random splitting. (2,922 frames)
- Augmented folder: This folder was used to hold all the augmented data that was not generated on the fly, for data “balancing”: Creating enough samples off all 3 main driving condition: Straight, left and right turns. The straight driving introduced a new parameter; the Straight driving threshold that allowed to consider “straight” driving even when we were “slightly” steering between –x and +x degrees.

After the collection process and the augmentation (please refer to “data_setup” function on Model.py), I had about 94,500 data points. I then preprocessed this data by adding random brightness and shifting (as mentioned above)

I finally randomly shuffled the data set and put, as it’s common in these cases, 20% of the data into a validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was between 7-9 as evidenced by the use of early stop since after these EPOCHS it was no improvement. I used an adam optimizer so that manually training the learning rate wasn't necessary.

In my original report I dedicated the last 2 sections to talk about the "takeaways" and the "Challenges"..which I refuse to do again after so much time and effort put into this.

Note: I collected all the figures of my training sessions and I can provide a deeper explanation of some results if needed. I'm just too burnt out after 45 days on this plus losing the fruits of my work - Report