

Behavioral Cloning

Overview

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

Quality of code

Is the code functional?

My submission includes the following files:

- `cloning.py` containing the script to create and train the model
- `drive.py` for driving the car in autonomous mode
- `model_nvidia_1counterclock.h5`, which contains a trained convolution neural network
- this report discussing on the results and procedure
- `TrainingDataPictures` folder containing sample pictures of the dataset
- `driving_log.csv` file, result of the collection data process on the simulator

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

```
python drive.py model_nvidia_1counterclock.h5
```

Is the code usable and readable?

You can see in the file `cloning.py` how I trained my model. I conducted my model training, by storing the training data in memory.

Model architecture and training strategy

Has an appropriate model architecture been employed for the task?

At first, I implemented the LeNet architecture, but I couldn't get my car drive in the track autonomously. I continued with implementing the Nvidia neural network architecture;

The network consists of a normalization layer (I used a lambda layer), followed by 5 convolutional layers, followed by 4 fully connected layers.

```
model.add(Lambda(lambda x: x/255.0 - 0.5, input_shape=(160, 320, 3)))
model.add(Cropping2D(cropping=(50,25),(0,0)))
model.add(Convolution2D(24, 5, 5, subsample=(2,2), activation = 'relu'))
model.add(Convolution2D(36, 5, 5, subsample=(2,2), activation = 'relu'))
model.add(Convolution2D(48, 5, 5, subsample=(2,2), activation = 'relu'))
model.add(Convolution2D(64, 3, 3, activation = 'relu'))
model.add(Convolution2D(64, 3, 3, activation = 'relu'))
model.add(Flatten())
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
```

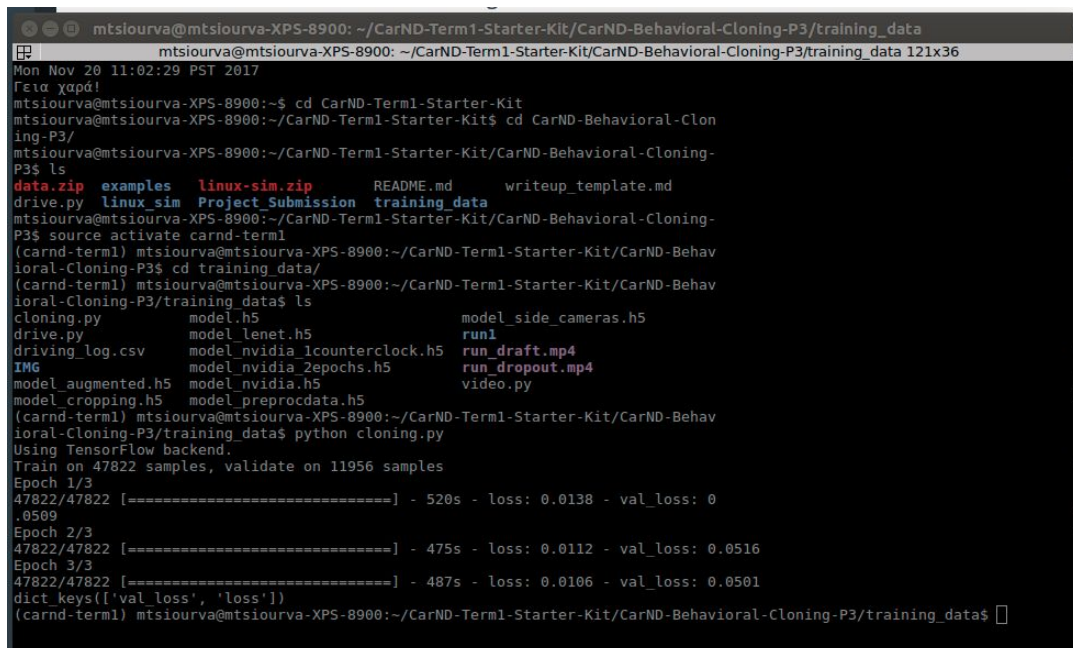
Non-linearity was introduced by using ReLUs as activation functions. You can see the filter sizes above as well.

I used also the `Cropping2D` layer provided by Keras for image cropping within the model (cropped 50 rows pixels from the top and 20 rows pixels from the bottom), so as to discard information that the images capture and I don't need (sky, hood of the car, etc.), and train my model faster. The cropping is done within the model, because is relatively faster than doing it outside of it (the model is parallelized on the GPU).

By using the Nvidia architecture, I managed to drive the car autonomously in the first track. You can find the output video file in the submission folder.

Has an attempt been made to reduce overfitting of the model?

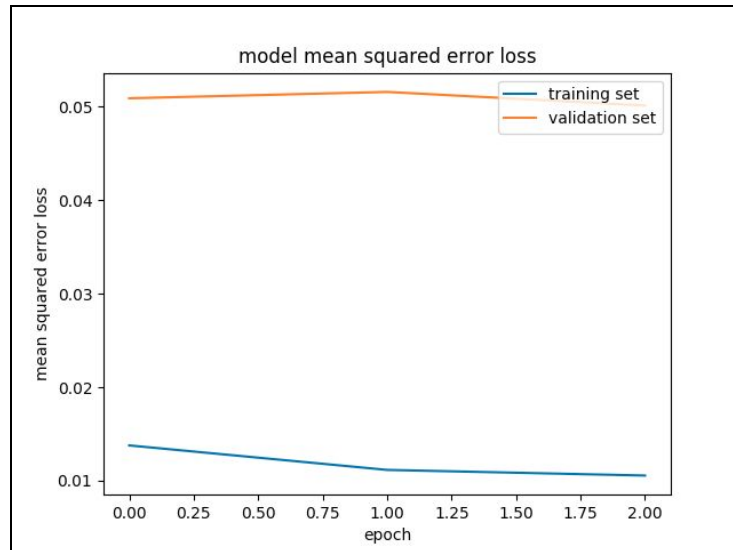
In order to prevent overfitting, I shuffled the data and used 20% of them for validation. I also reduced the number of epochs to 3 (the default number of epochs in Keras is 10). I tried using dropout layers as well, but that didn't work out very well for me.



```
mtsiourva@mtsiourva-XPS-8900: ~/CarND-Term1-Starter-Kit/CarND-Behavioral-Cloning-P3/training_data
mtsiourva@mtsiourva-XPS-8900: ~/CarND-Term1-Starter-Kit/CarND-Behavioral-Cloning-P3/training_data 121x36
Mon Nov 20 11:02:29 PST 2017
Feia Xapdi
mtsiourva@mtsiourva-XPS-8900:~$ cd CarND-Term1-Starter-Kit
mtsiourva@mtsiourva-XPS-8900:~/CarND-Term1-Starter-Kit$ cd CarND-Behavioral-Cloning-P3/
mtsiourva@mtsiourva-XPS-8900:~/CarND-Term1-Starter-Kit/CarND-Behavioral-Cloning-P3$ ls
data.zip  examples  linux-sim.zip  README.md  writeup_template.md
drive.py  linux_sim  Project_Submission  training_data
mtsiourva@mtsiourva-XPS-8900:~/CarND-Term1-Starter-Kit/CarND-Behavioral-Cloning-P3$ source activate carnd-term1
(carnd-term1) mtsiourva@mtsiourva-XPS-8900:~/CarND-Term1-Starter-Kit/CarND-Behavioral-Cloning-P3$ cd training_data/
(carnd-term1) mtsiourva@mtsiourva-XPS-8900:~/CarND-Term1-Starter-Kit/CarND-Behavioral-Cloning-P3/training_data$ ls
cloning.py  model.h5  model_side_cameras.h5
drive.py    model_lenet.h5  run1
driving_log.csv  model_nvidia_1counterclock.h5  run_draft.mp4
IMG           model_nvidia_2epochs.h5  run_dropout.mp4
model_augmented.h5  model_nvidia.h5  video.py
model_cropping.h5  model_preprocdata.h5
(carnd-term1) mtsiourva@mtsiourva-XPS-8900:~/CarND-Term1-Starter-Kit/CarND-Behavioral-Cloning-P3/training_data$ python cloning.py
Using TensorFlow backend.
Train on 47822 samples, validate on 11956 samples
Epoch 1/3
47822/47822 [=====] - 520s - loss: 0.0138 - val_loss: 0.0509
Epoch 2/3
47822/47822 [=====] - 475s - loss: 0.0112 - val_loss: 0.0516
Epoch 3/3
47822/47822 [=====] - 487s - loss: 0.0106 - val_loss: 0.0501
dict_keys(['val_loss', 'loss'])
(carnd-term1) mtsiourva@mtsiourva-XPS-8900:~/CarND-Term1-Starter-Kit/CarND-Behavioral-Cloning-P3/training_data$
```

Instead, I focused on collecting more data, by driving the car more rounds, driving it smoothly and on the road, driving it once counterclockwise in the track, and recording recovery driving data when in turns.

Eventually, I ended on training 47822 samples, and validating on 11956 samples.



Have the model parameters been tuned appropriately?

For the loss function I used Mean Squared Error. Here we have a regression network and not a classification network, we want to minimize the error between the steering measurement that the network predicts and the ground truth steering measurement. I also used an Adam optimizer;

```
model.compile(loss = 'mse', optimizer = 'adam')
```

Is the training data chosen appropriately?

In order to expand the model and help it generalize better, I augmented the dataset by flipping the images around y-axis and by inverting the steering angles (the track makes the car have a left turn bias) .

I also included steering measurements from the side cameras by creating adjusted steering measurements with a correction of 0.2.

The lines of the relevant code are 14-38 of the `cloning.py` file. As mentioned above I included `Cropping2D` within the model to discard irrelevant data information.

Architecture and Training Documentation

Is the solution design documented?

Is the model architecture documented?

Below, you can see my notes and steps towards my final implementation;

Tried with 7 epochs and 10 epochs

low mean squared error for training set, high mean squared error for validation set → overfitting

Steps to improve data

1. Preprocess the data

a. Normalizing the data → add Lambda layer to the model

Within this lambda layer I'll normalize the image by dividing each element by 255 (the maximum value of an image pixel)

b. mean - centering the data

mean - center the image by subtracting 0.5 from each element

Training and validation loss much smaller after this step, but model seems still to overfit

2. Add a convolution

Implementing LeNet architecture

1 output instead in the end, to predict steering angle

```
model.add(Convolution2D(6, 5, 5, activation = 'relu'))
model.add(MaxPooling2D())
model.add(Convolution2D(16, 5, 5, activation = 'relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(120))
model.add(Dense(84))
model.add(Dense(1))
```

Car pulling left, but driving is more smooth

Most of the time the model learns to steer to the left

How to address this problem??? By augmenting the data, help the model generalize better

Data augmentation and using side cameras do the same job

Approach: flipping the images, invert steering angle, teach the model to drive counter-clockwise as well

Benefits: more data, more comprehensive data

Why do we need three camera shots for each point in time?

It's simply more data.

We can use these camera shots to help us generalize our model.

It will teach our network how to steer back to the center.

Saw that the validation loss was increasing with every epoch→ reduced number of epochs to 3

Cropping Images with Cropping2D

Implementing more powerful network→ Nvidia

<https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/>

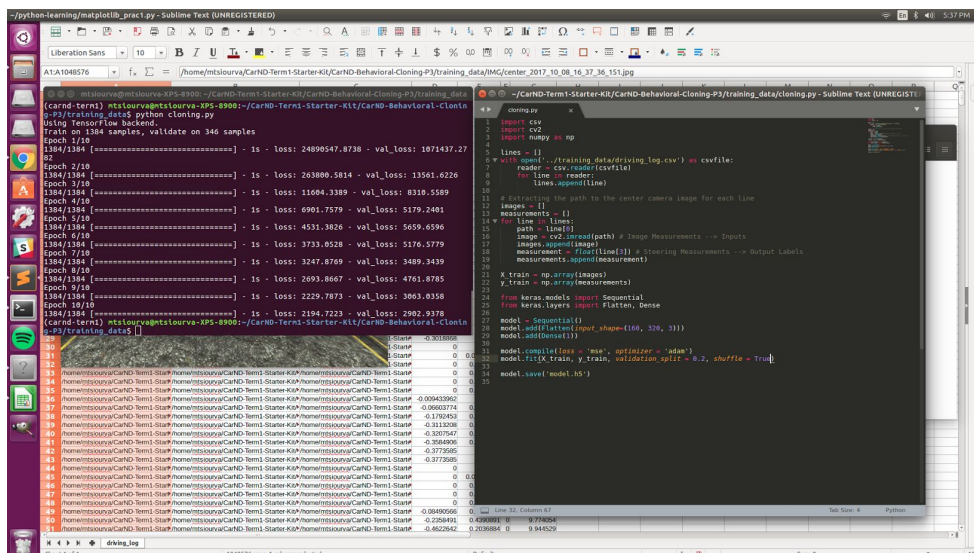
```
model.add(Convolution2D(24, 5, 5, subsample=(2,2), activation = 'relu'))
model.add(Convolution2D(36, 5, 5, subsample=(2,2), activation = 'relu'))
model.add(Convolution2D(48, 5, 5, subsample=(2,2), activation = 'relu'))
model.add(Convolution2D(64, 3, 3, subsample=(2,2), activation = 'relu'))
model.add(Convolution2D(64, 3, 3, subsample=(2,2), activation = 'relu'))
model.add(Flatten())
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
```

I recollected data, I corrected my driving, and then I was able to drive the track autonomously

But still validation loss kept rising

So I collected another lap of data, and result improved

The above are just notes that document the process I followed. You can see as well a picture of my desktop at the earliest steps of the project. For final tuning parameters refer to the rest of the writeup, as well as the `cloning.py` file.



Is the creation of the training dataset and training process documented?

Below you can see a random set of examples of my training data. The pictures are located in the submission folder (TrainingDataPictures folder) as well. As I said above, I collected more data, by driving the car more rounds, driving it smoothly and on the road, driving it once counterclockwise in the track, and recording recovery driving data when in turns.

Eventually, I ended on training 47822 samples, and validating on 11956 samples (these include the augmented data). You can find the driving_log.csv file in my submission.

Right



Left



Center



Is the car able to navigate correctly on test data?

Please find my video output in the submission folder.