Behavioral Cloning

Writeup, David Lopez Rubio

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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:

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- 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 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

```
_________________________________________________________________
Layer (type)                 Output Shape              
Param #
```

```
=================================================================
==========
cropping2d_1 (Cropping2D)    (None, 90, 320, 3)
0
_________________________________________________________________
_____________
lambda_1 (Lambda)            (None, 90, 320, 3)
0
_________________________________________________________________
_____________
conv2d_1 (Conv2D)            (None, 43, 158, 24)
1824
_________________________________________________________________
_____________
conv2d_2 (Conv2D)            (None, 20, 77, 36)
21636
_________________________________________________________________
_____________
conv2d_3 (Conv2D)            (None, 8, 37, 48)
43248
_________________________________________________________________
_____________
conv2d_4 (Conv2D)            (None, 6, 35, 64)
27712
_________________________________________________________________
_____________
conv2d_5 (Conv2D)            (None, 4, 33, 64)
36928
_________________________________________________________________
_____________
flatten_1 (Flatten)          (None, 8448)
0
_________________________________________________________________
_____________
dense_1 (Dense)              (None, 100)
844900
_________________________________________________________________
_____________
```

```
dense_2 (Dense)              (None, 50)
5050
_________________________________________________________________
_____________
dense_3 (Dense)              (None, 10)
510
_________________________________________________________________
_____________
dense_4 (Dense)              (None, 1)
11
=================================================================
==========
Total params: 981,819
Trainable params: 981,819
Non-trainable params: 0
```

The model is the Nvidia suggested on lectures https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/

Has a first layer that crops images (helps reducing data size, more training speed)

Lambda normalizes images

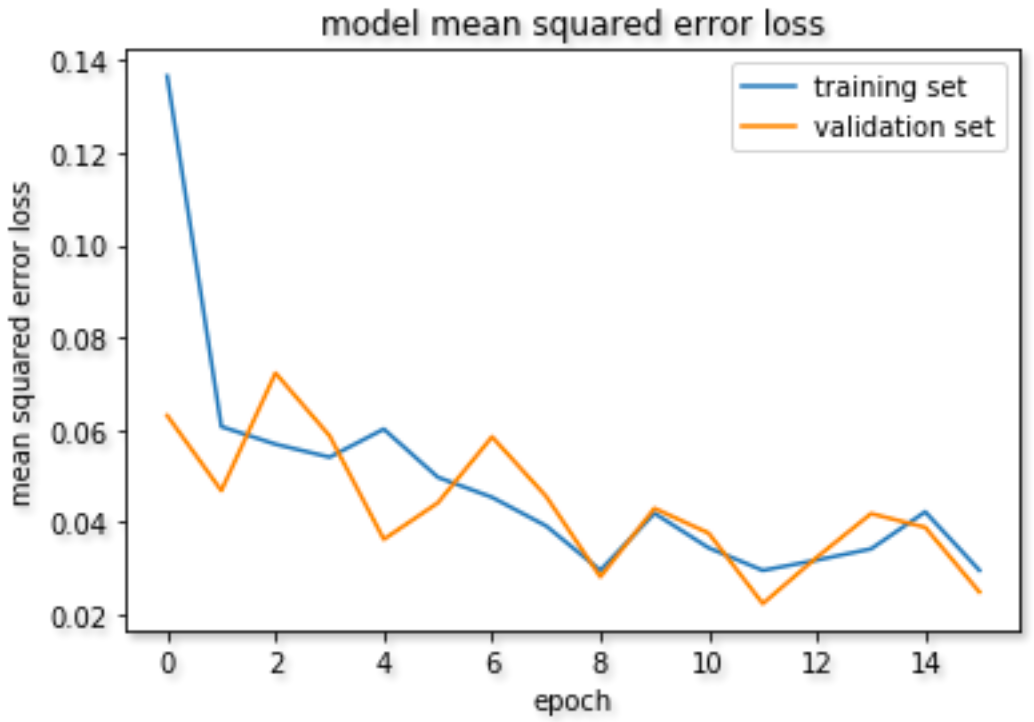
The rest are the 5 convolution layers, the flatten and the 4 fully connected layers as Nvidia folks designed it.

2. Attempts to reduce overfitting in the model

My model do not have dropout layers since original Nvidia didn't have. Instead, I focused on getting more training data, specially situations where you counter-steer, when you steer to come back

to road…. Also, at the difficult curves, I took training points driving back and forth, and steering at different points at the curve.

I checked the history of the Keras model as suggested, and since the graph is like this:



I didn't notice heavy overfitting, maybe a little bit, so I let it stay as is.

3. Model parameter tuning

My model is using Adam optimizer, so nothing manual tuned. Batch sizes are 128, pretty big to squeeze the 11GB of RAM of my 1080TI.

4. Appropriate training data

At first I recorded two complete laps with car at lane center.

Then I tried the model, and car was steered out of road at first step curve after the bridge.

Then I recorded more laps at center of the lane, also counterclockwise.

The car still does not turn properly at the curve after the bridge.

Hence I recorded the car driving along the curves. Recovery from the road endings boosted model behavior. After recording curves clock and counterclockwise 3-4 times, model performs OK and car steers OK at all curves.

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

Model Architecture and Training Strategy

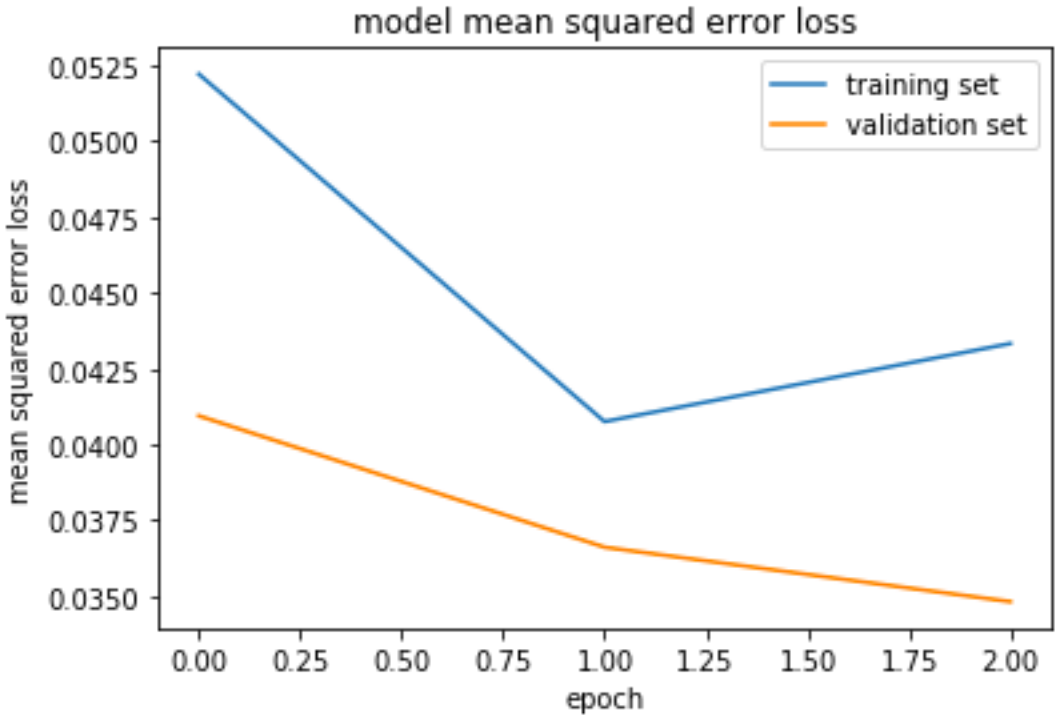
1. Solution Design Approach

The overall strategy for deriving a model architecture was to use a tested model (Nvidia SDC team one) and then trying by myself before making changes to see how I can really improve for my assignment.

My first step was to use a convolution neural network model similar to the Nvidia SDC team one. I thought this model might be appropriate because is derived from LeNet and is specifically made for this very one use.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. (80%)

The model at first was not overfitting heavily. So I didn't include a dropout layer.



To prevent future overfitting I focused on getting more data to train, since little differences are made in every lap, and curves can be negotiated from different angles.

Also, I learnt that suggested Keras model history feature graphs by epoch, so I selected a small steps_per_epoch number (2) and half validation steps (1 per 2 training, too much validation) to control the overfitting.

At first run, the car gets out the road at the first curve after the bridge (first steep curve).

After getting more and more datapoint of the vehicle through the curves, the model performs OK and at the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

Final architecture is described above as not changes have been made. Is coded using new Keras v2.0 API.

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to correct when is out of trajectory. These images show what a recovery looks like starting

Then I repeated this process on track two in order to get more data points.

After the collection process, I had 8784 number of data points x3 different cameras. I then preprocessed this data just normalizing it.

I finally randomly shuffled the data set and put Y% of the data into a validation set.

I put more effort in driving more time than in coding an augmenting dataset function.

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 around 14 as evidenced by the stalling loss and validation. (14 epoch with 128 batch size and 2 steps per epoch). I used an adam optimizer so that manually training the learning rate wasn't necessary.