Project: Behavioral Cloning

Juan Cheng 2017.08.23

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

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
- video.py for creating a video of the vehicle when it is driving autonomously
- model.h5 containing a trained convolution neural network
- model_log.txt containing the log output when training the model
- model_mse.png containing the plot of training and validation loss for each epoch
- run7.mp4 containing the video of driving autonomously for one full lap
- Behavioral_Cloning_Writeup_Report.pdf summarizing the results

2. Submission includes functional code. Using the Udacity provided simulator and drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.h5 run7
```

3. Submission includes a video which is generated by executing

```
python video.py run7 --fps 48
```

The video shows the vehicle drives autonomously around the first track and remains on the road for an entire loop around the track.

4. 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 5x5 filter sizes and depths between 24 and 48 (model.py lines 71-77), and 3x3 filter with depth 64 (line 79-83).

The model includes RELU activations to introduce nonlinearity (line 71-83), and the data is normalized in the model using a Keras lambda layer (line 60).

2. Attempts to reduce underfitting or overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting or underfitting. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used both side camera images for data augmentation. The correction of steering measurements (a constant parameter) for the left and right camera images was tuned based on empirical outcome (line 17).

The model used a Cropping2D layer to remove information not related to the road surface in the upper and lower portions of the screen. It also helped the model train faster. The region of interest was tuned for best outcome (line 63-66).

The model used an adam optimizer, so the learning rate was not tuned manually (line 99).

4. Strategies for collecting 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 back to center, driving counter-clockwise for model generalization, flipping the images and using multiple cameras for data augmentation.

5. Solution Design Approach

The overall strategy for driving the model architecture was to avoid underfitting or overfitting.

My first step was to use a convolution neural network model that was introduced by Nvidia. Since it's referenced and recommended by the class, I thought this model might be a good starting point. I also started with using just the example training data. In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a high mean squared error on both the training set and the validation set. This implied that the model was underfitting.

To combat the underfitting, I double-sized the training data by including left-to-right flipped copies of all of the images. It helped reduce the mean squared error on both training set and validation set, but the car still fell off the track when testing the model.

I then used the left and right camera images, again with left-to-right flipped copies, to triple the size the training data to further avoid underfitting. Use of the left and right camera angles should also help the model to recover from being off-center. This was the first time that I saw the car not to fall off the track at any point but there were still some portion of the lap that the car drove very close to the edge of the track. So I experimented on different correction adjustment values for the steering measurements of the side cameras images and found a fairly good setting.

Since the training image size is 160x320, while the input shape of Nvidia model is 66x200, I cropped the images from top, bottom, left and right to attempt to exclude upper and lower regions with no track information and make the images fit the size of Nvidia input. This helped with expediting training, however some of the useful info along the left and right sides of the road got removed too which could cause the car sometimes drive off the center.

I then tried only cropping the images from top and bottom and feeding input size of 66x320 to the Nvidia network. The model finally worked well.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road and almost always stay in the center of the road

6. Final Model Architecture

My final model consisted of the following layers:

Layer	Description

Input	160x320x3 RGB image
Preprocessed Input	66x320x3 RGB image
Convolution 1	Input = 66x320x3 Output = 31x158x24 5x5 kernel, 2x2 stride
RELU	
Convolution 2	Input = 31x158x24 Output = 14x77x36 5x5 kernel, 2x2 stride
RELU	
Convolution 3	Input = 14x77x36 Output = 5x37x48 5x5 kernel, 2x2 stride
RELU	
Convolution 4	Input = 5x37x48 Output = 3x35x64 3x3 kernel, 1x1 stride
RELU	
Convolution 5	Input = 3x35x64 Output = 1x33x64 3x3 kernel, 1x1 stride
RELU	
Flatten	Input = 1x33x64 Output = 2112
Fully Connected 1	Input = 2112 Output = 2112
RELU	
Fully Connected 2	Input = 2112 Output = 100
RELU	
Fully Connected 3	Input = 100 Output = 500
RELU	
Fully Connected 4	Input = 50 Output = 1

It differs from the Nvidia model in the sizes of the convolutional layers and the first fully connected layer because the input image size is bigger. It also differs in the size of the last fully connected layer as only a single output, the steering angle, is required.

7. Final Model Mean Squared Error Loss

Here is the training epoch log for the final model:

Train on 38572 samples, validate on 9644 samples

Epoch 1/5

38572/38572 [==============================] - 536s - loss: 0.0229 - val_loss: 0.0247

Epoch 2/5

38572/38572 [==============================] - 545s - loss: 0.0188 - val_loss: 0.0248

Epoch 3/5

38572/38572 [==============================] - 546s - loss: 0.0166 - val_loss: 0.0271

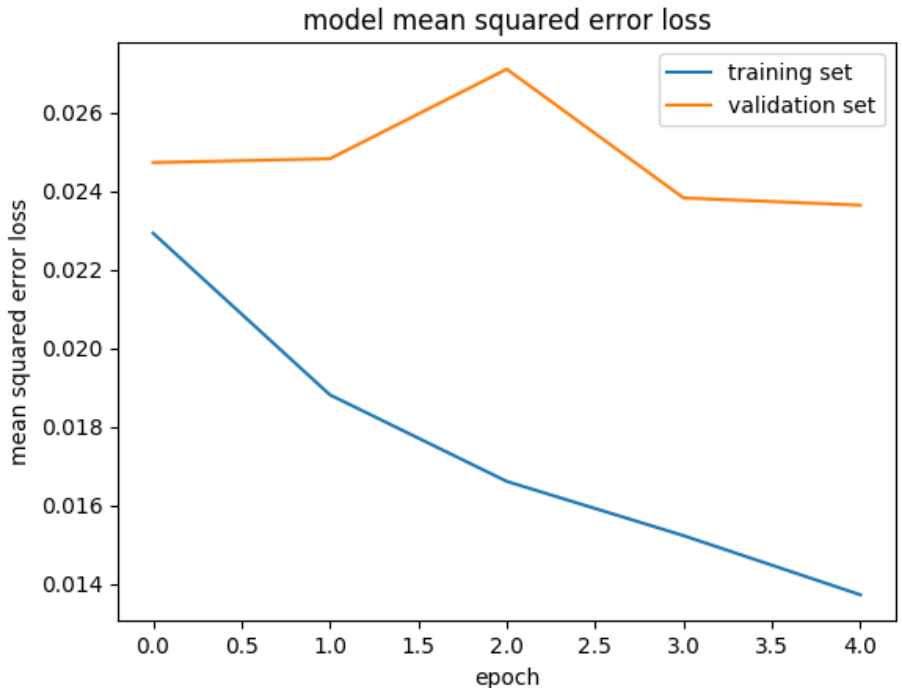
Epoch 4/5

38572/38572 [==============================] - 544s - loss: 0.0152 - val_loss: 0.0238

Epoch 5/5

38572/38572 [==============================] - 543s - loss: 0.0137 - val_loss: 0.0236

Here is the visualization of the final model mean squared error loss:



The training mse keeps dropping dramatically after each epoch run and the validation mse has some minor up and down jitter but gradually goes down to a lower value. Both imply the model doesn't have overfitting or underfitting issues. I've also noticed that a lower mse doesn't necessarily mean the car drive well. Some of my intermediate models have lower mse, but the car drove worse than the final model.