Behavioral Cloning

Writeup Report

By Liang Hu

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.md or 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

In the model.py, lines 54~73 are how I built this model. My model first normalized data using a Keras lambda layer. Then I cropped parts of the image on the top and bottom which are not very important in training the network. Then I adopted a similar model architecture to Nvidia's network. There are 5 convolution layers and 3 fully connected layers. The filter sizes and depths for each convolution layer are shown in the below table. After each convolution layer, I use ReLU activation to introduce nonlinearity and dropout to reduce overfitting.

	Filter size	Depth	Stride	Padding
Convolution layer 1	(5,5)	24	2	valid
Convolution layer 2	(5,5)	36	2	valid
Convolution layer 3	(5,5)	48	2	valid
Convolution layer 4	(3,3)	64	1	valid
Convolution layer 5	(3,3)	64	1	valid

2. Attempts to reduce overfitting in the model

The model contains a dropout layer after each convolution layer in order to reduce overfitting, and the drop rate is 0.5.

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 77-81). 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 an adam optimizer, so the learning rate was not tuned manually (model.py line 76).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, counter-clock wise center lane driving, and recovering from the left and right sides of the road.

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 adopt the Nvidia's network for self-driving, but to add a dropout layer after each convolution layer to reduce overfitting. I thought the Nvidia's network model might be appropriate because it has been used in the real world driving.

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 low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting. The Nvidia's network has multiple convolution layers, so in this simulation overfitting might be a problem.

To combat the overfitting, I modified the model by adding a dropout layer after each convolution layer. The drop rate is 0.5 so that half of neurons will be randomly dropped.

In total this model has 5 convolution layers and 3 fully connected layers.

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 run the simulator again to collect more data, including counter-clock wise driving, recovering from left and right, etc.

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 (model.py lines 54-73) consisted of a convolution neural network with the following layers and layer sizes, as shown in the below table.

Layer	Size
Input image	(160, 320, 3)
Cropped image	(65, 320, 3)
Convolution layer 1	(31, 158, 24)
Convolution layer 2	(14, 77, 36)
Convolution layer 3	(5, 37, 48)
Convolution layer 4	(3, 35, 64)
Convolution layer 5	(1, 33, 64)
Flatten layer	2112
Fully connected layer 1	100
Fully connected layer 2	50
Fully connected layer 3	10
Output	1

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded 2 laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded 1 lap on track on track one also using center lane driving, but the driving direction is counter-clock wise.

Third, I recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to avoid drive off track. This is an image showing what a recovery from left looks like.

To augment the data sat, I also flipped images and angles thinking that this would avoid too much left turns or right turns in the collected dataset. For example, here is an image that has then been flipped (the raw image is the above image):

Also, I include the training dataset given by the workspace. Therefore, in total I had 65,652 number of data points. I then preprocessed this data by normalization and cropping. The cropped part is ((70,25), (0,0)) so that the sky and the car hood are removed as noises.

I finally randomly shuffled the data set and put 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. I used 2 epochs 2. I used an adam optimizer so that manually training the learning rate wasn't necessary.