Behavioral Cloning

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

The model employed is a modified version of the NVIDIA architecture. The NVIDIA architecture consists of five convolution layers and three fully-connected layers. I added three more fully-connected layers to the NVIDIA architecture for better non-linear learning space.

The first three convolution layers have a 5x5 kernel size with depths of 24, 36, and 48. These layers also have RELU activation layers with sub-sample of 2x2. The fourth and fifth layers both 3x3 kernel sizes with a 64 depth.

The five fully-connected layers break down the neurons to 400, 200, 100, 50, 10, all the way to 1.

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure the model was not overfitting. The model was tested by running it through the simulator to ensure the vehicle was staying on course.

3. Model parameter tuning

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

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of five different data sets: center lane driving, reverse direction, recovering from the left and right sides of the road, dirt road turn, and sharp turn.

The test data was mostly comprised of center lane driving. I used the mouse to provide a more continuous steering angle when driving. I also drove around the same speed as the drive.py setting of around 9mph.

To generalize a bit, I drove in the reverse direction. This would help reduce the left lean tendency.

For robustness, I added recovering from the left and right sides of the road. In my first implementation I drove from off the road back in. I found later that it was better to still be within the road but at an angle towards the left or right side then recover worked better.

I ran into some difficulty with the turn after the bridge where the right side contained dirt. I provided more data sets with dirt road turning to help overcome the tendency to drive through the dirt.

The model did not turn sharp enough on the turn following the dirt turn so I provided additional training data of that intersection as well.

Later data sets contained a lower resolution which helped save memory. If I were to capture the data again I use the lower resolution image setting.

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 autonomously drive around the track without leaving the road.

My first step was to use a convolution neural network model similar to the LeNet architecture which I thought was a starting point and was available through the lectures.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. The training and validation results were higher than my final solution. Training and validation results were close in value.

Then I proceeded to test the results on the simulator with lack-luster results. I decided to iterate on the LeNet, modifying training data and customizing the model architecture. I almost was successful failing at the last sharp turn.

At this point, I decided to make a radical change based on comments from Paul Heraty. I changed the architecture from LeNet to NVIDIA architecture. I switched from keyboard control to mouse to smoothen the training data.

The NVIDIA architecture was much faster than the LeNet which help in the iterations. It also converged a lot faster. I think I could of reduced the number of epochs from 5 to 3 but I kept 5 due to Paul Heraty's suggestions.

When I tested the results on the simulator I would get up to the dirt road and fail. Even when I added training data of the dirt road it would still fail. I decided to add another two fully connected layers to the NVIDIA architecture and succeeded in passing the dirt road but failing on the sharp right turn.

I then added another fully connected layer and training data and succeeded in going around the track autonomously.

2. Final Model Architecture

The final model architecture (model.py lines 18-24) consisted of a convolution neural network with the following layers and layer sizes.

1. Convolution Layer 5x5, 24 depth, 2x2 RELU
2. Convolution Layer 5x5, 36 depth, 2x2 RELU
3. Convolution Layer 5x5, 48 depth, 2x2 RELU
4. Convolution Layer 3x3, 64 depth
5. Convolution Layer 3x3, 64 depth
6. Flatten Layer
7. Fully Connected Layer 400
8. Fully Connected Layer 200
9. Fully Connected Layer 100
10. Fully Connected Layer 50
11. Fully Connected Layer 10
12. Fully Connected Layer 1

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded three laps on track one using center lane driving. I also recorded two laps on track one using reverse 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 recover to the center. These images show what a recovery looks like starting from right then centering:

To augment the data set, I also flipped images thinking that this would help generalize the model. I found this did not help as the model tended to oscillate more aggressively.

I recorded more training data for the dirt turn and the sharp right turn in both directions.

After the collection process, I had 21,953 number of data points. I then preprocessed this data by cropping the data. I took 50 pixels from the top and 20 pixels from the bottom. The right and left side remain the same.

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. The ideal number of epochs was 5 as evidenced by the stabilization training and validation results. I used an adam optimizer so that manually training the learning rate wasn't necessary.