Behavioral Cloning

Project Writeup by Aidos Utegulov

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 convolutional 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

I based my model on NVIDIA's Architecture for end-to-end learning model described in [this paper] (http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf). The model contains 5 Convolutional Layers, a Flatten layer and 4 Dense layers. The first 3 Convolutional layers use strided convolutions with a 2x2 stride and a 5x5 kernel size, and the remaining two Convolutional layers use non-strided convolutions with a 3x3 kernel size I've also added 2 Lambda layers: first one for cropping the top 60 and bottom 24 pixels, and the second one for normalizing the images. (model.py lines 53-67)

The model includes RELU layers to introduce nonlinearity (provided as 'activation' argument in the Convolution2D layers), and the data is normalized in the model using a Keras lambda layer (code line 55).

2. Attempts to reduce overfitting in the model

The model contains 2 dropout layers with a probability of keeping activations of 0.5 in order to reduce overfitting (model.py lines 62 and 66).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 70). 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, mse was chosen as the loss function (model.py line 69).

4. Appropriate 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, and turning the car around and driving in the opposite direction to try and balance the data

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 base the model on the already working model from NVIDIA, simply adding the image preprocessing Lambda layers and dropout

My first step was to use a convolution neural network model similar to NVIDIA's end-to-end model Architecture. I thought this model might be appropriate because it already

proved to be working for a real self-driving car, plus it was given to us in the introductory videos for the project and the Project 3 Q&A Session recorded by David Silver on youtube.

In order to gauge how well the model was working, I split my image and steering angle data into a training (80%) and validation set (20%). I found that my first model had loss of 7% on the training set and 5% on the validation set. This was a bit high, and the car went off road and to the right going full-speed (I changed the value of speed from 9 to 30 in drive.py) in the autonomous mode.

To reduce the loss, I added the dropout layers to the model and it gave me the loss of 4% on the training set and validation loss of 3%.

Then I re-recorded the data, recording a lap of recovery driving and 2 laps of driving in the opposite direction in addition to 2 laps of normal driving.

The final step was to run the simulator to see how well the car was driving around track one. I decreased the speed in drive.py to 19 (my only change in drive.py) and ran the simulator in autonomous mode. In some spots, especially on the sharp turns, the car came dangerously close to the edges of the road, but it never went off the road, and was able to correct itself.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road, although it does keep wiggling a bit on certain segments.

2. Final Model Architecture

The final model architecture (model.py lines 53-67) consisted of a convolution neural network with the following layers and layer sizes: 1)Lambda layer for cropping the images with input shape of (160, 320, 3); 2)Lambda layer for input normalization; 3)Convolution2D layer - 24 5x5 filters with 2x2 stride; 4)Convolution2D layer - 36 5x5 filters with 2x2 strides; 5)Convolution2D layer - 48 5x5 filters with 2x2 strides; 6-8) 3 Convolution2D layer - 64 3x3 filters; 9)Flatten layer; 10) Dropout layer with p_keep = 0.5; 11)Dense(100) layer; 12)Dense(50) layer; 13)Dense(10) layer; 14)Dropout layer with p_keep = 0.5; 15)Dense(1) layer;

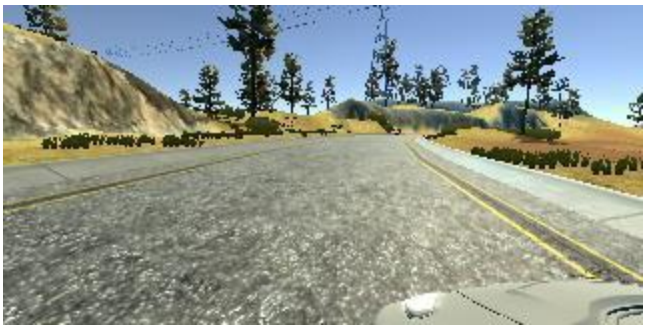
3. Creation of the Training Set & Training Process

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 recover back to the center of the lane from the edges of the road. These images show what a recovery looks like starting from the left edge the road:

Then I turned the car around and recorded two more laps in the opposite direction in order to get more data points.

To augment the data set, I also flipped images and angles thinking that this would balance the measurements set for steering. For example, here is an image that has then been flipped:



After the collection process, I had 17481 number of data points. I then preprocessed this data by cropping the top 60 and bottom 24 pixels from the images and normalizing the images using the formula image = image/255.0 - 0.5.

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 line 70. I used an adam optimizer so that manually training the learning rate wasn't necessary.