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

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with filter sizes between 5x5 and 3x2, depths between 24 and 64. It has 5 layers of convolution network and 3 fully connected layers. The model includes RELU layers to introduce nonlinearity after each convolution layers. It did dropout after each layers to decrease overfitting. Data is normalized in the model using a Keras lambda layer. Also, I balanced data at the very beginning, make sure each angle group will not be over represented(line 24 to 33).

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 15). I added two additional central lap one counterclockwise lap. For certain part of the road, I did additional training to reinforce the turning capability. 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 80).

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/right sides of the road and a counterclockwise lap.

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 referring existing famous model and did modification.

My first step was to use a convolution neural network model similar to the Nvidia did. I thought this model might be appropriate because it has been recommended by many people.

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

Initially I only used given data sample. The mean squared err is low, but auto driving mode doesn't perform well. I think it's because sample data close too much to the center. Once car leaves the center it can not go back. I added more training data.

Then, I found my 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.

To combat the overfitting, I modified the model and added dropout layers. Also I tried to balance my model, make sure each group of angles have equal chance to represent in the model.

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 added additional test data in these spots.

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 58-78) consisted of a convolution neural network with the following layers and layer sizes.

A lambda layer to normalize image.

A cropping layer to cut image.

Five convolutional layers, their sizes are 24x5x5, 36x5x5, 48x5x5, 64x3x3, 64x3x3 with activation function RELU. First three of convolution2D layers have 2x2 strides.

Three fully connected layers with size 100, 50, 10.

Output layer's size is 1.

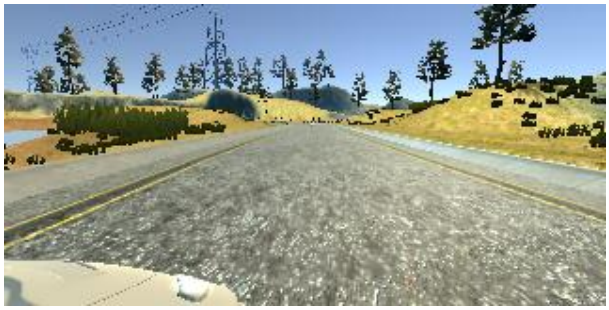
Also 0.2 dropout rate is applied between every two layers.

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 how to back to the center. These images show what a recovery looks like starting from sides:

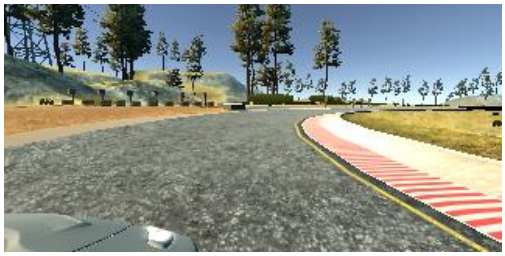




Then I repeated this process on track two in order to get more data points.

To augment the data set, I also flipped images and angles thinking that this would balance the model.

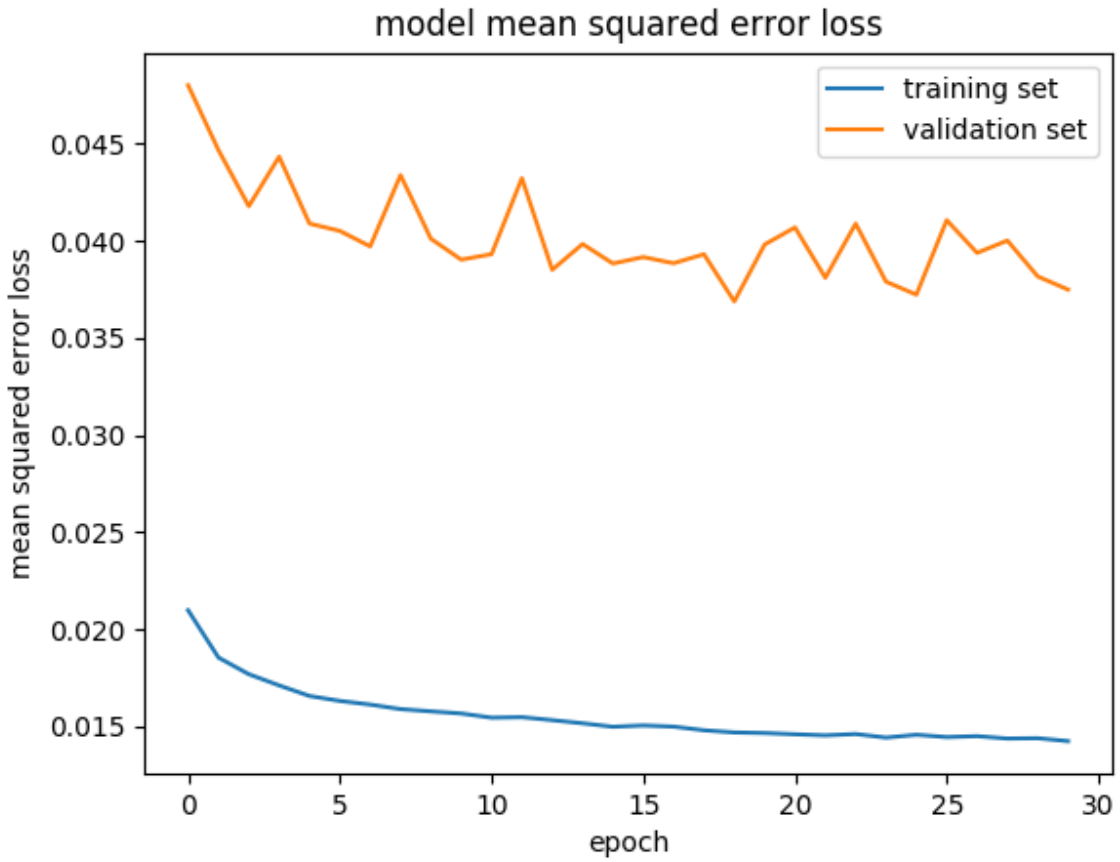
I also drove a counterclockwise lap.

After the collection process, I had 80k+ number of data points. I then divided these data into 2000 groups, with each group no more than 200 examples. Total near 30k data are trained in the model.

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 10 as evidenced by the following graph:



I used an adam optimizer so that manually training the learning rate wasn't necessary. Although the model looks like a little bit overfitting, the result in the auto mode is not bad.