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.pdf summarizing the results
* run1.mp4 showing two successful runs around the track

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
```sh
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

Convolutional Neural Network using Keras's Sequential model:

Pre-processing and normalisation:
1. Normalisation using the Keras Lambda layer (line number: 56)
2. Image resizing using Keras Cropping2D

Layers:
Layer 1 - Convolutional2D with RELU activation

Layer 2 - Convolutional2D with RELU activation
Layer 3 - Convolutional2D with RELU activation
Layer 4 - Convolutional2D with RELU activation
Layer 5 - Convolutional2D with RELU activation
Layer 6 - Fully Connected with Flatten, Dense and Dropout
Layer 7 - Fully Connected with Dense and Dropout
Layer 8 - Fully Connected with Dense
Layer 9 - Fully Connected with Dense

Model Compilation:
Using the Keras Adam optimiser and 'mse' loss (model.py lines 91)

Train the model:
Using Keras fit, I shuffled and split the data and specified 5 epochs (model.py lines 94)

Save the model:
Using Keras save (model.py lines 97)

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 71, 81).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 94). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. A video showing two successful runs around the track and was captured and saved in the submission folder as 'run1.mp4'.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 91).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used the training data given in the project repository and removed roughly half of the 0.00 steering values due to the exploration results displaying an uneven distribution of steering values with a large amount tending to 0.00. This resulted in much better training and validation results.

MODEL ARCHITECTURE AND TRAINING STRATEGY

1. Solution Design Approach

The overall strategy for deriving a model architecture was to be able to successfully learn from the training data so that the model would be able to successfully learn how to make it around the track.

My first step was to use a convolution neural network model similar to the LeNet architecture as I had spent a lot of time getting to know this through the last project. I thought this model might be appropriate because it would give me a starting point to work from.

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.

To combat the overfitting, I modified the model so that it contained dropout layers and also reduced the number of epochs for training.

The final step was to run the simulator to see how well the car was driving around track one. There was one spot where the car kept running straight on into the sand pit. In order to improve the driving behavior in these cases, I did the following:

- Reduced the dropout layers to just two, one at Layer 6 (model.py lines 77) and Layer 7 (model.py lines 79).

- Augmented the data to include flipped versions of the images and measurements.

I then ran the training again, followed by another simulator run. Disaster! This time it was even worse. Next iteration:

- Removed augmentation.

- Identified through data visualisation that the data was not very well distributed, so I removed roughly half of the 0.0 steering values.

- Increased the epochs slightly to 5.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

Convolutional Neural Network using Keras's Sequential model:

Pre-processing and normalisation:
1. Normalisation using the Keras Lambda layer (line number: 56)
2. Image resizing using Keras Cropping2D

Layers:
Layer 1 - Convolutional2D with RELU activation

Layer 2 - Convolutional2D with RELU activation
Layer 3 - Convolutional2D with RELU activation
Layer 4 - Convolutional2D with RELU activation
Layer 5 - Convolutional2D with RELU activation
Layer 6 - Fully Connected with Flatten, Dense and Dropout
Layer 7 - Fully Connected with Dense and Dropout
Layer 8 - Fully Connected with Dense
Layer 9 - Fully Connected with Dense

Model Compilation:
Using the Keras Adam optimiser and 'mse' loss (model.py lines 91)

Train the model:
Using Keras fit, I shuffled and split the data and specified 5 epochs (model.py lines 94)

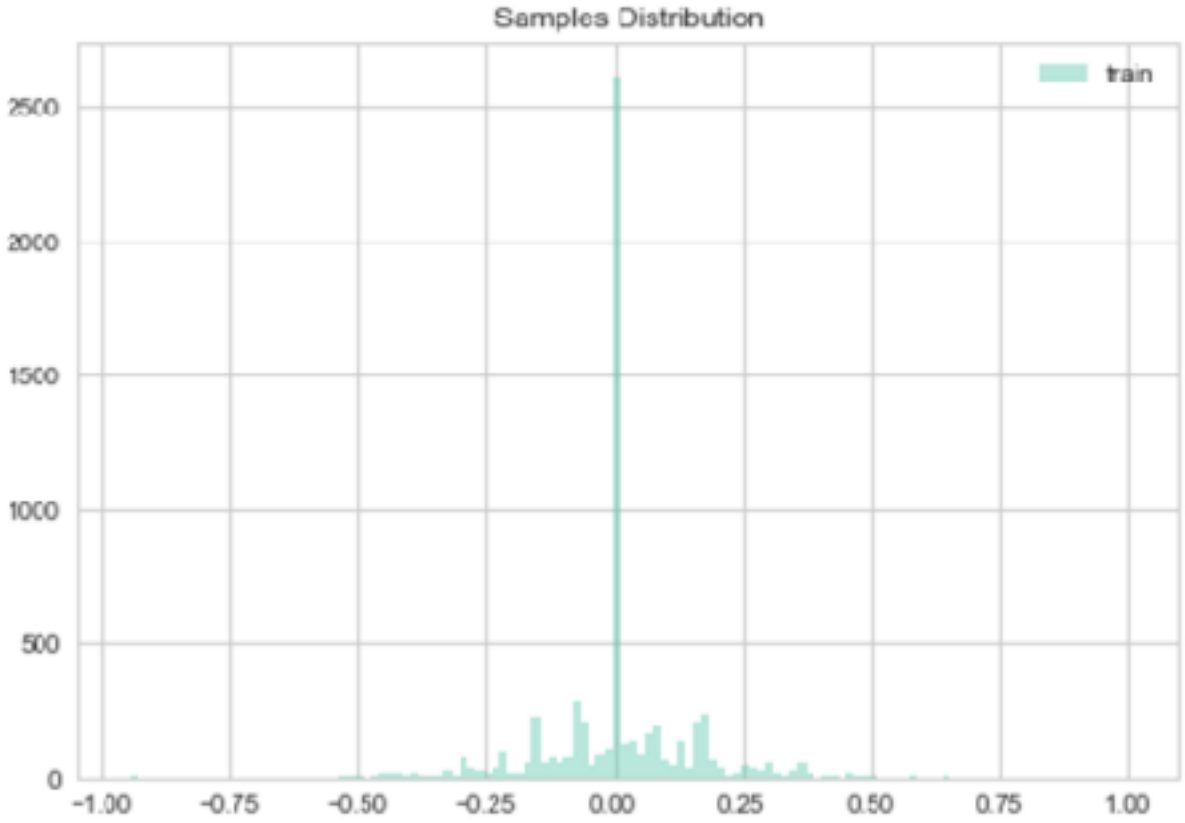
Save the model:
Using Keras save (model.py lines 97)

3. Creation of the Training Set & Training Process

In one iteration I also flipped images and angles thinking that this would help by increasing the training data and the tendency to steer off the track to the left. However, this lead to a decrease in performance and the car would consistently crash over the edge into the water. So I removed the augmentation and applied the following exploration steps.

I used the training data provided in the project repository and completed the following exploration:

Visualised the samples distribution and noticed that there was a large amount of 0.00 steering values. After several iterations of training and running the simulator I removed roughly half of the 0.00 steering values making the distribution much better. This was the final distribution:



NOTE: I did remove much more of this data, however the performance was hit because the model lost valuable information about the data that was deleted. Eventually, removing roughly half the values seem to keep enough of the valuable information but negate the tendency toward 0.00 steering.

After the collection process, I had 6275 number of data points. I then preprocessed this data by applying normalisation using the Keras lambda layer and then applied cropping to the images using Keras Cropping2D layer.

I finally randomly shuffled the data set and put 30% 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 how well the model trained, low training and validation loss, and then the overall result of the successful runs around

the track in simulation mode. Additionally, I used an adam optimizer so that manually training the learning rate wasn't necessary.