Behavioral Cloning

Overview

There is a high dependency of data in any learning algorithm, be it machine learning , Deep learning , and any other iterative algorithms. This leaves us with the question , how to get such large datasets? . One of the ways is to use simulators or specifically designed games . When data deficient, we can augment or create our own data from scratch by simulating complex physical models representing real life situations using simulators and games. This “Behavioral Cloning” project uses one such game designed by the “Udacity Team” . There are different views or cameras attached to the dummy car present in the game through which we collect the data(images , Steering angle, Throttle, Braking) over the course of the car maneuver on the map. This data is then used as training and validation data for training

the network for predicting the steering angle. This deep learning network is then saved and used for driving the car autonomously over the map.

Goals

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

Specifications.

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

My model consists of a convolution neural network with 5x5 filter sizes and depths 24,36,48, 3x3 filter sizes and depths 64 , 64 (model.py lines 153-165)

The model includes ELU(Exponential Linear Units) layers to introduce nonlinearity , and the data is normalized in the model using a Keras lambda layer .

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 172,176).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 20-127). 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 187).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving data, left camera data with an offset in the steering angle by +0.2 , right camera data with an offset in the steering angle by -0.2.

Also used the data collected by driving the car in the opposite direction this also augments the data there by making the network generalize well on new data from similar environments.

For each image in the training data and validation data (Split using sklearn train_test_split method) I augmented the batch data with the corresponding

flipped image horizontally and also changed steering accordingly(multiplying by -1).

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

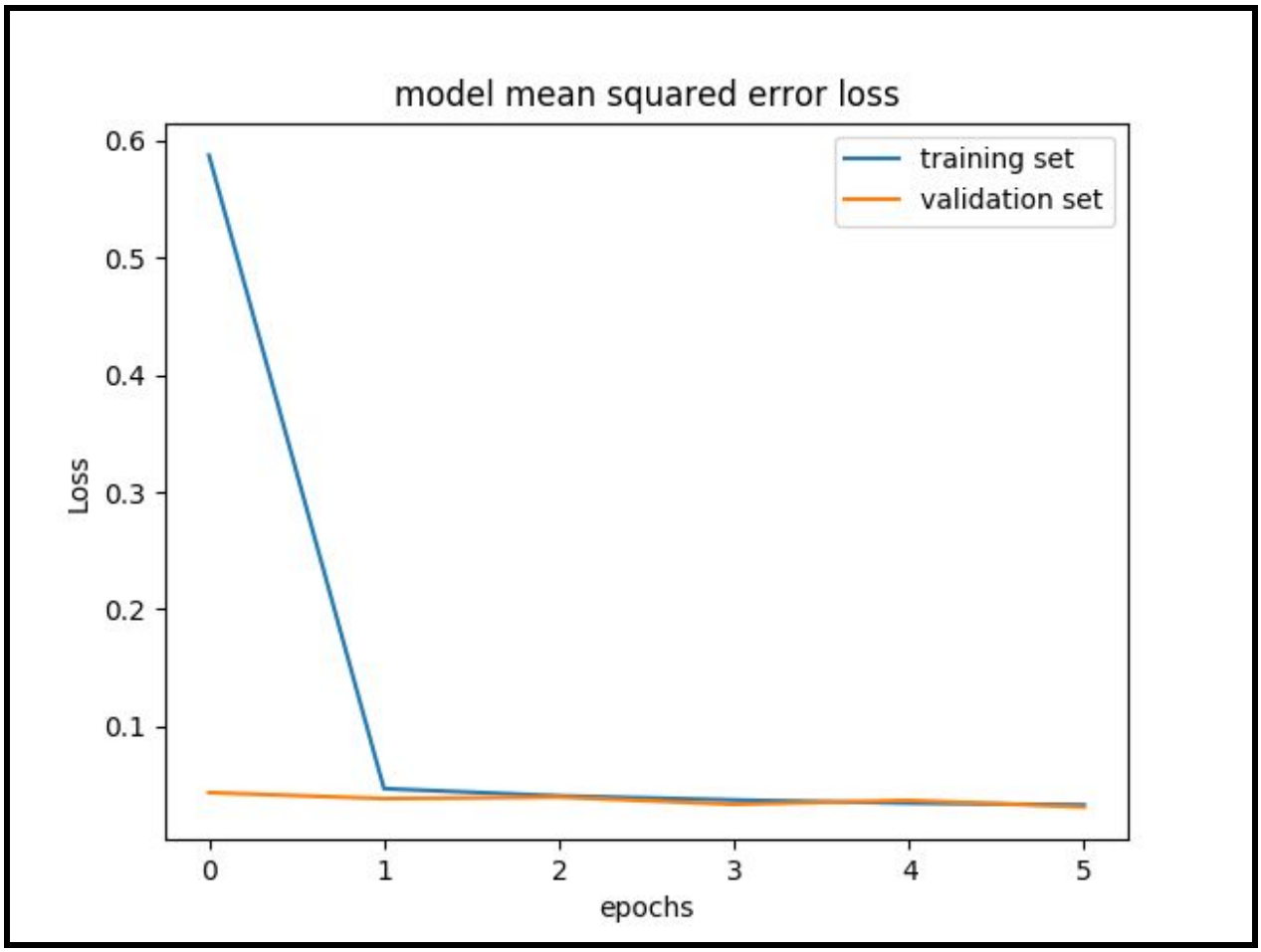
1. Solution Design Approach

The overall strategy for deriving a model architecture was to make the car drive autonomously on the map .

My first step was to use a convolution neural network model similar to the model mentioned in "NVIDIA's, End-to-End Deep learning for Self Driving Cars" Article , thought this model might be appropriate because this was designed for a specific use case similar to this project.Initially tried including some "Max Pooling layers" in between layers in order to make the "Effective receptive field" of the last convolutional layer cover whole picture given as input so as to use most of the information from the image or picture, but measurements for height and width for subsequent convolutional layers were becoming small, this issue made me to revert back to the original standard model.

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 tried with just two laps of data from the game but this didn't give much reliable validation loss decrease and performance in game when switched to autonomous mode, also the data used here doesn't include any sort of augmentation or flipping etc. Based on the previous model performance observations collected three laps of data and also collected while driving in the opposite direction. This time the individual batch data is being augmented with the left , right camera images, flipped images. I found that my second model had a low mean squared error on the training set and relatively low mean squared error on the validation set. This implied that the model will work with good and quality data. But still there could be a decrease in the validation set error (because the model may have overfitted the data because of the large data).

To combat the overfitting, I modified the model by adding two dropout layers in between fully connected layers with different leave probabilities . After training the loss curve :



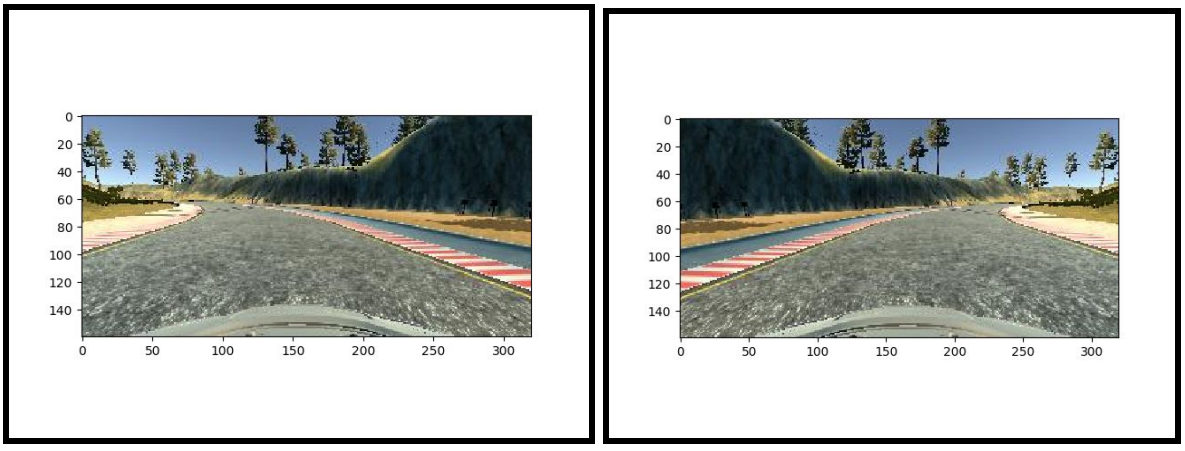
From the above loss curve over some epochs it is evident that it is generalizing well on the data.

The final step was to run the simulator to see how well the car was driving around track one. At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded three laps on track one using center lane driving and three laps in opposite direction center lane driving Here is an example image of center lane driving:

To augment the data set, I also flipped images and angles thinking that this would make it generalize well on the images with similar structure of road and texture and environment that it has never seen in the training set For example, here is an image that has then been flipped:



Original Image

Flipped Image

After the collection process, I had 6 images per entry in the “driving_log_combined.csv” file , so i had a total of 3917 * 6 = 23502 in total, in each batch i had 192 images .

I have used a generator function common for both train and valid data to provide with a specific batch data on the go without needing to load all of the data at once into memory. In each epoch the data is shuffled and then divides into batches and is then given for training or for validation.

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 6(for me), i tried with 10 epochs but at 6 th epoch the validation set error increased and it crossed the training set accuracy .The following plot shows that the model did not perform well or did not generalize well after 6th epoch because it crossed the training loss. I used an adam optimizer so that manually training the learning rate wasn't necessary.