# Behavior Cloning

Feng Xiao (xiaofeng.ustc@gmail.com)

## Table of Contents



## Behavior 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

## 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 defining, training and saving the convolution neural network. To make the code more readable, I organize the code into the following 4 different classes:

- Model: responsible for building, training and saving the convolution neural network. It uses the other 3 helper classes.
- DataLoader: contains helper functions to load sample lines from dataset specified by the csv files. To ease the usage of multiple dataset, it supports loading dataset from a list of csv files and get all the samples.
- Generator: contains the python generator used by the model when applying Keras model fit_generator to fit the model. It also contains the data augmentation code.
- CNNetwork: contains the factory method to create the convolution neural network used by the model.

The file is self documented. It shows the pipeline I used for training and validating the model, and it also contains comments to explain how the code works.

# Model Architecture and Training Strategy

## 1. An appropriate model architecture has been employed

The final model I choose is the architecture from Nvidia:

- https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/

Aside from this original architecture, I added several different layers

- A 2D cropping layer: considering some part of the top and bottom are not needed for the training, I crop them out.
- RELU layers: after each convolution and fully connected layer, except for the last fully connected layer and the flatten layer, I add a RELU activation layer to introduce non linearity to the model.
- Dropout layers: to add regularization and prevent over-fitting, I added many layers of dropout to the model.

And the same as this the Nvidia model, I added a normalization layer at the beginning of the model.

For the details of the model I deploy, please refer to the model.py:

- CNNetwork.pilotnet().

## 2. Attempts to reduce overfitting in the model

To reduce overfitting, on one hand, I add dropout layers to the model above.

On the other hand, I try to train and validate the model on more data. You can see the Model.train() function takes a list of dataset as input. At the same time, I augment the data by flipping the original image (in Generator.batch_generator()). This doubles the size of the dataset and also helps to reduce overfitting.

## 3. Model parameter tuning

- Learning rate: As I use the adam optimizer with Keras (as shown in function CNNetwork.pilotnet()), the learning rate is not tuned manually.
- Batch size: Since I use generator to provide training set and validation set for the model fit function, the batch size is constraint by the GPU memory and system memory size. When too small batch size is used, it will take more time to train the model. On the other hand, if a too big batch size is used, it may not be able to fit to the memory. After some tuning, I choose the batch size to be 128 on my laptop.
- Dropout rate: After some tuning of the dropout rate, I choose it to be 0.05.
- Epochs: by plotting the loss value history of the training, I choose the epochs to be 10.

## 4. Appropriate training data

Training data is chosen to keep the vehicle driving on the road.

I collect my training data by driving the car counter clockwise in the center of the road, and I also use the sample training data which was collected by driving the car clockwise.

And to train the car to drive back to center, I also collect training data by recovering from left and right sides of the road.

For details about how I collect the training data, please find in the next section.

# Model Architecture and Training Strategy

## 1. Solution Design Approach

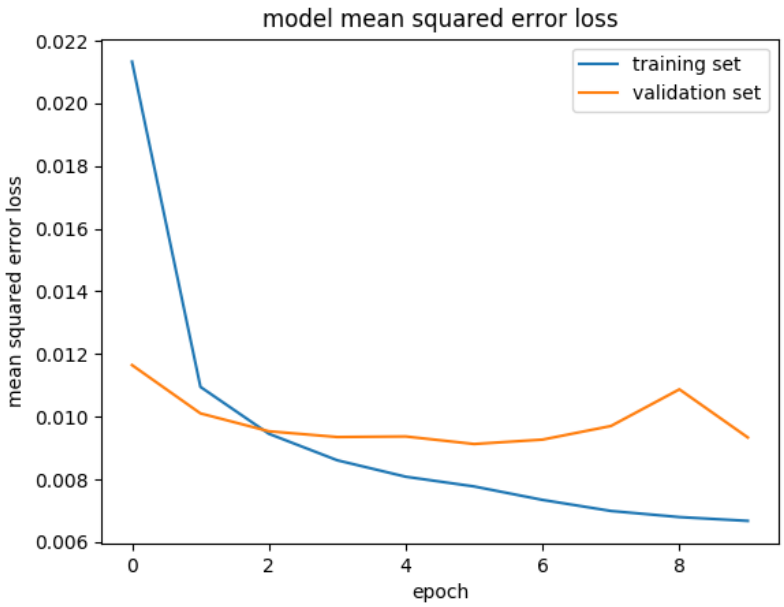
### Overall Strategy

When choose the model architecture, the over all strategy for me can be shown as following steps:

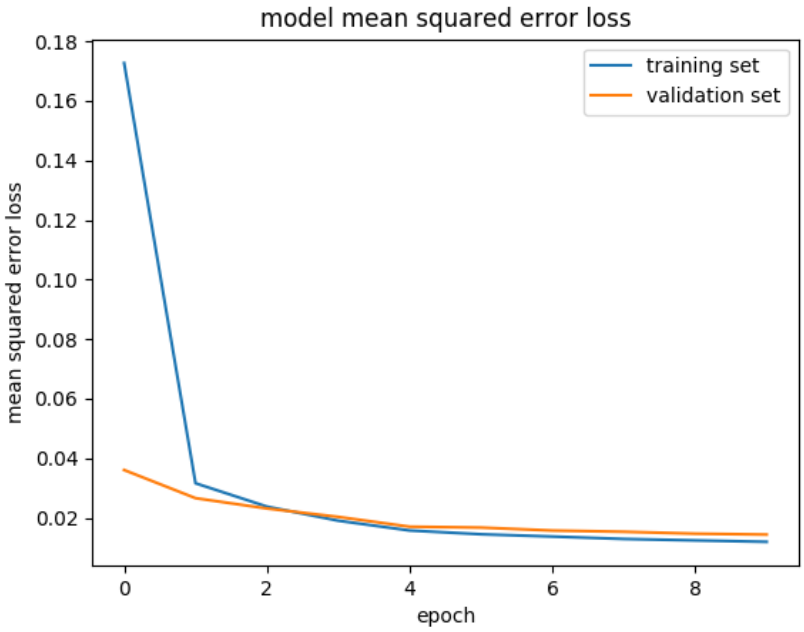
1. Try some model architecture.
2. Train the model with enough data.
3. Check the loss curve with respect to epochs and see if the model is overfitting or underfitting.
    A) If the model is over-fitting, will try to add more training data until it is not overfitting.
        a) If the model performs good enough, stop
        b) if the model performs not good enough, will try some more capable model.
    B) if the model is underfitting and the performance is not good enough, will try some more capable model.
4. Test the model in autonomous driving model of the simulator and check the performance of the model.
    A) If the car can run through the whole lap, stop.
    B) If the car failed running along the whole lap, go back to step 1.

### Try out of LeNet

For the first step, I try to use the LeNet learned from the course. Since it is easy to create and fast to train, it should provide me some insight about the capability of the model needed by this problem.  I tried to use a LeNet without dropout, and I got the following learning curve:




From this learning curve, it is apparent that the model is over-fitting. Therefore, I added dropout layers to the LeNet, after some tuning, I select the dropout rate to be 0.2. And this time I got the learning curve as following:




It seems the LeNet fits the problem quite well from the learning curve above. However when I use this model in the autonomous driving mode of the simulator, the car will leave the track at the sharp left turning point as shown in the following image:

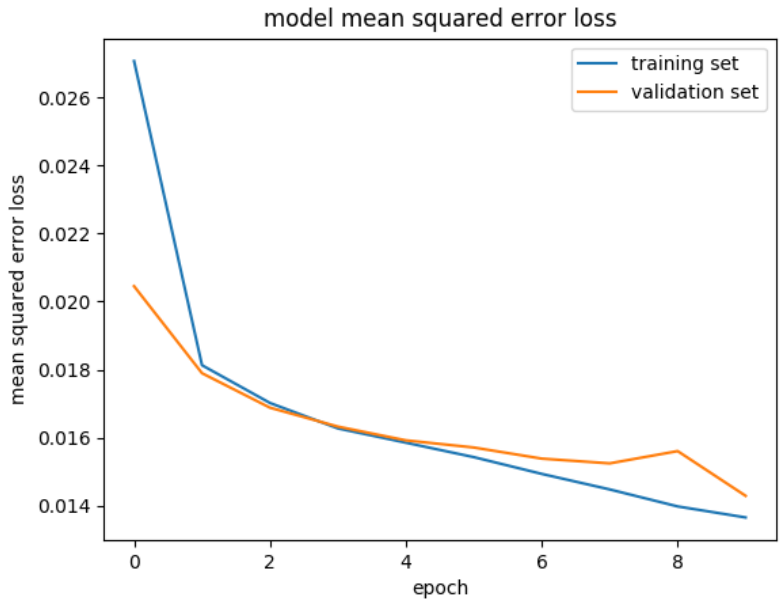

This seems the LeNet is underfitting the problem, and we need some more capable network architecture. And after this I changed to the convolution neural network architecture from Nvidia.

**CNN from Nvidia**

One reason to choose this model is this model has already been used by the Nvidia team to do similar work. In their work, they use this model not only in simulation environment but also with real data. I think it already shows the capability of the model. Also in the course, this model is also introduced as an improvement to the simple architecture.

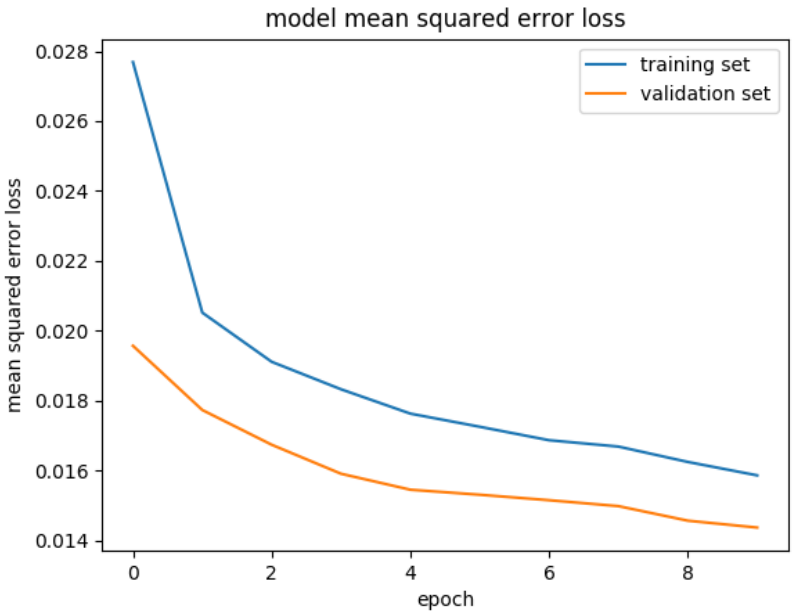
In order to gauge how well the model is working, I split my image and steering angle data into a training and validation set. The validation set is 20% and training set is 80% of the whole data set.

For the first time, I try out the Nvidia CNN without any dropout, and I got the following learning curve:



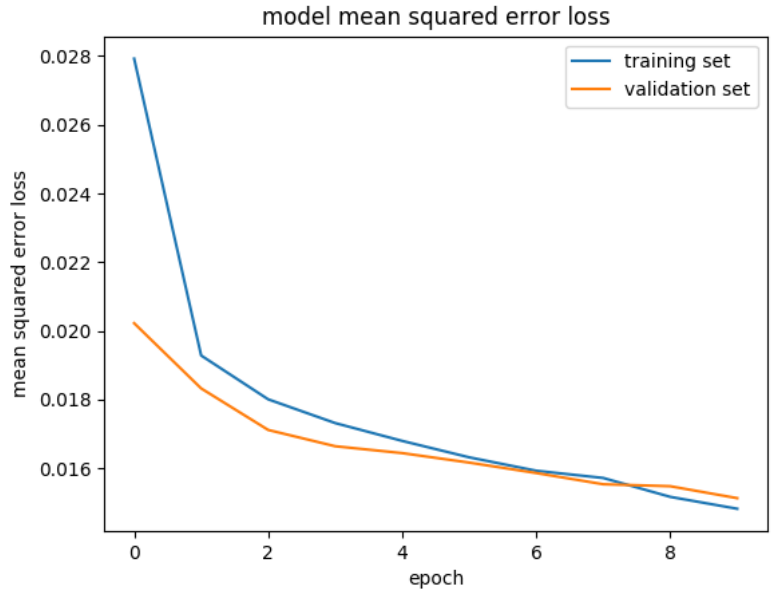

From the learning curve, we can see that the model is a bit over-fitting. Therefore, I added the dropout

layers to the CNN and after some tuning, I choose the dropout rate to be 0.1. And I get the following learning curve:

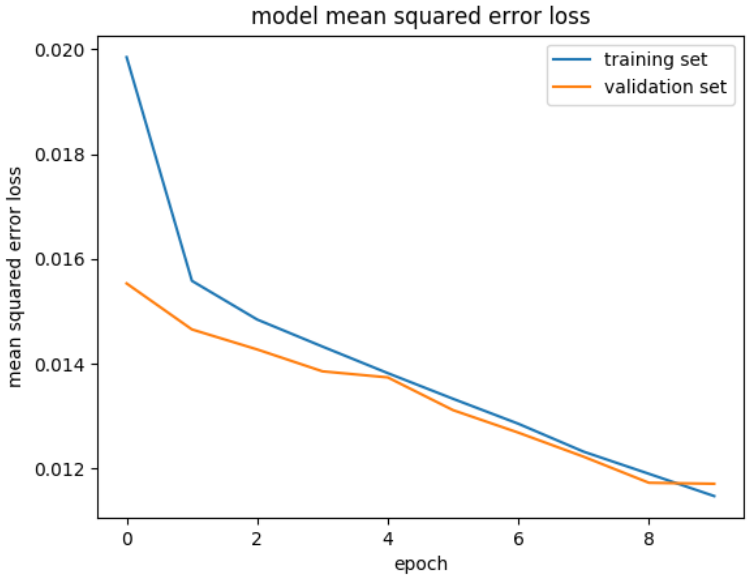



From the learning curve, we can see the model is a bit underfitting with dropout rate 0.1, but I would like to test this model with the simulation first and see how it is going. And it proves this model is already able to drive the whole lap without leaving the lap. This means this architecture is capable enough for this problem.

Though the model is good enough to finish the whole lap without leaving the lap, at some points the car touches the lane border. To make the model even better, I tried to reduce the drop rate of the model to be 0.05, and I got the following learning curve:



From this we can see the model works quite well, just a bit overfitting. And I decide to feed the model with more data to solve the over-fitting. After feeding the model with more data, I got the following learning curve:




When driving with this model in the simulation, the vehicle is able to drive autonomously around the track without leaving the road or even touching the border of the track.

## 2. Final Model Architecture

The final model architecture can be found from file model.py at CNNetwork.pilotnet(). It is based on the convolution neural network from Nvidia with the following layers and layer sizes:

| |
|---|
| Cropping layer |
| Normalization Layer |
| Convolution 5 x 5 x 24, 2 x 2 striding, valid padding |
| RELU |
| Dropout |
| Convolution 5 x 5 x 36, 2 x 2 striding, valid padding |
| RELU |
| Dropout |
| Convolution 5 x 5 x 48, 2 x 2 striding, valid padding |
| RELU |
| Dropout |
| Convolution 3 x 3 x 64, 1 x 1 striding, valid padding |
| RELU |
| Dropout |
| Convolution 3 x 3 x 64, 1 x 1 striding, valid padding |

| |
|---|
| Flatten layer |
| Fully connected layer, with 1164 outputs |
| RELU |
| Dropout |
| Fully connected layer, with 100 outputs |
| RELU |
| Dropout |
| Fully connected layer, with 50 outputs |
| RELU |
| Dropout |
| Fully connected layer, with 10 outputs |
| RELU |
| Dropout |
| Fully connected layer, with 1 output |

## 3. Creation of the training set and training process

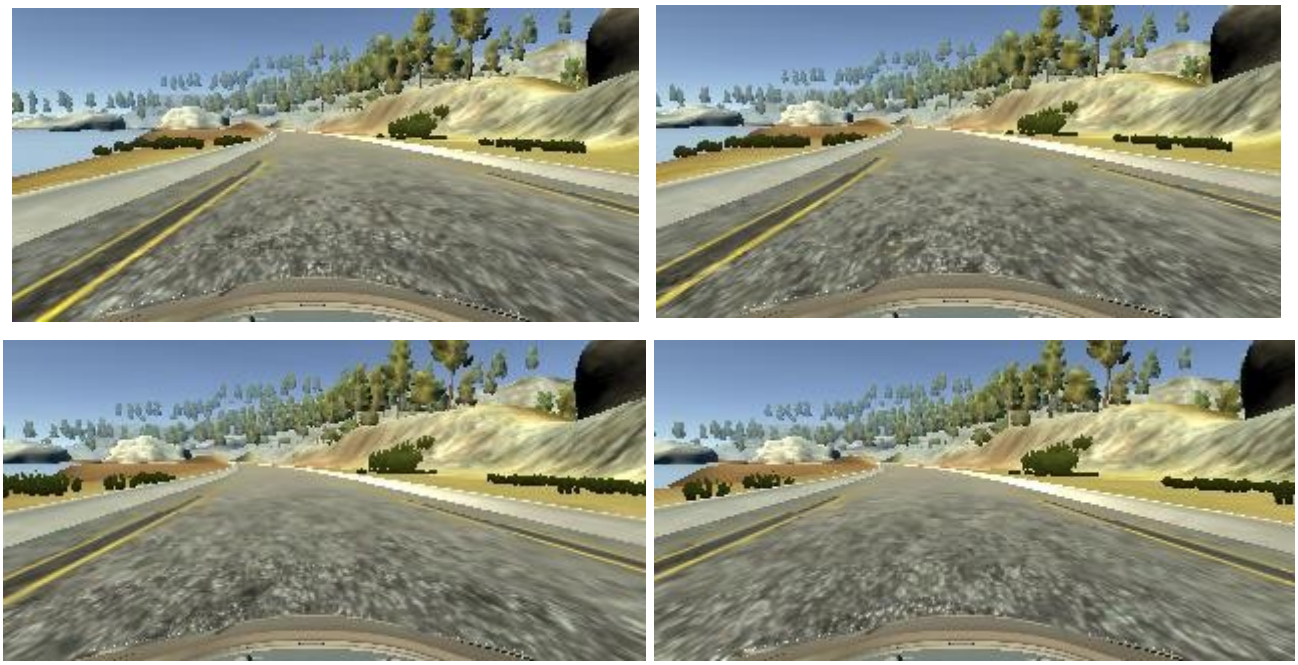
### Center driving data

Since in simulation mode, the car is driving along track 1 counter clockwise, to capture good driving behavior, I first record 2 times of 2 laps of track one using center lane driving in counter clockwise direction. Here is an example image of this data set:

### Recovering back to center data

After that I recorded the vehicle recovering from left side and right side of the road back to center for 2 laps of track one, so the vehicle would learn to recover to the center of the road when it approaches the edge, the following images show what a recovery looks like. The top left shows when the vehicle is approaching the left side of the lane. The top right and bottom left images are showing the middle state of recovering the vehicle to

the center of the lane, and the bottom right image shows the vehicle finally recovers back to the center of the lane.



**Center Driving Data clockwise**

After some tuning, I finally selected the dropout rate of my model to be 0.05 and based on the data above the model is a bit overfitting. And I decide to feed the model with more data. As mentioned, those center driving collected so far is in counter clockwise direction, I want to get some center driving data in clockwise direction. Here I make use of the sample training data provided by the course. The following is a sample image:

From the position of the rock and lake, we can see the vehicle is driving in different direction as those center driving data above.

**More Center Driving Data counter clockwise**

After applying the clockwise data set, I found the model is still a bit over-fitting. Therefore, I collected another 4 laps of center driving data in counter clockwise. And after this, the model fits quite well as shown in the section before.

**Data Augmentation**

Besides collecting different data sets, I apply data augmentation to all dataset. The data augmentation is just a horizontal flip operation. As introduced in the course, this will make the vehicle more robust for turning and also make it better staying in the center of the lane. The following is an example of the flipping, the left is the original image, and the right is the one flipped horizontally.

**Final Dataset**

After the collection process, I get 30482 data points. The preprocessing of the image is just done in the model as a normalization layer and a cropping layer. And I finally randomly shuffle the whole data set, and put 20% of the data into the validation set.

I used this training data for training the model. The validation set helps determine if the model is overfitting or underfitting. The ideal number of epochs is 10 as evidenced by those learning curves shown in section above.

And I use an adam optimizer so that manually training the learning rate is not necessary.