#Behavioral Cloning

##Writeup Template

###You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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 CNN_generator_training.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 is basically the same as the nvidia model described in the lectures with the exception of utilizing dropout layers.

The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer, as well as images cropped using a keras crop layer.

Attempts were made to use ELU layers, but those proved harder to train and performed worse than when using RELUs.

####2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting.

The model was trained and validated on different data sets to ensure that the model was not overfitting. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

####3. Model parameter tuning

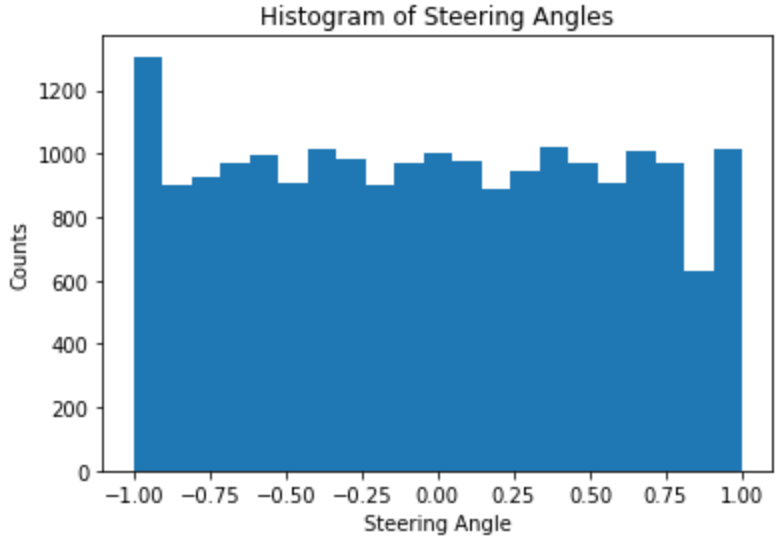
The model used an adam optimizer, but the learning rate was reduced from 0.001 to 0.0001.

####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, driving around the second track, and spending lots of time collecting data from sharp turns.

I created a script to normalize the distribution of angles that are fed into the training script. I divided the range of [-1.0, 1.0] into 21 bins and thresholded the number allowed

in these bins to 1000. After a lot of data recording the distribution looks like the following:



With the following counts:

[1000, 864, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 498, 832]

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 try to make an existing model work for my application.

My first step was to use a convolution neural network model similar to the nvidia model because it was proven to work well for this application.

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 to incorporate dropout layers after the last convolutional layer and first fully connected layer, as well as expand the data set.

After constructing the model and testing with my own dataset of a few laps around each track, the CNN was still having a somewhat difficult time keeping the car on the track and seemed to oscillate quite a bit. As with most neural network tasks, it seems that the distribution of the data set is more important than the actual format of the network. I therefore created a script to bin the steering angles into 21 bins and threshold the acceptable number of items in each bin to 1000. After quite a bit of data collection and this binning operation, I was able to have 21 bins each of nearly 1000 items, producing a very flat distribution. I then saved this new dataset as a separate csv and used that to train the model.

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

####2. Final Model Architecture

The final model architecture consisted of a convolution neural network with the following layers and layer sizes:

```
def nvidia():

    shape = (160,320,3)

    model = Sequential()

    model.add(Lambda(lambda x: x / 255.0 - 0.5, input_shape = shape))

    model.add(Cropping2D(cropping = ((50,20),(0,0))))

    model.add(Convolution2D(24,5,5, subsample=(2,2), activation = 'relu'))

    model.add(Convolution2D(36,5,5, subsample=(2,2), activation = 'relu'))

    model.add(Convolution2D(48,5,5, subsample=(2,2), activation = 'relu'))

    model.add(Convolution2D(64,3,3, activation = 'relu'))

    model.add(Convolution2D(64,3,3, activation = 'relu'))

    model.add(Dropout(0.5))
```

```
    model.add(Flatten())
    model.add(Dense(100))
    model.add(Dropout(0.5))
    model.add(Dense(50))
    model.add(Dense(10))
    model.add(Dense(1))

    return model
```

####3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded a handful of laps around track 1 going counterclockwise, then a handful more going the opposite direction. I then recorded a couple laps of recovery driving from near the edges of the lanes.

I then repeated this process one track 2 to get more data points. I then repeated portions of track 2 in both directions that had very sharp turns so that the network had enough examples of these to perform well on them.

I did not perform any dataset augmentation other than the methods described above with the histogram equalization. After the collection and equalization process, I had 20194 data points.

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 noting when the validation training loss began to increase as compared to the training data set. I used an adam optimizer with a manually overridden learning rate of 0.0001.

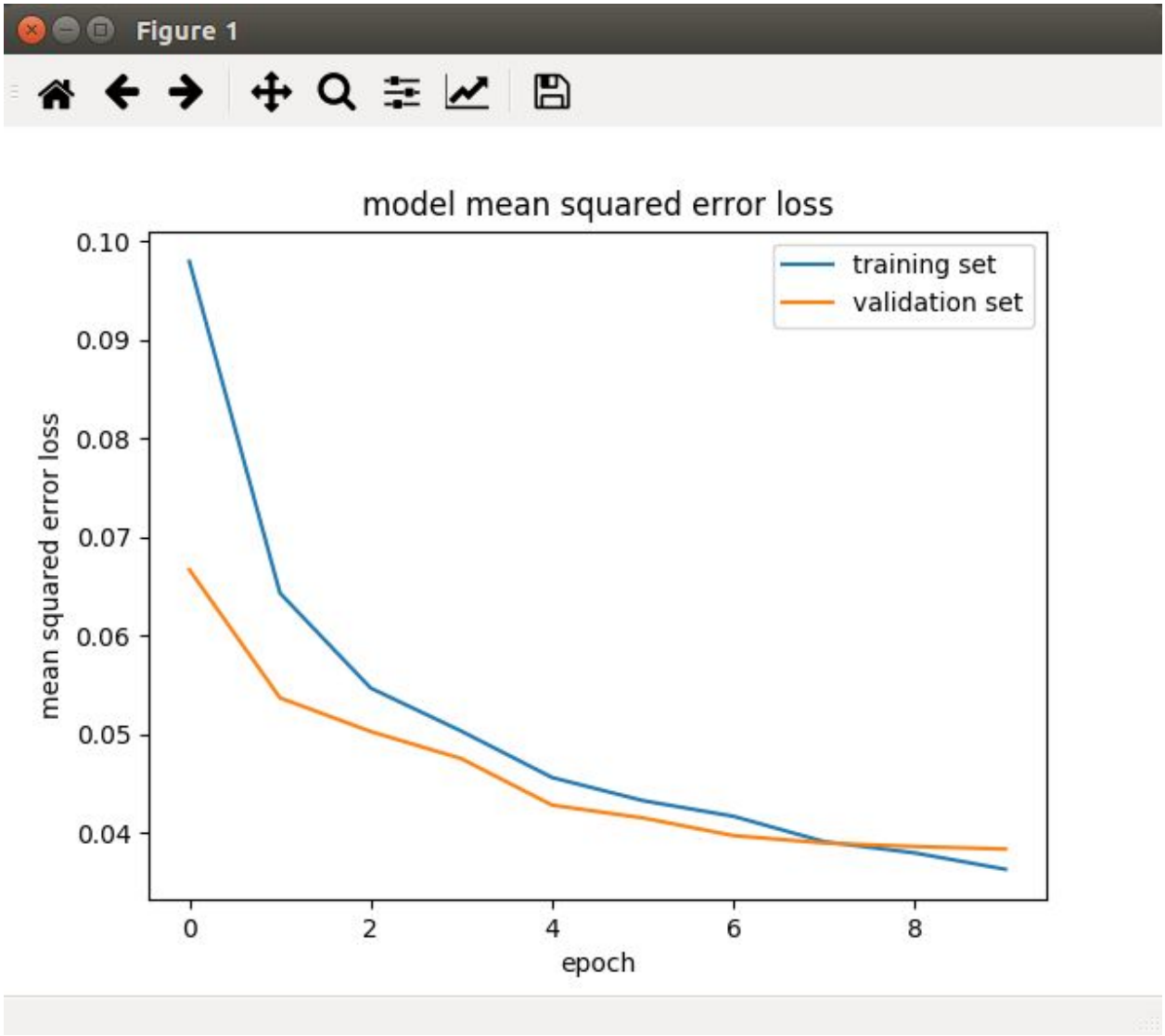


Figure 1
model mean squared error loss
training set
validation set
mean squared error loss
epoch
0.10
0.09
0.08
0.07
0.06
0.05
0.04
0
2
4
6
8