Behavior Cloning

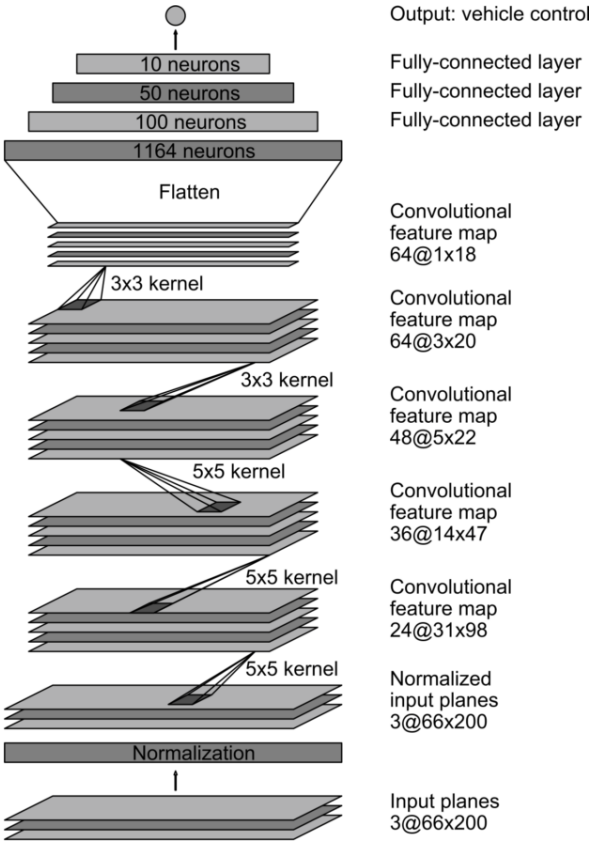
Using Convolution Neural Network

ZEESHAN ANJUM ▸ Udacity ▸ 2017-02-17

Behavior Cloning

Using Convolution Neural Network

Convolution Neural Network has been extensively used in self-driving vehicles. Here in this project I started with a very basic simple network later I added more layers modify some of these. I tried to use most from Nvidia's model.



Dataset:

I drove around 25k frames including 1st & 2nd map in the simulator. Most of the frames are from center lane driving and 20% are from side lane recovery. I used PS4 joystick as it is only the appropriate way for proper driving. My target was to train this network only for 1st map as required for this project. I read in a blog post that at least minimum 40k images are required to train a perfect model. I used 25k to save time for training the model, and to avoid memory overflow. And it works.

Following are the details of my dataset.

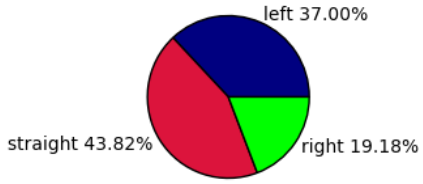
Total no. of features: 25656

Shape of single feature: (100, 220, 3)

Shape of all features: (25656, 100, 220, 3)

Shape of labels(steering): (25656,)

```
Examples data: Left: 9492, Straight 11243, Right 4921
```



Pre-Processing:

I cropped the upper area of images in the dataset, as most of the time in real world road remains same shape and color but upper environment change. And I also reduced the scale to (200,100,3) as it is enough to train the model. I didn't rotate, scale, shear the images because the purpose of this project is to test model only in the simulator and simulator is always providing good quality images.

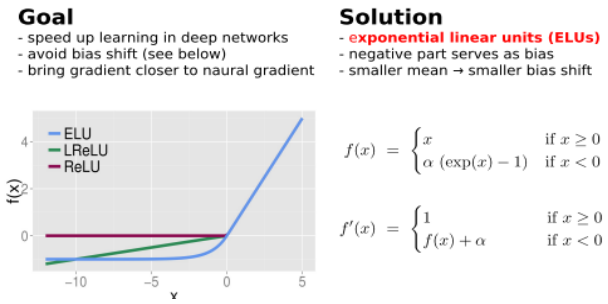
Preparing data to train:

I trained model on 20524 (80%) samples, validated on 5132 (20%) samples. For testing I used manual test. I shuffle the training and validation data as

Activation Function:

I used Exponential linear unit ELU instead of ReLU because it has fast learning curve than ReLUs.

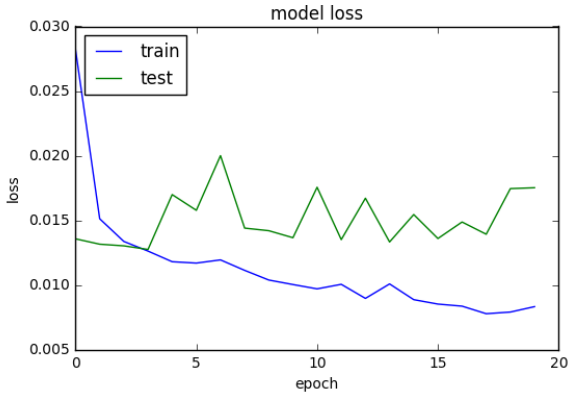
As mention in following paragraph of an article. Rectified linear units (ReLUs), leaky ReLUs (LReLUs) and parametrized ReLUs (PReLUs), ELUs also avoid a vanishing gradient via the identity for positive values. However, ELUs have improved learning characteristics compared to the other activation functions. In contrast to ReLUs, ELUs have negative values which allows them to push mean unit activations closer to zero. Zero means speed up learning because they bring the gradient closer to the unit natural gradient. Like batch normalization, ELUs push the mean towards zero, but with a significantly smaller computational footprint. While other activation functions like LReLUs and PReLUs also have negative values, they do not ensure a noise-robust deactivation state. ELUs saturate to a negative value with smaller inputs and thereby decrease the propagated variation and information. Therefore ELUs code the degree of presence of particular phenomena in the input, while they do not quantitatively model the degree of their absence. Consequently dependencies between ELU units are much easier to model and distinct concepts are less likely to interfere [*abstract of ELU-Network paper*].



Training:

I trained this model on my laptop (core i7 2.4, 8GB RAM). It took approximately 1 hour and 8 minutes. I used 20 epoch with batch size 360. I set learning rate for Adam optimizer as 0.0001 as it seems reasonable, when I tried with 0.001 lr., loss result for training become very slow over time.

Following is the graph showing loss over each epoch.



Optimizer and Loss Function:

I used Adam Optimizer, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

For this type of scenario, MSE is good, as the **mean squared error** (**MSE**) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors or deviations. Here we need the values of steering angles which aren't 100%, as good model shouldn't memorize the training data but generalize it.

Conclusion:

I find that if we train our data in fast simulator mode (which doesn't use shadow and high mip mapping on textures) and test on Good quality (which has shadows, low mip map for textures), then it would not work as we expected as in low quality image the details of road lanes and side markers are lost, and

shadows (Illumination invariance) also play a role in the accuracy of the model.

References:

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

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

https://en.wikipedia.org/wiki/Activation_function

http://image-net.org/challenges/posters/JKU_EN_RGB_Schwarz_poster.pdf

https://arxiv.org/abs/1412.6980

https://en.wikipedia.org/wiki/Mean_squared_error

https://carnd-forums.udacity.com/questions/26214464/behavioral-cloning-cheatsheet