CSCI 2271: Steering Angle Prediction for Self Driving cars

Aahlad Manas Puli
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
apm470@nyu.edu

Bharathipriyaa Thangamani
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
bt978@nyu.edu

Pratheeksha K Seetharama
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
pks329@nyu.edu

1 Introduction

We predict steering angles in the effort at teaching a car how to drive using only cameras and deep learning. Steering angle prediction is a critical component of a driverless car which makes it capable of sensing its environment and navigate. The motivation was to build a component of an end-to-end solution of a self-driving car, using only ConvNets.

2 Problem Statement

This problem was released as an open competition by Udacity as part of their Nano-degree program on Self driving cars. We use the same dataset. The purpose of the challenge is to take image frames from a camera mounted to the windshield of the car, and predict the appropriate steering angle. A similar problem is addressed by the self-driving car team at NVIDIA, in the paper End-to-end learning of Self-driving cars. [3]

2.1 Data Pre-processing

The data is organized using two comma-separated-files: camera.csv and angles.csv. camera.csv contained the mapping between time-stamp and image filename while angle.csv mapped time-stamp and angle of steering recorded. These mappings exists because the timestamps are recorded with nano-second precision and so match only upto 11 or so digits. We implement some of the pre-processing techniques used in the NVIDIA challenge as well as an implementations by the communities on Udacity's github, which we use as the baseline.

A typical image looks like in figure 1.

3 Related Work

There has been a lot of work done in the domain of self-driving cars. Companies like Comma-ai, Nvidia, Uber, Google have dedicated teams to build end-to-end systems to enable driverless cars. In particular we looked into Nvidia's model [4], whose DAVE-2 deep learning system is capable of driving in many different weather conditions, avoiding obstacles, and even going off-road. Nvidia's

Figure 1: Example image before pre-processing

model is composed of a 9-layer CNN which takes single images of a scene paired with the steering angle as output. We explore the same model and compare the performance of different parameter settings along with some other experimental models explained below.

4 Experimental Evaluation

4.1 Data

All sensor data including imagery and steering wheel angles was provided in the ROSbag format which primarily has the video frames from the camera mounted on the self-driving car. We had to extract this into individual frames. We used an existing docker container that was setup by [6], to convert ROSBag data to jpeg files. Each of the .bag files pertained to a particular driving scene. For e.g One of the bag files was a little over an hour of driving from El Camino Real and the another bag had a combination of highway driving and curvy driving over Highway 92 from San Mateo to Half Moon Bay. We chose the following three driving scenarios for our experiments .

- HMB 1: 221 seconds, direct sunlight, many lighting changes. Good turns in beginning, discontinuous shoulder lines, ends in lane merge, divided highway
- HMB 4: 99 seconds, divided highway segment of return trip over the summit
- HMB 5: 212 seconds, guardrail and two lane road, shadows in beginning may make training difficult, mostly normalizes towards the end

We have a total of 10609 images for training and 5614 images for testing. The data was cleaned by Udacity for [2] so that it doesn't contain any stop signs or other traffic signals. Each image originally is of dimensions 640x480.

4.2 Evaluation Metric

We use Root mean squared error(RMSE) as the evaluation metric in the test case. While training we used the SmoothedL1criterion(squared loss between -1 and 1, linear otherwise) for it's robustness to outliers. This decision was taken after we saw our models fail to learn anything due to exploding gradients.

4.3 Methodology

Over the course of our project, the major changes we made are the following:

1. We decided to not use data other than front-center camera image; we wanted the network to learn to recognize road outlines with just the steering angle.
2. We cropped and re-sized input images to improve execution time and performance.Reducing the amount of cropping of the original image from 200 to 100 pixels boosted the performance. From Figure 3, its clear that Cropping reduces the amount of external noise that is input to the model

3. We chose to experiment with Residual Networks[6] considering their success in object detection in addition to existing SOTA neural net models for steering prediction.
4. After a couple of trials we observed that the 'adam' optimizer provided better convergence over the regression models.
5. We experimented with different learning rates and image-formats for 3 models:
   (a) A 3-convolutional layer CNN based on comma-ai steering model[1].
   (b) 5-convolutional-layer CNN from Nvidia's paper[4].
   (c) A Residual network with 18 layers.
6. We added a small amount of translational and rotational noise without changing the steering angle. This did not affect the performance.
7. We discretized our regression output value and reduced our problem to classification with 40 classes. Our models did not do too well on this either.
8. We used the initialization from Kaiming He [6] for the Residual network and Xavier-caffe initialization
9. We resampled the dataset(uniform over 40 classes) to make sure that the extreme angles are represented well enough. We couldn't see this to completion because more often than not our models threw NANs during validation.
10. We expect the CNN to capture the curvature of the road after couple of layers. Figure 2 depicts the layer output after the first CNN layer.

(a) Original image (b) A channel from the first layer

Figure 2: Visualizing output of NVIDIA - 1st CNN-layer

(a) Original image (b) Cropped image: top 100 pixels

Figure 3: Data preprocessing - Cropping 100 pixels

4.4 Models

4.4.1 Baseline

Our baseline model is a 3-layer convolutional neural network with PReLU activation function. We used Adam optimizer. The intuition behind this model was obtained from comma.ai's self driving

car model. Since the structure of the model was quite simple, we chose to benchmark it as our baseline model .

4.4.2 Experiments with Comma-AI steering model

The results of experiments with changing learning rate, different color transforms are shown in figures 4 and 5

```
    nn.Sequential {
  [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9)
  -> (10) -> (11) -> (12) -> (13) -> (14) -> output]
  (1): nn.SpatialConvolution(3 -> 16, 8x8, 2,2)
  (2): nn.SpatialSubSampling
  (3): nn.PReLU
  (4): nn.SpatialConvolution(16 -> 32, 5x5, 2,2)
  (5): nn.SpatialSubSampling
  (6): nn.PReLU
  (7): nn.SpatialConvolution(32 -> 64, 5x5)
  (8): nn.SpatialSubSampling
  (9): nn.View(128064)
  (10): nn.Dropout(0.200000)
  (11): nn.PReLU
  (12): nn.Linear(128064 -> 512)
  (13): nn.PReLU
  (14): nn.Linear(512 -> 1)
}
```

4.4.3 Experiments with Criterion

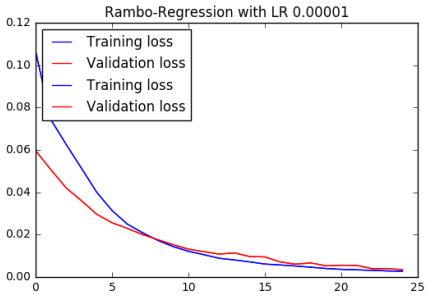
- SmoothL1Criterion
- MSECriterion

Although initially Smooth-L1-criterion did better than MSEcriterion for convergence, we realized later that this was mainly due to a problem with the initialization of network. After that was fixed, there wasn't much difference between the test errors for either criterion. This is surprising because MSE weighs the outliers error much more.

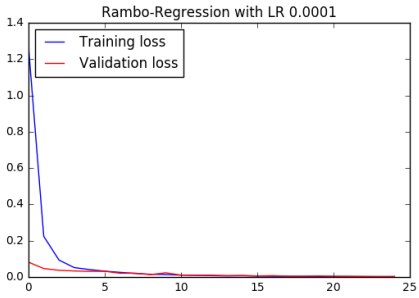
4.4.4 40-class classification with Nvidia

Although our primary problem of predicting steering angle was more suited to be a regression problem, we also transformed the problem into a classification one, where the objective was to predict the steering class. We defined 40 classes with uniform intervals of around 0.12 within the range [-2.4, 2.4]. The following experiments were performed with learning rates and color transforms. Refer Figure 7 and Figure 8

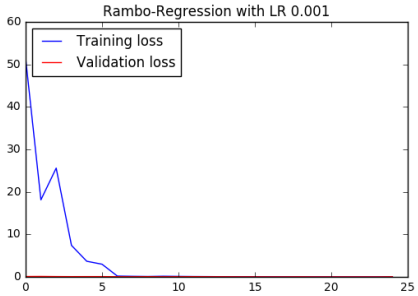
```
    nn.Sequential {
  [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9)
  -> (10) -> (11) -> (12) -> (13) -> (14) -> (15) -> output]
  (1): nn.Sequential {
    [input -> (1) -> (2) -> (3) -> output]
    (1): nn.SpatialConvolution(3 -> 24, 5x5, 2,2)
    (2): nn.SpatialBatchNormalization (4D) (24)
    (3): nn.ReLU
  }
  (2): nn.Sequential {
    [input -> (1) -> (2) -> (3) -> output]
    (1): nn.SpatialConvolution(24 -> 36, 5x5, 2,2)
    (2): nn.SpatialBatchNormalization (4D) (36)
    (3): nn.ReLU
```



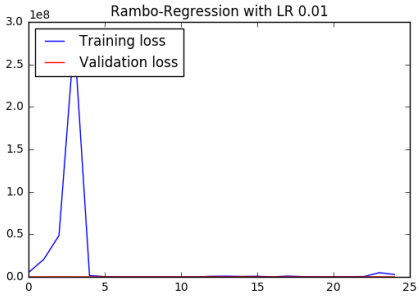
(a) Comma-ai with LR -0.00001



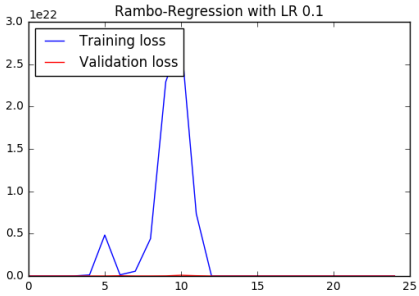
(b) Comma-ai with LR -0.0001



(c) Comma-ai with LR -0.001

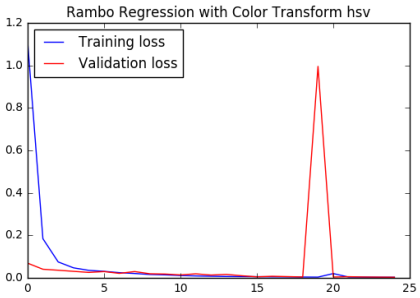


(d) Comma-ai with LR -0.01

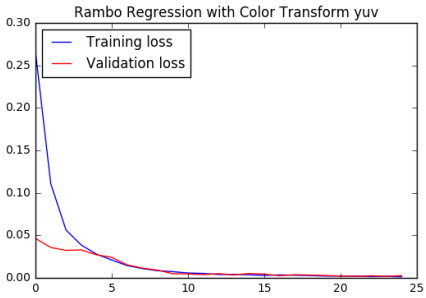


(e) Comma-ai with LR -0.1

Figure 4: Learning rate experiments with Comma-ai with batch size = 32



(a) Comma-ai with hsv transform



(b) Comma-ai with yuv transform

Figure 5: Image transform experiments with Comma-ai - Regression

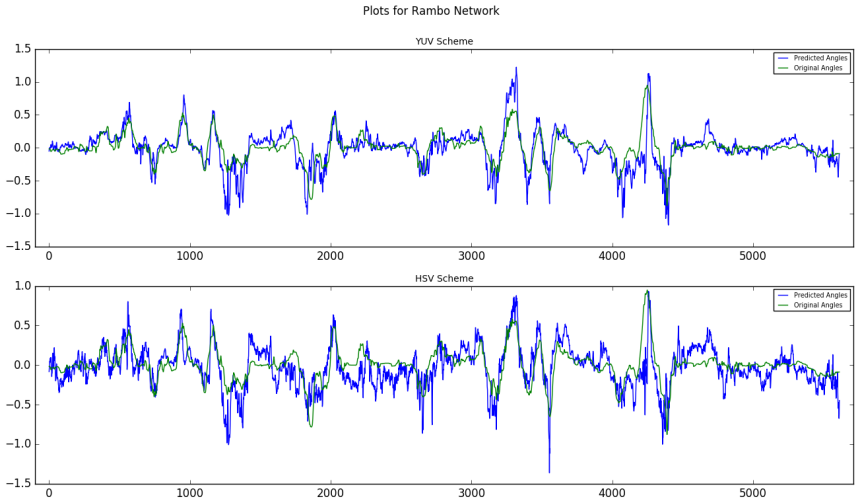
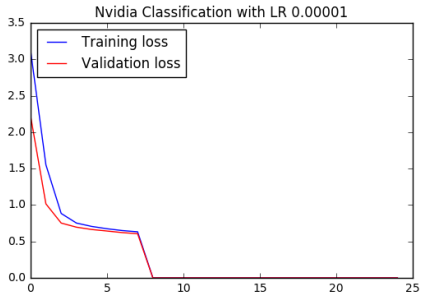


Figure 6: Comma-ai network angles with different color schemes.

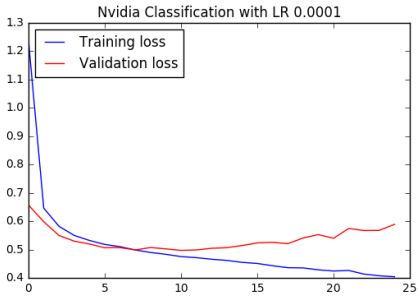
```
  }
  (3): nn.Sequential {
    [input -> (1) -> (2) -> (3) -> output]
    (1): nn.SpatialConvolution(36 -> 48, 5x5, 2,2)
    (2): nn.SpatialBatchNormalization (4D) (48)
    (3): nn.ReLU
  }
  (4): nn.Sequential {
    [input -> (1) -> (2) -> (3) -> output]
    (1): nn.SpatialConvolution(48 -> 64, 3x3)
    (2): nn.SpatialBatchNormalization (4D) (64)
    (3): nn.ReLU
  }
  (5): nn.Sequential {
    [input -> (1) -> (2) -> (3) -> output]
    (1): nn.SpatialConvolution(64 -> 64, 3x3)
    (2): nn.SpatialBatchNormalization (4D) (64)
    (3): nn.ReLU
  }
  (6): nn.View(21120)
  (7): nn.Linear(21120 -> 1164)
  (8): nn.ReLU
  (9): nn.Dropout(0.500000)
  (10): nn.ReLU
  (11): nn.Linear(1164 -> 100)
  (12): nn.ReLU
  (13): nn.Linear(100 -> 50)
  (14): nn.ReLU
  (15): nn.Linear(50 -> 40)
}
```

4.4.5 Nvidia- Regression

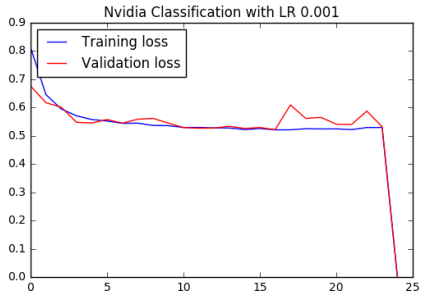
We use the same model as for classification but the final output layer is just 1 neuron. We report the errors along with the predicted-angles plotted against the actual test angles.



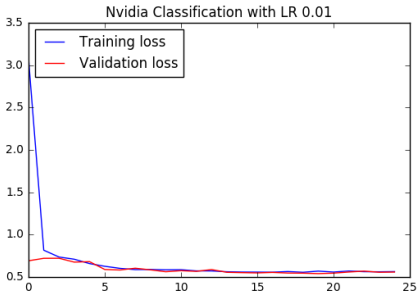
(a) Nvidia-40 with LR 0.00001



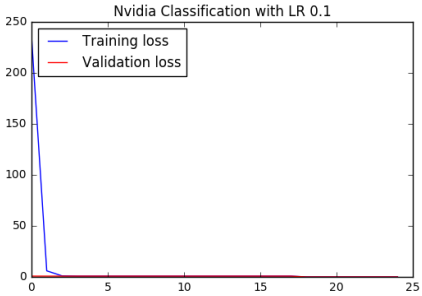
(b) Nvidia-40 with LR 0.0001



(c) Nvidia-40 with LR 0.001

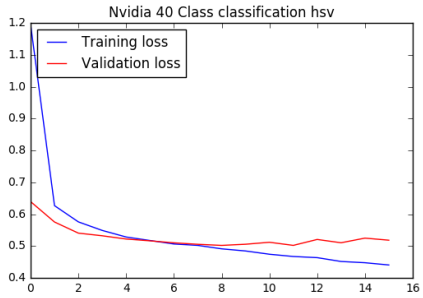


(d) Nvidia-40 with LR 0.01

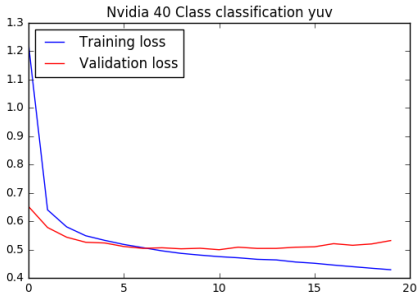


(e) Nvidia-40 with LR 0.1

Figure 7: Learning rate experiments with NVIDIA-Classification with batch size = 128



(a) Nvidia-40 class with hsv



(b) Nvidia-40 class with yuv

Figure 8: Image transform experiments with NVIDIA- Classification

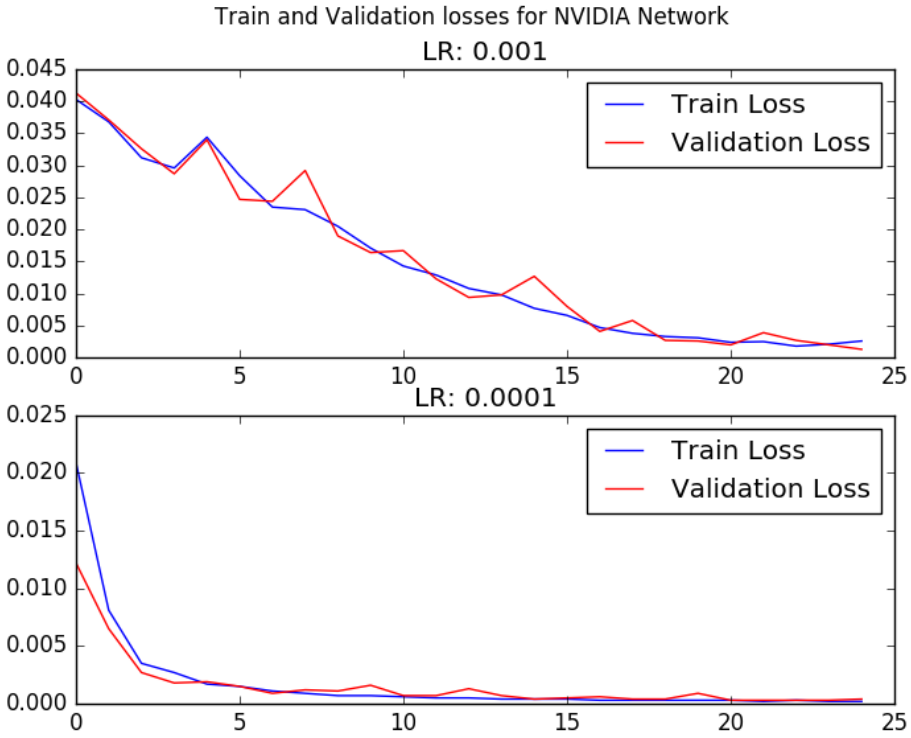


Figure 9: Nvidia network errors with different learning rates.

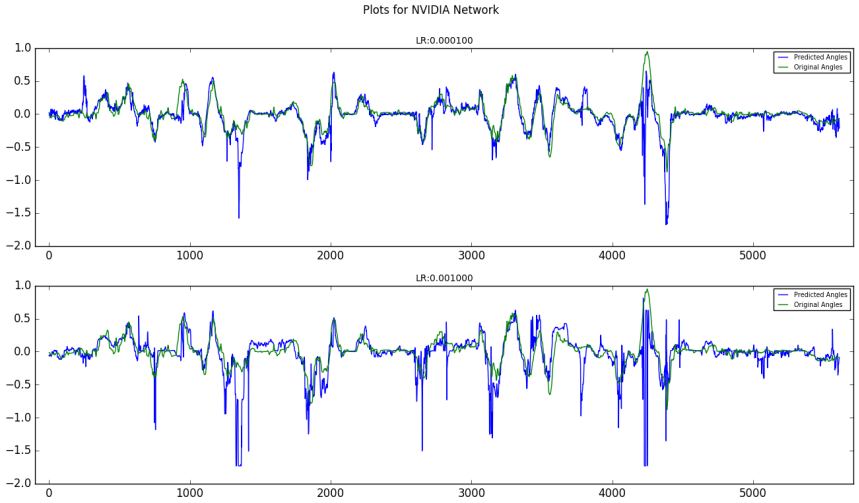


Figure 10: Nvidia network angles with different learning rates.

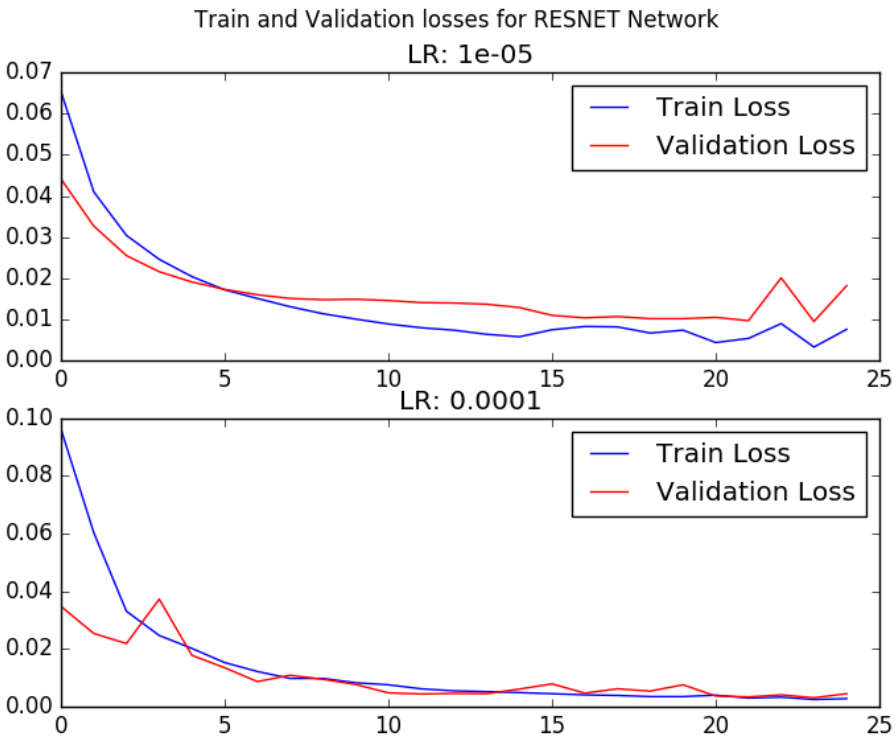


Figure 11: Resnet errors with different learning rates.

4.4.6 Residual Networks

Residual Neural Networks were presented for improving the efficiency of training deep neural networks. There is empirical evidence that residual networks are easier to optimize and substantial improvements have been obtained using residual networks on Image-Net and COCO data-sets. We used a basic residual net with 18 layers to predict steering angles. In 11 12 we report the errors along with the predicted-angles plotted against the actual test angles.

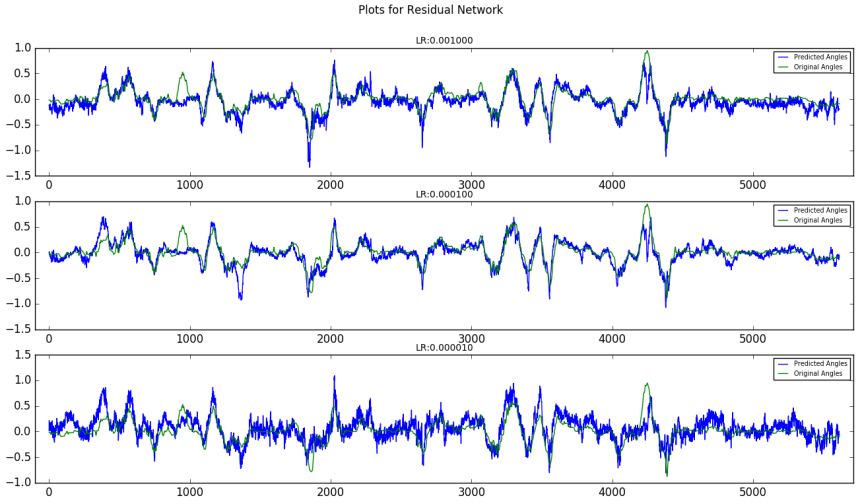


Figure 12: Resnet angles with different learning rates.

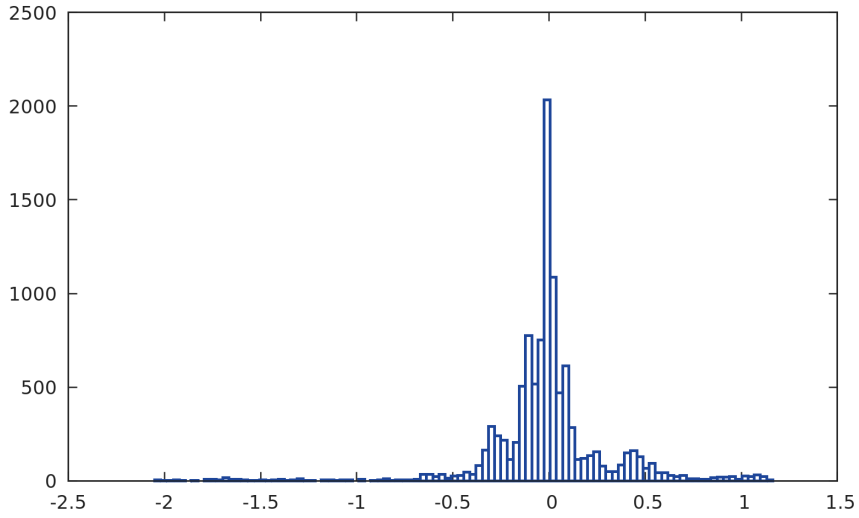


Figure 13: Data Distribution of steering angles

5 Data Resampling

We observed from out plots that a good amount of error turns up from the extreme cases. Figure 13 shows the distribution of angles in the training data:

From 13, we inferred that there a standard resampling strategy should help. In regression problems, this isn't as straightforward. So we discretized the angles into 40 classes just as before. The idea was to make each output value be represented uniformly in the first few epochs and then smoothly approach the actual distribution towards the end of the epochs. After much experimentation, we got the training error to fall similar to our original resnet model. The problem we faced was that we couldn't get the validation loss to converge while resampling the labels.

5.1 Results

Model	Train loss	Valid loss	Test Error
Baseline	0.0012	0.0026	0.04
Nvidia-40 class	0.44	0.51	-
Resnet	0.0028	0.0045	0.019
Nvidia-regression	0.0002	0.0004	0.03

We report MSE for Resnet and Baseline , Smoothed-L1-loss for Nvidia network regression and Cross-Entropy fro Nvidia network Classification. We switched to Smoothed-L1-loss early on because our models weren't converging. The Nvidia-regression results are from then. The final test-error turned out to be the same if the model converged for the same number of epochs. Later on we moved to ADAM optimization and specific random initialization procedures like in [5]and got both Resnet and Nvidia to converge. Considering this MSE seemed like a more natural measure.

6 Conclusion and Future Work

Our networks fail to recognize the relation between the trajectory of the road and the steering angle over time. This is evident from the noisy prediction angles in the plots of steering angle vs. time. So, one natural way to improve upon this would be to stack a CNN under an RNN(and variants like LSTMs or GRUs) and run a truncated back-propagation through time. Because the steering angle would not depend on any image more than approximately 30 frames away, the truncation would be a good idea. We expect this to drastically improve the performance but that would require much more data.

Another possible approach would be to skew the data distribution to favor extreme angles instead of 0 angles, (driving straight), rather than considering a uniform data distribution through resampling. This may enable ConvNets to learn the relation between the curvature of the road and the steering angle better.

We also propose to train a classifier for the discretized output values and then add this output class vector to the penultimate fully-connected layer, in another network, and regress for the steering angles. This would help in smoothing out the variations in the predicted steering angles.

7 Bibliography

1. https://github.com/commaai/research
2. https://www.udacity.com/self-driving-car
3. https://github.com/udacity/self-driving-car/tree/master/datasets/CH2
4. End to End Learning for Self-Driving Cars https://arxiv.org/abs/1604.07316
5. http://wiki.ros.org/rosbag
6. https://github.com/Kaixhin/nninit
7. https://github.com/facebook/fb.resnet.torch
8. https://github.com/rwightman/udacity-driving-reader
9. Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In International Conference on Artificial Intelligence and Statistics.
10. He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. arXiv preprint arXiv:1502.01852.