Applying Techniques in Supervised Deep Learning to Steering Angle Prediction in Autonomous Vehicles

Petar Penkov[1], Vinay Sriram[2], and James Ye[3]

[1]*ppenkov@stanford.edu*
[2]*vsriram@stanford.edu*
[3]*yej@stanford.edu*

I. Overview

The development of effective autonomous vehicles is, in today's day and age, an undoubtedly popular application of research in machine learning and control. One of the most fundamental tasks in the design of algorithms to control autonomous vehicles is steering angle prediction based on real-time driving data. In this paper, we discuss the application of techniques in supervised learning to predict the vector of steering angles required to navigate a car through some specified trajectory.

The attributes upon which we construct features and train supervised learning algorithms are provided in a dataset recently released by Udacity [1]. Each file in the dataset describes a trajectory (roughly a 15 to 45 minute drive), over the duration of which attributes are collected. For each point along the trajectory, the steering angle at that point is recorded along with [1] the GPS coordinates of the car, [2] the instantaneous speed at which the car is traveling, and [3] a set of 3 images $\{\texttt{left}, \texttt{center}, \texttt{right}\}$ captured by onboard forward facing cameras in the vehicle.

Our analysis of the problem can be split broadly into three phases, with primary emphasis on the last phase. The first phase concerns the prediction of steering angle from GPS coordinates and speed alone. The second phase of the analysis concerns the evaluation of conventional techniques in supervised learning (such as support vector regression) to predict steering angle from (1) pre-processed dataset images and (2) extracted features of these images. The third and final phase of the analysis concerns the evaluation of deep learning schemes to predict the steering angles (from both raw images and pre-processed ones). For this last phase in particular, we consider both a multi-layer convolutional neural network and a multilayer perceptron model.

II. Phase I

A. Data Pre-Processing

In order to make the GPS data useful, we perform the following preprocessing steps to improve the generalization ability of any learning model we wish to fit.

1) We remove all points corresponding to a speed of zero, as steering angle measurements taken when the car is not moving are meaningless.
2) We construct a normalization mapping from longitude and latitude to points (x, y) in 2D space.The very small local region of the Earth over which the test drives were done can be accurately approximated as a plane, and so the drive region is a latitude/longitude square centered at $(37.05, -122.05)$ of side-length 0.1. In Euclidean space, this is a rectangle with side lengths 8.88km and 11.12km. We use these values to normalize the data.
3) We apply a Savitsky-Golay filter (with a second order polynomial over intervals of 51 points) in order to remove high-frequency noise from the GPS measurements.

B. Limitations of the Physical Model

The GPS coordinates essentially define a 2D curve over which the car travels. In theory, there is an explicit solution for the steering angle of the car which depends on the parametric equations of the curve, the car length, L, and the steering-angle-to-wheel turning ratio, n (which specifies how much the wheels turn for every degree the steering wheel turns). It can trivially be derived that the steering angle is related to the GPS trajectory of the car $\vec{r}(t) = (x(t), y(t))$ by:

$$\alpha(t) = n \arcsin L \frac{x'(t)y''(t) - y'(t)x''(t)}{\left(\sqrt{(x'(t))^2 + (y'(t))^2}\right)^3}$$

We are limited, however, by the fact that the above approach is only valid if the car can be reasonably approximated as a line segment of length L and the road as a class C^2 smooth curve (a curve for which $x(t)$, $x'(t)$, $x''(t)$, $y(t)$, $y'(t)$, and $y''(t)$ are all continuous). Obviously, in practice, with our given data, this is not at all the case despite our smoothing of the signals $\vec{r}(t)$ and $\vec{r}'(t)$. The results indicate that direct computation of steering angle based on physical principles is highly impractical. It leads to intolerable RMS error between true and computed steering angle signals, order-of-magnitude over-prediction of turns, and failure to recognize bimodalities (situations in which a second turn is started before an initial turn has finished).

C. Support Vector Regression on GPS Data

Due to the limitations of the physical model, we construct features to train a support vector regression algorithm based on the locality (neighborhood) of every training example $r_i = (x^{(i)}, y^{(i)})$. This locality is defined with some constant k as the k points in either direction of r_i. The precise features are the first and second derivatives of $x(t)$ and $y(t)$ at r_i, and the following:

- For all pairs of consecutive points in the locality, the difference between their first derivatives (scaled by the average speed at the two points).
- For all pairs of consecutive points in the locality, the difference between the second derivatives.

We make use of a MATLAB framework for support vector regression, using a Gaussian kernel. The results of this model, however, were not particularly promising. In the training data, the turns (including bimodal turns) are indeed recognized, but there exists a substantial and intolerable discrepancy in the turn magnitude. The generalization error is also high, indicating that the model is under-fitting and has high bias (fig. 1).

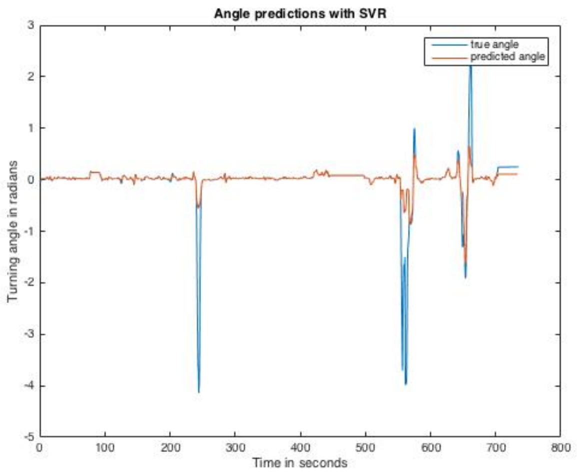

Fig. 1. Support vector regression on interpolated GPS coordinate data.

These results are unsurprising given that GPS measurements are sparse and guarantee 95% accuracy of prediction within an error margin as large as 8 meters [5]. Because of the data sparsity and high probability of error in any given measurement, the interpolated GPS data that is matched to steering angles is likely not an accurate model of the specific turns in the road. The Support Vector Regression model failed to fit the data, and the physical model was inaccurate presumably because of the data sparsity and high error margin in the measurement.

III. Phase II

In light of the ineffectiveness of supervised learning on position data, we evaluate supervised learning strategies that aim to predict steering angle based on the provided images.

A. Image Preprocessing

We apply a pre-processing stage in order to extract aspects of these images relevant to angle prediction (fig. 2). The following steps are taken to prepare the images for the various learning algorithms we evaluate.

1) All data points with an associated speed less than 50 miles per hour are not taken into consideration. This restriction allows us to reduce the size of the data set from 600,000 to 100,000 points and significantly improves the speed of training to several hours. This also has the effect of restricting our dataset to highway driving, in which turns are more subtle and thereby need to be more accurately predicted.
2) Images are next passed through a Sobel filter for edge detection. The Sobel operator convolves input images X' with 3×3 kernels such that only the edges present in the images are preserved. Specifically, we define:

$$(S_X, S_Y) = \left(\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * X', \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * X' \right)$$

Then, we separately apply a thresholding filter to the images to extract all pixels that have near-white color. Specifically, for each (i, j) in the output image X, we set $X_{(i,j)} = 255$ if either of the following are true, and set $X_{(i,j)} = 0$ otherwise.

- The Sobel gradient magnitude $||S^2_{X_{(i,j}} + S^2_{X_{(i,j)}}||$ is above some cutoff threshold.
- The grayscale value of the pixel $X'_{(i,j)}$ is above some cutoff threshold.

3) Finally, all images are downsampled from 640x480 to 64x48, and then further reduced to 64x24 by removing the top half of the image. This operation is permissible because the top half contains no road features and is therefore irrelevant. This downsampling and cutting reduces the image size by a factor of 200, which significantly improves speed of training and testing while eliminating irrelevant features.

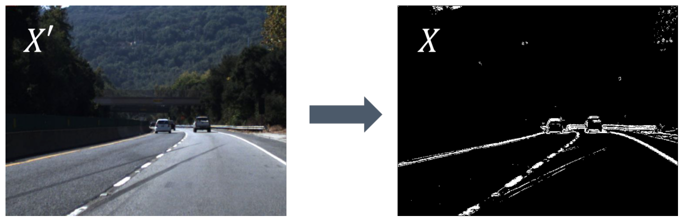

Fig. 2. Transformation of raw images to pre-processed images via the use of a Sobel kernel and thresholding.

B. Support Vector Regression on Image Data

One approach to angle prediction on image is Support Vector Regression on vectorized binary data. The vectorized data simply encodes each pre-processed image (as specified in the above section) as 1536-dimensional bit vector in which the pixel rows are concatenated, 255 is encoded as 1, and 0 is encoded as 0. Despite the high dimensionality of the feature space, this model suffers from high bias. On a given dataset with a 70:30 train-to-test partition, this model not only experiences very poor generalization, but also fails to fit the training data itself (for gaussian and polynomial kernels).

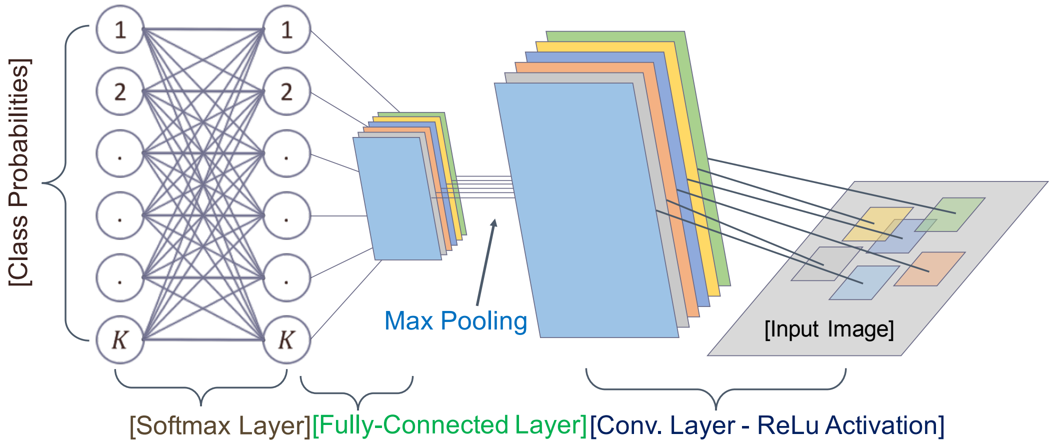

Fig. 3. Convolutional Neural Network Architecture.

C. Feature Extraction from Images

An alternative idea is to first extract relevant features of the image, and then apply Support Vector Regression. The specific feature of interest is the curvature of the road, which we model using the following scheme. For every row i in the image, we iterate across the pixels of the row and determine the largest stretch of black pixels $\{p_{i1}...p_{ik}\}$ that is contiguous. Then, we determine the average of this stretch a_i, and append this value in a vector. After running this on each row, we fit a polynomial through the determined row centers $\{a_1...a_{24}\}$. This polynomial provides a rough heuristic for the "curve" of the road. On a given dataset with a 70:30 train-to-test partition, this model too experiences the same issue as the method detailed in part B: namely, that the model is high bias and has poor fitting and generalization.

IV. Phase III

Evidently, conventional techniques in supervised learning on the provided image set are ineffective to accurately predict steering angles. We therefore turn to deep learning techniques, implemented using a MATLAB framework for training and testing different neural network architectures.

A. Steering Angle Discretization

We simplified the problem from a regression problem of continuous values to a classification problem in which the steering angle is discretized. Specifically, we assign angles to classes, each of which spans 0.01 radians. This scheme is justified because the jitter for a human driver (defined as the deviation from 0 radians when driving on a straight road) is measured to be within an error of 0.01 radians. This scheme produces a total of 101 classes (50 classes on each side of the 0-class) to encapsulate the full range of turning.

B. Convolutional Neural Network Model

In particular, we consider a convolutional neural network model (fig. 3) that has the following architecture: [Input Layer, Convolutional Layer with ReLu activation, Downsampling Layer, Fully-Connected Layer, Softmax Layer].

The input layer consists of one-channel 24×64 images, and the convolution layer uses 20 different 5x5 image kernels, with weights initialized randomly (i.e. using a gaussian distribution). The ReLu activation function simply employs the standard $max(x, 0)$ thresholding. This is immediately followed by a max pooling layer has a stride of 2, which allows for non-overlapping pooling regions. This layer downsamples by a factor of 2 in either dimension simply by extracting the max pixel within each pooling rectangle. Because we reduced the problem to classification on K labels, terminating the algorithm with a K-neuron softmax layer is the natural method of generating class probabilities.

The forward propagation [6] algorithm for neural networks of this form is fairly standard. We define it here for our specific problem. Take $\{a^{(1)}, a^{(2)}, a^{(3)}, a^{(4)}\}$ to be the activations of the various layers. For the first layer, we have that for $i \in \{1, .., 20\}$: $a_i^{(1)} = max(0, W_i^{(1)} * X)$ where $X \in \mathbb{R}^{24\times 64}$ is an input training example and $a_i^{(1)} \in \mathbb{R}^{24\times 64}$. For the second layer, we have $a_i^{(2)} = \texttt{downsample}(a_i^{(1)})$. For the third layer, we must define $a^{(2)}$ to be a vectorized form of all of the downsampled matrices $\{a_1^{(2)} \dots a_{20}^{(2)}\}$ so that we can write $a^{(3)} = W^{(3)}a^{(2)} + b^{(3)}$. Note that this layer does not have an activation function; it is simply a FC layer. Finally, we have $h_{W,b} = a^{(4)} = f(W^{(4)}a^{(3)} + b^{(4)})$ where f is the softmax function. We define for $k \in \{1...K\}$ that:

$$f_k(z) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Note that each $f_k(z)$ defines the class probability $P(y = k|X)$. We seek to minimize the cost function over m training examples. This cost function may be defined simply as:

$$J(W, b) = \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(X^{(i)}) - y^{(i)}\right\|^2\right)\right]$$

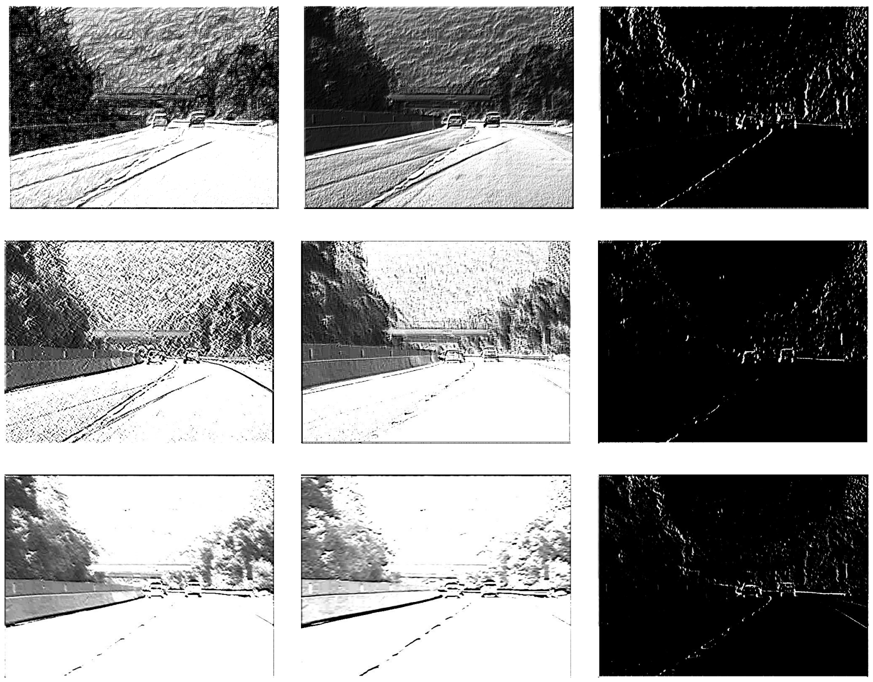

Fig. 4. Trained 5x5 Image Kernels for Convolutional Layer

In order to minimize this cost, the network is trained using a back-propagation algorithm with stochastic gradient descent. For our particular run, we used a learning rate of 0.1 and chose to terminate the learning after 20 epochs. Back propagation [6] updates for normal layers is defined by:

1) $\nabla_{W^{(l)}} J = \delta^{(l+1)} (a^{(l)})^T$
2) $\nabla_{b^{(l)}} J = \delta^{(l+1)}$
3) $\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$

For convolutional layers [7], we instead have:

1) $\nabla_{W_k^{(l)}} J = \sum_{i=1}^{m} (a_i^{(l)}) * \texttt{rot}(\delta_k^{(l+1)})$
2) $\nabla_{b_k^{(l)}} J = \sum_{a,b} (\delta_k^{(l+1)})_{a,b}$
3) $\delta_k^{(l)} = \texttt{upsample}\left((W_k^{(l)})^T \delta_k^{(l+1)}\right) \bullet f'(z_k^{(l)})$

The update rule for stochastic gradient descent is given as follows for each layer l:

$$(\Delta W^{(l)}, \Delta b^{(l)}) := \left(\Delta W^{(l)} + \nabla_{W^{(l)}} J, \Delta b^{(l)} + \nabla_{b^{(l)}} J\right)$$

$$(W^{(l)}, b^{(l)}) := (W^{(l)}, b^{(l)}) - \left(\frac{\alpha}{m} \Delta W^{(l)}, \frac{\alpha}{m} \Delta b^{(l)}\right)$$

C. Multi-Layer Perception with Image Pre-Processing

After training on the raw image data using the aforementioned convolutional neural network scheme and viewing the images resulting from the 5×5 convolutions (fig. 4) on a test input, we observe the following. Many of the kernel-transformed images are remarkably similar to the output of the Sobel kernel with thresholding. This motivates one final experiment: evaluating a neural architecture consisting of only a multilayer perceptron (MLP) model [6] in which the input layer consists of the Sobel-filtered images, the hidden layer is a fully-connected layer, and the output layer is a softmax layer. Gradient descent and back propagation are done in much the same manner.

D. Results

The RMS error in the training data is 0.0126 for the convolutional network (trained/tested with raw images), and 0.0241 for the MLP model (trained/tested with Sobel-filtered input images). Clearly, both models effectively fit the provided training data, though the convolutional model is measurably better-fitting than the MLP model. This result indeed generalizes to unseen data. On the test set, the RMS error for the convolutional network model is 0.0238, while the RMS error for the MLP model is 0.0327. These RMS values are visually consistent with an extracted segment of training data (fig. 5), using which we compare the two methods. Clearly, the subtleties of the turns in the "true" steering angle signal are better-predicted using the convolutional model. However, both produce respectable performance.

V. Conclusions

The results clearly indicate that while conventional learning techniques (support vector regression, in particular) suffer from high bias and are able neither to effectively fit training data nor generalize to test data, techniques in deep learning are able to achieve acceptable errors in both training and test on images. Given the specified discretization scheme, we observe that both the MLP model with Sobel-filtered input images and the full convolutional neural network model predict steering angle effectively. While the latter is noticeably more accurate in predicting the subtleties of turning behavior (and thereby exhibits lower test RMS error), the former has lower computational cost. Our runtime benchmarking in the

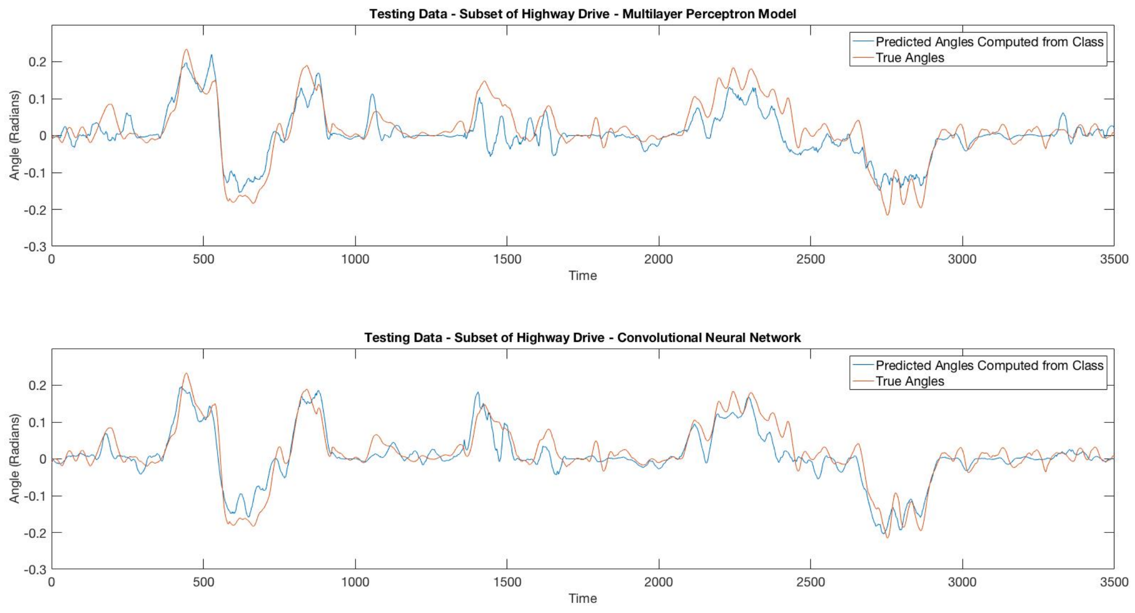

Fig. 5. Prediction on a selected subset of testing data, generated using the multilayer perceptron model on Sobel-filtered input images (left) and using the convolutional neural network model on raw input images (right).

MATLAB framework indicates that a single frame can be classified by the MLP network roughly 20 times as fast as a single frame can be classified in the Convolutional Network. Thus, in design situations where control frequency must be prioritized over high precision, one may opt to use the MLP with Sobel-filter preprocessing rather than a convolutional network.

VI. Future Work

When evaluating the deep learning networks, we restricted the training data to highway speeds, which typically have more gradual turns and clearer lane markers. However, in the future, we wish to generalize these prediction results to data on local roads and driving situations under which turns are sharper and less clearly marked. For this, we must implement neural networks with a much higher number of convolutional layers and pooling stages. Moreover, our discretization scheme can be made more fluid by both expanding the number of class labels and collecting data such that all class labels contain sufficient training volume (rather than disproportionately high training volumes in the low-magnitude angle classes). One final method of feature extraction that might provide effective steering angle prediction is the computation of cross-image gradients. Within the viscinity of a particular steering angle measurement (considering k frames backward), such gradients could make use of the way in which features change between images in order to determine the magnitude and duration of turns. Even in situations where conventional lane markers and road signs are not present can potentially be predicted.

VII. Acknowledgements

We would like to thank Dr. Andrew Ng and Dr. John C. Duchi for their instruction and for introducing us to many relevant concepts and ideas in machine learning. Additionally, we would like to thank Mr. Nihit Desai for his guidance regarding the analysis and techniques presented in this paper.

References

[1] Cameron, Oliver. ”Open Sourcing 223GB of Driving Data.” Medium, 05 Oct. 2016.
[2] Pomerleau, Dean A. ”Neural Network Perception for Mobile Robot Guidance.” (1993): n. pag. Web.
[3] ”End-to-End Deep Learning for Self-Driving Cars.” N.p., 25 Aug. 2016. Web.
[4] Wang, Zhangyang, Shiyu Chang, and Yingzhen Yang. ”Studying Very Low Resolution Recognition Using Deep Networks.” Computer Vision and Pattern Recognition (n.d.): 4792-800. Print.
[5] ”GPS Accuracy.” GPS.gov: GPS Accuracy. N.p., 5 Dec. 2016. Web.
[6] ”Multi-Layer Neural Network.” Unsupervised Feature Learning and Deep Learning Tutorial. Stanford University, n.d. Web.
[7] ”Convolutional Neural Network.” Unsupervised Feature Learning and Deep Learning Tutorial. Stanford University, n.d. Web.