A Behavior Reflex Approach to Autonomous Driving using Deep Regression Networks

E4040.2017Fall.MECH.report.bj2364.rpn2108.rs3226

Brian Jin, Rayal Raj Prasad
Dept. of Mechanical Engineering
Columbia University
bj2364, rpn2108

Rajinder Singh-Moon
Dept. of Electrical Engineering
Columbia University
rs3226

***Abstract*—Situational awareness is a key component to autonomous driving. In this project, we implement a vision based approach for autonomous driving using data collected from a driving simulator. We compare two successful architectures, AlexNet and NvidiaNet, and select the best architecture for our predictive model. Results showed that our network identified relevant features from front-facing images, in both simulated and real environments, and inferred steering information.**

I. Introduction

Current research in autonomous driving uses a combination of high-end range sensors and vision sensors. While these provide more information than a standalone camera, they are cost-prohibitive and computationally expensive. Alternatively, cameras are low-cost and produce information at higher resolutions even when compared to state-of-the-art laser sensors [2]. Additionally, modern computer vision techniques can reliably detect important features for autonomous decision-making [10]. As a result, combining vision algorithms with image data is a promising low-cost solution to autonomous driving.

There are two major paradigms for vision-based autonomous driving [1]. The first is a mediated perception approach, which recognizes important driving-related features (i.e. traffic signs, pedestrians, lanes, cars, etc.), and uses these to extract information on the vehicle's immediate state. An AI-based engine can then decide on its action using this information. The major disadvantage of this approach is that many features detected are irrelevant to the decision-making process, which adds unnecessary complexity to an already difficult task.

The second paradigm refers to a behavior reflex approach, which directly map input data to a driving command through end-to-end decision making. While this approach is simple in concept, the trained decision-making model can run into problems executing complicated maneuvers, such as weaving through traffic. These problems are a result of the unpredictability in human action. For the same image of a car in front, a human driver can choose to make a turn or stay in the lane, which is difficult to learn. Additionally, behavior reflex approaches are limited only to low-level decisions.

In this project, we employ a behavior reflex framework to implement a self-driving car on an open track. For this application, a behavior reflex approach is better suited due to its computational efficiency and performance in uncluttered environments. The environment used for simulating the race car was TORCS, a race car simulator with real physics. The goal is to develop an optimized architecture for training the decision-making model and demonstrate its performance. By implementing the car on an open track, we eliminate many of the problems associated with behavior reflex approaches, mainly making high-level decisions in cluttered environments. Another challenge of this project was data collection and testing the performance of the model, and this was addressed using shared memory with runtime integration.

II. Summary of The Original Paper

A. Methodology

The original paper from Chen et al. [1] implemented a novel approach, a hybrid between the Mediated Perception approach and the Behavioral approach, to solve the self-driving car problem. An illustration of the three different approaches is described in Fig. 1. The approach, called the direct perception approach, learned to map image inputs to a set of affordances, which were a compact set of features that described the car's current state. The affordances were used as inputs to a policy that computed high level driving decisions.

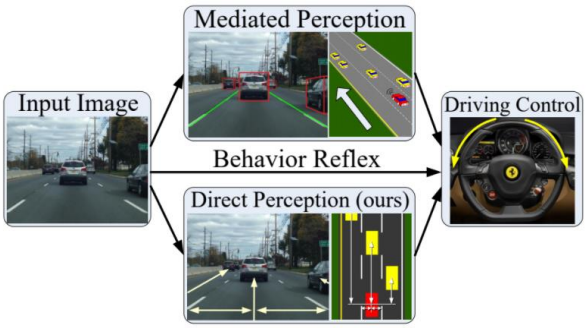

Fig. 1. Approaches to vision-based autonomous driving. The third (Direct Perception) approach represents the work done by the Vision and Robotics Lab at Princeton University.

For an open track, the direct perception approach is unnecessary because there are no obstacles on the road. Instead, we implement a behavior reflex approach, which should intuitively produce an acceptable performance on the road. We demonstrate that the training process for our model is optimal and test this in simulation.

B. Key Results

Qualitatively, the direct perception approach performed well in the TORCS simulation. No collisions were observed, and the controller would only occasionally overshoot during lane switching.

Results showed that the direct perception approach outperformed both the behavior reflex and mediated perception models. The behavior reflex model performed well for open tracks, but produced erratic trajectories when traffic was present. The mediated perception model produced larger mean absolute errors between affordance estimations and ground truth distances compared to the errors produced by the direct perception model.

The direct perception model was also tested on real world data. The model was able to determine the correct lane position, localize the car, and recognize lane transitions. Overall, the original paper presented a novel autonomous driving paradigm and proves that the approach can perform well in simulated and real environments.

III. Methodology

In this section, we describe our approach to training a decision-making model for autonomous driving. The objectives and challenged are discussed in detail in the following subsections.

A. Objectives and Technical Challenges

The primary objective was to compare different architectures for autonomous driving and demonstrate their performance on an open track. The architectures reviewed in this paper are AlexNet and NVIDIA's net, both of which have demonstrated success in recent studies. Our comparison methodology includes hyperparameter sweeps as well as architecture refinements.

The main challenge of this project was data acquisition. The Vision and Robotics Lab at Princeton University has training data available online. However, the data set is not labelled for end-to-end learning and was collected on modified tracks unavailable in TORCS by default. As a solution, we generated our own data in TORCS using a programmed robot that automated the process. This data collection method is described in detail in Section 4.2.

The other major challenge was integrating the system. TORCS was developed in C++, while TensorFlow models must be read in Python. This meant that a common runtime had to be set up for these components to communicate seamlessly. Our solution was to use ROS as a runtime environment that interfaced both TORCS as well as TensorFlow.

B. Problem Formulation and Design

Vision-based autonomous driving relies on accurate extraction of pertinent details from a matrix of numerical values, namely an image from the driver vantage point. This problem can be represented by a function (ψ) that projects a *d*-dimensional feature space (pixels values in an image, X) to a scalar (steering command) as shown in (1).

$$\psi(\boldsymbol{X}, \Theta) \rightarrow \delta, \qquad \boldsymbol{X} \in \mathbb{Z}^d, \delta \in \mathbb{R} \tag{1}$$

Here, Θ is the set of parameters that yield direct mapping from image to steering command. For a given set of images, $\boldsymbol{X} = \{X_0, X_1, \dots X_N\}$ with corresponding ground truth steering commands $\delta = \{\delta_0, \delta_1 \dots \delta_N\}$, the supervised learning problem aims to determine parameter estimate $\widehat{\Theta}$ that minimizes the discrepancy between the estimated steering command $\hat{\delta}$ and the actual steering commands (loss function L). In this work, the Mean Squared Error (MSE) is employed as the cost function and the optimization problem are shown in (2).

$$\begin{aligned} &argmin_{\widehat{\Theta}} L\left(\hat{\delta}\left(\widehat{\Theta}\right)\right) \\ &L\left(\hat{\delta}\left(\widehat{\Theta}\right)\right) = \frac{1}{n}\Sigma\left(\hat{\delta} - \delta\right)^2 \end{aligned} \tag{2}$$

The loss function is minimized using the Adam optimizer, a variant of Stochastic Gradient Descent.

IV. Implementation

Implementation details can be split into two parts: the back-end system, which describes the training process for the deep learning network, and the front-end, which describes the interface between the trained model and TORCS.

A. Back-end: Deep Learning Network

Two architectures are explored in this study, AlexNet and NvidiaNet (refer below). Chen [1] trains his direct perception model using AlexNet [3] which includes a total of 5 convolution layers and 3 fully-connected layers. The architecture can be seen in Fig. 2.

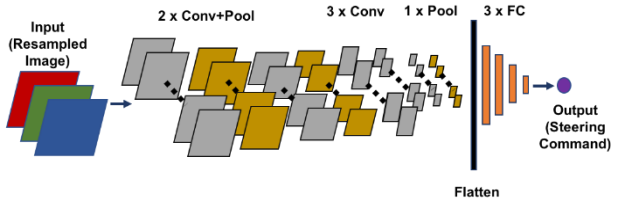

Fig. 2. AlexNet architecture used by Chen et al.

NVIDIA [9] also implemented their own end-to-end CNN model and demonstrated its performance on a real vehicle. We modified the architecture for our problem, and the new model is illustrated in Fig. 3. Note how the architecture is less complex compared to the AlexNet architecture.

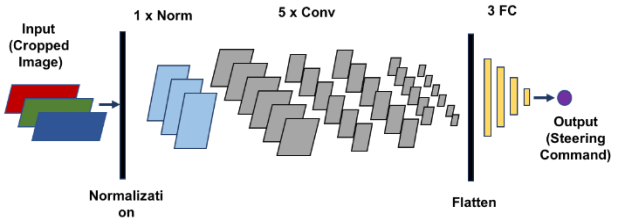

Fig. 3. Architecture used by NVIDIA Corporation.

To adapt the architecture for the problem, the input layer was modified to accommodate a rectangular cropped image of 66x200x3. The image was subject to AlexNet computations as follows: two consecutive layers of convolution and subsequent max pooling layers, three convolutional layers, and a flattening operation followed by three fully connected (FC) layers. All convolutional operations were followed by ReLU activation.

Image and kernel dimensions, pooling shapes, and stride lengths are depicted within the architecture figure.

In addition to AlexNet, the CNN proposed by Bojarski [9] was also tested. This architecture is referred to as NvidiaNet throughout the remainder of this work. The network topology is depicted in Figure **X** and is simpler in the number of trainable parameters. On a high level, the cropped image is normalized before being propagated through five Convolution+ReLU operations and three subsequent FC+ReLU and an affine output layer.

Regarding data collection, each sample displayed the view from the driver's perspective, and was collected at a frame rate of 5 Hz at a native resolution of 480x640. During training, the data was preprocessed (e.g. down sampling, cropping) to preserve system memory. The tracks used for data collection are shown in Fig. 4. The tracks represented with thicker lines contained glare from simulated sunlight to introduce noise.

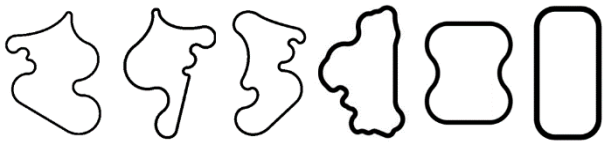

Fig. 4. Tracks used for training and testing the model.

Models were training using back-propagation and Adam optimization to determine parameters for each model. Once trained, the model was then tested on previously unseen tracks on TORCS, which has a library of over 50 tracks available for use.

B. *Front-End: TORCS Simulator*

TORCS is a race car simulation with built-in physics and programmable AI drivers. The simulator served as a test platform for our trained models and for data collection. We leverage memory sharing to directly send commands from our model to the TORCS robot.

C. *Runtime System: ROS*

A runtime system was setup to establish communication between different components. The Robot Operating System (ROS) [6] is an ideal framework to interface different components of software. ROS provides a publish-subscribe framework compatible with C++ and Python, addresses the problem of TORCS running on C++ and the TensorFlow scripts written in Python. ROS usage through the project is depicted in Fig. 5 and 6. A brief description of the various components is described below:

1) TORCS/ROS Bridge

This package [7] allowed TORCS to communicate with the runtime by using shared memory pointers, thereby enabling communication between ROS and TORCS.

2) OpenCV

This library [8] was used to interpret the ROS image message published via the bridge. OpenCV is designed for computational efficiency and real-time applications, making it useful for implementing our prediction model.

3) ROS Bag

The preferred method of storing ROS message and sensor data, bags are comparable to recordings of the outputs from the bridge. These were used to process data offline.

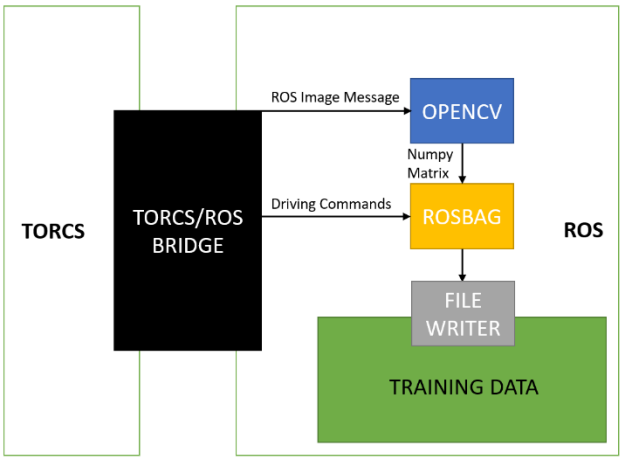

Fig. 5. Software architecture for training.

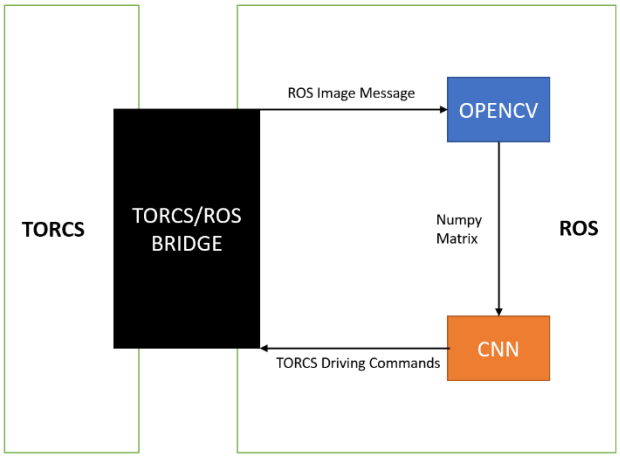

Fig. 6. Software architecture for testing.

D. *Robots*

Robots are simulated AI drivers running in TORCS. There were two main robots that were programmed - one for data collection and one for testing the model predictions.

The data collection robot utilized the *angle* sensor data output by TORCS, and a proportional controller was implemented to command the car to follow the track based on this sensor data. This was implemented purely to record the data, and sensor data apart from the camera was not used for the actual training and testing.

The robot used to simulate our model was directly interfaced with ROS and received it's driving commands (*steering, acceleration, brake*) from the TORCS bridge.

V. Results

Project results are split into sections according to the model training and optimization process.

A. *Project Results*

a) Parameter Sweeps:

The first important decision to be made was the selection of the network architecture. In particular, AlexNet and the modified NvidiaNet had to be compared to decide which network to proceed with. The preliminary step was to identify a set of optimal hyperparameters for each architecture before comparing the performance of these networks.

For the sweep, the models were tested on a range of hyperparameter values on a logarithmic scale for only a small number of iterations. Results of various combinations of batch sizes and learning rates are shown in Fig. 7. From these surfaces, the optimal values were extracted. Note that warmer colors represent higher loss values.

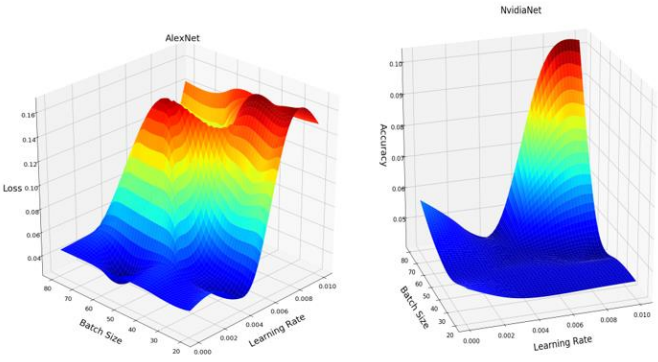

Fig. 7. Surface plots for the Hyperparameter sweeps. (Left - AlexNet, Right - NvidiaNet)

b) Architecture Selection:

From the hyperparameter sweep, optimal parameters for each architecture were selected and applied during training. Figure_ compares the training loss of the optimized AlexNet model and the optimized Nvidia model, calculated as mean square errors (MSE). The comparison in Fig. 8 shows that the Nvidia model was a better prediction model for our problem. The test loss is also plotted, and suggests that the current model still has problems generalizing.

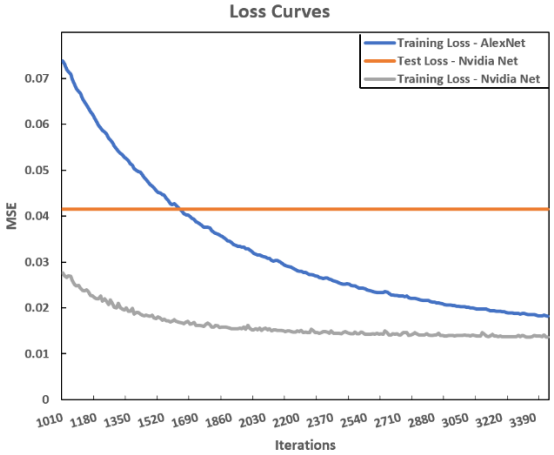

Fig. 8. Loss curve comparisons for both networks. Note that results are shown only for a subset of the total training iterations, and the test loss of NvidiaNet is plotted as a constant line for reference.

c) Feature Extraction:

With a better loss value, NvidiaNet was selected and trained on a dataset of images and labels collected by running the controller-driven robot. Details of the data are mentioned in Section. 4.1.

After training for approximately 30,000 iterations, the model was verified using a test set, and response maps from intermediate convolution layers were collected to verify the feature extraction capabilities of the network.

The response maps are shown in Fig. 9. Note that the last response map was computed on a real camera image.

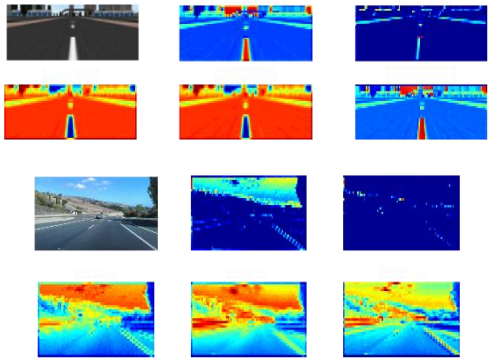

Fig. 9. Feature response maps of a TORCS test image (top) and a real test image (bottom).

d) Results:

The response maps suggest that the model can discern relevant driving features, such as the middle lane marking and the roadside. It was very accurately recognizing lane markings and edges of the track. However, some filters heavily weighed the surrounding landscape, which may have negatively affected some predictions.

It is interesting to note that the model performed poorly in generalizing autonomy to real-world images. This could be due to the model overfitting to the simulator environment. Concurrently, TORCS environment images are extremely low resolution compared to photos taken from a camera, and many objects in TORCS are not realistic. For generalizing real world images, it would be better to include real world data.

Fig. 10 shows some verification results for a range of test results. Note how the model has understood the nuances between turning left and right.

B. *Discussions of Insights Gained*

Results showed that the simpler NvidiaNet architecture was more suitable for autonomous driving around an open track compared to AlexNet, a more complex architecture. Intuitively this observation makes sense because open tracks have few features that must be recognized for proper decision making. Complex models such as AlexNet recognize these low-level features early in their layers, but may overanalyze the decision process at higher levels. Chen [1] also mentions the problems inherent in mapping high dimensional world representation,

which are present in complex architectures, to low dimension decisions, which is car steering.

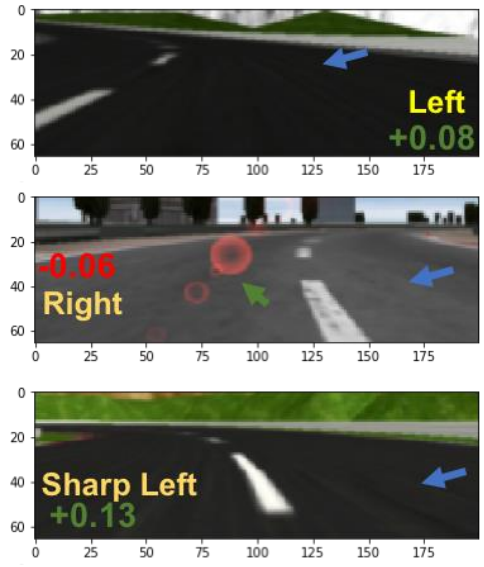

Fig. 10. Verification of testing results. Note that negative values refer to the steering wheel turned right.

Data collection and processing was also a major component to this project. The quality of the data collected and the method of processing the data to feed into the model made a significant difference in the training and testing accuracy. For example, the first data set that was used for training was taken directly from the Computer Vision and Robotics Lab website. This data had different lane markings and obstacles on the road, which were not present in the tracks that our car was tested on. Consequently, results were poor. The second data set was collected using our data collection software, but this time the raw data was used for training. While results improved compared to the Princeton data results, the car was still making poor decisions. We hypothesized that model was recognizing irrelevant features in the background and attempting to make decisions based on those features. Our final data set contained cropped images of the recorded data, where the road was the main focus and background features were reduced. Training on this data set produced the best results because the model could focus on road features, which are the important features for making driving decisions.

VI. Conclusion

We compare two network architectures and determined that the simpler NvidiaNet outperformed the more complex AlexNet in the regression task or predicting steering commands. The optimized NvidiaNet was then implemented on a TORCS robot, and demonstrated low-level steering decisions from image data.

From this project we learned the significance of data collection and preprocessing, and the importance of selecting the right model for a task. Even generally successful architectures, such as AlexNet, are not applicable for all situations, and simple models can be acceptable solutions.

Future work could include training an even simpler model compared to the NvidiaNet to reduce redundant features used for the prediction. Furthermore, it would be interesting to use a traditional pre-trained classifier appended with a newly trained FC layer, and observe that model's accuracy.

References

[1] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving", *2015 IEEE Int. Conf. Comput. Vis.*, pp. 2722-2730, 2015.

[2] A. Rait, L. Mohan, S. Selvaraj, "Deep Learning for Autonomous Cars", *unpublished*.

[3] A. Krizhevsky, I. Sutskever, G. Hinton, "ImageNet Classification with Deep Convolutional Networks", *Advances in Neural Information Processing Systems 25 (NIPS 2012)*.

[4] B. Wymann, E. Espie, C. Guionneau, C. Dimitrakakis, R. Coulom, A. Sumner, "TORCS, The Open Racing Car Simulator", *http://www.torcs.org*, 2014.

[5] C. Szegedy, W. Liu, Y. Jia, "Going deeper with convolutions", *2015 IEEE Int. Conf. Comput. Vis.*, pp. 1-9, 2015.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Ng, "ROS: an open-source Robot Operating System", *2009 ICRA Workshop on Open Source Software*.

[7] F. Mirus, "Repo for interfacing TORCS with ROS", *https://github.com/fmirus/torcs_ros*.

[8] Itseez, "Open Source Computer Vision Library", *https://github.com/itseez/opencv*.

[9] Bojarski, et al., "End to End Learning for Self-Driving Cars", *NVIDIA Corporation*, 2015.

[10] Zendel, et al., "Analyzing Computer Vision Data — The Good, the Bad and the Ugly", *2017 IEEE Int. Conf. Comput. Vis.* Pp 1063-1068, 2017.

Appendix

Individual Student Contributions

UNI	***bj2364***	***rpn2108***	***rs3226***
Last Name	**Jin**	**Nalam Venkat**	**Singh-Moon**
Contribution Fraction	⅓	⅓	⅓
Contribution	ROS Integration	ROS Integration	Network Design
	Data Collection	Network Design	Training and Testing
	Documentation	Documentation	Documentation