End-to-end Learning for Retrofitting Autonomous Vehicle Longitudinal Behavior

Anye Zhou
Georgia Tech
Atlanta, GA, USA
azhou46@gatech.edu

Hao Zhou
Georgia Tech
Atlanta, GA, USA
zhouOP-hao@gatech.edu

Cole Biafore
Georgia Tech
Atlanta, GA, USA
cbiafore3@gatech.edu

Abstract

Traditional radar-based adaptive cruise controllers are outdated, and more unfortunately most of them cannot navigate in low speed due to the limitation of the radar. The state of the art in AV industry is switching to camera-based solutions thanks to the recent development in computer vision and deep learning. To understand the behavior of existing AVs that use vision-based longitudinal motion planning, this study attempts three different methods to retrofit the AV longitudinal driving behavior (i.e., estimating the vehicle speed and acceleration given video input). After retrofitting, we can obtain a deep neural net model to describe how those black-box AVs will react to ambient traffic. We also present a custom longitudinal control model that shows the potential to implement different driving style or address any special needs on speed control.

1. Introduction

Autonomous vehicles are merging into our society as the technology continues to move forward. However, the impact of AVs on traffic is still not explicitly comprehended. Specifically, there is a lack of reliable and robust car-following model to describe the longitudinal behavior of AVs in the mixed-flow traffic (i.e., traffic consists of human-driven vehicles and AVs). Moreover, considering the issues of cost effectiveness, scalability, and the efforts of model tuning, the vision-based machine learning driving models are gaining more and more interests and popularity. For instance, Tesla launched the camera-based full self-driving functionality (FSD) which includes the traffic-aware cruise control. The leading AV start-up Comma.ai is also leveraging vision-based machine learning algorithms in their product Autopilot, which can be directly installed on specific vehicle models to achieve level-2 autonomous driving in most daily driving scenarios.

Therefore, in order to capture the intrinsic car-following dynamics of AVs that operate based on camera-based machine learning algorithms, we apply the techniques of end-to-end learning to retrofit the vehicle motion (i.e., estimate vehicle speed and acceleration) from the dashcam video input. Specifically, we use deep neural network to process the video input, and map the features in the images to the vehicle speed/acceleration.

1.1. Background and motivation

Among many model frameworks used in AVs, end-to-end learning provides a number of advantages compared to other designs. One such advantage is inherently it's approach to solve a problem domain using a single trainable neural network. This allows for a direct understanding as to how the problem domain has been approached, making both coding, training, and validating more intuitive. Note that this is because their is exactly one single set of inputs passed into a single neural network, which produces a single set of outputs. When unexpected behavior is encountered, the location of the failure is known and can be managed accordingly. However, when multiple models and data processing techniques are used to approach a problem domain, training the models becomes vague and difficult. It is also unclear as to whether a failure is caused by the network itself, or some alterations of the data between model evaluations.

This intuition introduces the motivation for this framework. If longitudinal behavior of an Autonomous Vehicle can be determined through an end-to-end learning model, more models should be evaluated and compared to provide researchers what are the essential neural network structures and layers that will benefit the performance of retrofitting.

1.2. Related works

Numerous studies have proposed the end-to-end model in fitting the vehicle driving command. For instance, NVIDIA [1, 2, 12] implemented convolutional neural network (CNN)-based deep neural nets to steer an AV to drive on a curve road. For the longitudinal movements, considering the spatial-temporal characteristics and the memory impact of vehicle longitudinal trajectories, the Long Short-term Memory (LSTM) or Gated Recurrent Unit (GRU) aug-

mented deep CNN [4, 13, 6] are applied to artificially forget or remember the historical frame features to improve the accuracy of vehicle longitudinal commands (i.e., speed, acceleration) prediction. Moreover, to improve the model estimation performance, the end-to-end learning approach can be enhanced with some extra features or side-line tasks. For example, [5], [14], [7], and [9] pooled the vehicle kinematic information (e.g., vehicle position, speed, and acceleration in previous time steps) with the video frames using concatenating layers to enhance the prediction of steering angle and acceleration.

Therefore, considering the trending use of end-to-end learning, we decide to leverage deep neural networks to retrofit the longitudinal model using open-source AV driving data. In this project, we will implement different neural network structures, and use different preprocessing techniques to comprehend the advantage of end-to-end learning, and how can we improve the model performance. The first model is a called the CNN-BGRU model which concatenates a deep CNN with a bidirectional gated recurrent unit, and a couple linear layers. The second model is called CNN-LSTM model with extra kinematic information. This model is based on concatenating CNN-RNN structure (similar to CNN-BGRU model), while it pools a series of vehicle speeds from previous time steps to augment the estimation accuracy. The last model is derived based on the Comma.ai Supercombo model, which has been implemented on the Comma.ai Openpilot to achieve vehicle trajectory planning. We remove the extra components not related to longitudinal motion planning (e.g., decision at intersections, lane change maneuvers), and then well-tune the model to cater for the needs of retrofitting longitudinal behaviors.

2. Methodology

This section introduces the deep neural network models that we apply in this project. The neural network structures, parameter settings, and the processing of data are specifically discussed.

2.1. CNN-BGRU model

The input of this end-to-end longitudinal driving model is a series of video frames obtained from the vehicle's front camera, and the output of it is the estimated speed/acceleration at the current time step.

2.1.1 Neural network structure

Figure 1 illustrates the structure and parameter setup of CNN-BGRU. This end-to-end driving neural network model applies the architecture of CNN, RNN, and fully connected layers where CNN is used for extracting useful features from the video frames, while the "TimeDistributed" layer will distribute the series of extracted features into the Bidirectional Gated Recuurent Unit (BGRU) layer. The BGRU layer will then capture the temporal dependency of those features to fit the speed and acceleration profiles. Note that [15] pointed out RNN can improve the accuracy of vehicle trajectory estimation due to the memory effects. Specifically, the deep CNN neural net is constructed referring to the NVIDIA model [1], which has been implemented in the real world to steer the vehicle driving in a curve road. After each CNN, we apply an activation layer to extract useful features. For the RNN layer, we apply BGRU to explore the correlation of data series both forward and backward in time to improve the estimation accuracy. Note that the specific parameter settings of all layers are included in Figure 1 as well.

2.1.2 Dataset description

The dataset we applied in the training process contained a 17 minute video and corresponding speed trajectory values extracted from the Comma 2K19 dataset [11], which is an autonomous driving dataset. The autonomous vehicle is driven by the Comma ai Openpilot, and the driving scenarios include free-flow traffic and congestion on highway and local roads.

2.1.3 Video frame processing

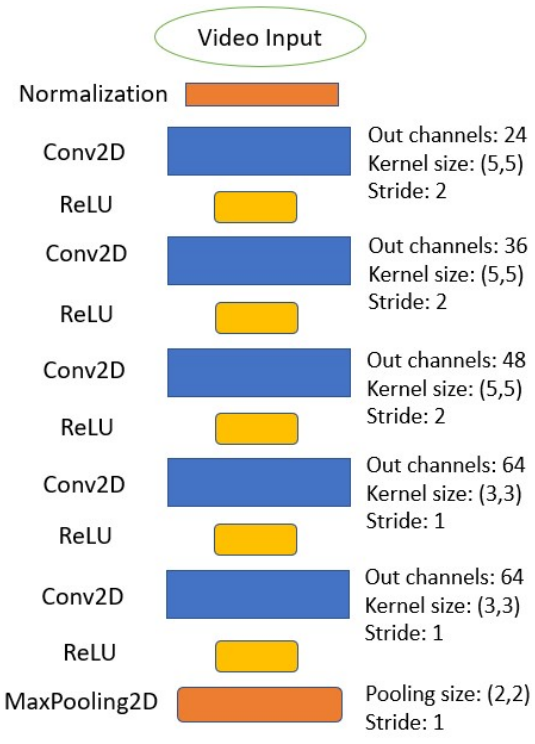
We performed some processing steps on the video input, such that the training results can be more robust if disturbances are confronted. The first operation is to crop the video frames to remove some residue features so that some perturbations are eliminated. Specifically, we want to focus on road signs, lanes, and ambient vehicles (these are essential observations for driving tasks on the highway). Thus, we remove the portions at the edges of video frames that include the vehicle dashboard, road shoulders, and sky. The second operation is to augment the video frames by adjusting the brightness, saturation, blurring, shifting and mirroring the video frames. We also transform the color base from RGB to YUV to enhance the performance.

2.1.4 Trajectory processing

The trajectory data can include noisy measurements, especially in the extracted acceleration profile. The noisy and choppy profile can contain faulty information, and even destabilize the training outcomes. Thus, we apply the Savitzky-Golay filter [10] to smooth the acceleration profile to improve the smoothness of the data.

2.2. CNN-LSTM Model

In this section, we attempt to build a end-to-end neural network capable of handling speed estimation based on the input of 4 consecutive video frames and the vehicle speed of



(a) CNNs

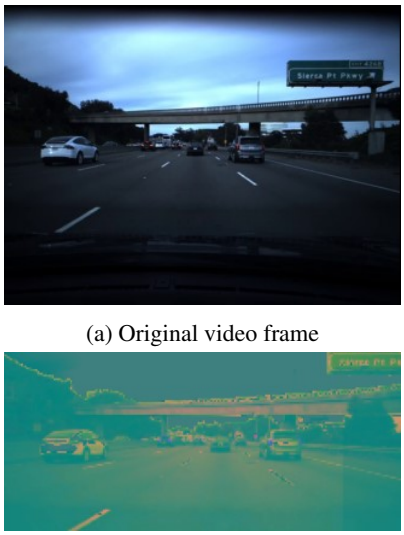
(b) RNN and perceptrons

Figure 1: CNN-BGRU

those respective frames. This network can be used to simulate speed prediction given a front facing camera sensor from a car and a velocity reading from a speedometer.

2.2.1 Neural network structure

The architecture for this model, as shown in figure 3 consists of a variety of convolutional layers applied to each of the 4 input images to attempt to determine any key features within the images. This output is then flattened and used as inputs to an LSTM layer, which will look for time-specific relationships between each frame. analyzing the convolutional output in this manner will allow the network to determine features such as blurs or changes in the position of objects, such as a car or sign, and make an estimation of the AVs speed. Note that this output is then concatenated with the speed of the car from the 4 previous frames and connected to a series of fully connected layers. These layers are structured in such a way that the final output from the network is an estimation of the AVs speed.



(a) Original video frame

(b) Processed video frame

Figure 2: Comparison of video frames

One notable feature of this model is the simplicity of its design, yet ability to produce reasonably accurate results. The reason this is significant is because the network itself is able to be trained on a modern day personal computer in only a couple hours. Therefore, given the capabilities of this network on a personal computer, the training time required on a professional computer built for deep learning would be significantly less. This would allow the a programmer to train the network with more epochs and produce more accurate results given that measures are taken to prevent over-fitting.

2.2.2 Training and Validation

During training of the CNN-LSTM model, a mean square error function was used to calculate the total loss. During each epoch, the total loss noticeably decreased from the previous epoch, resulting in a final training total loss of 7.558. Note that this loss function was calculated with dropout layers included. This is significant because we expect these layers to increase total training loss, but also help prevent over-fitting. The final validation loss calculated for the CNN-LSTM model was 0.0827.

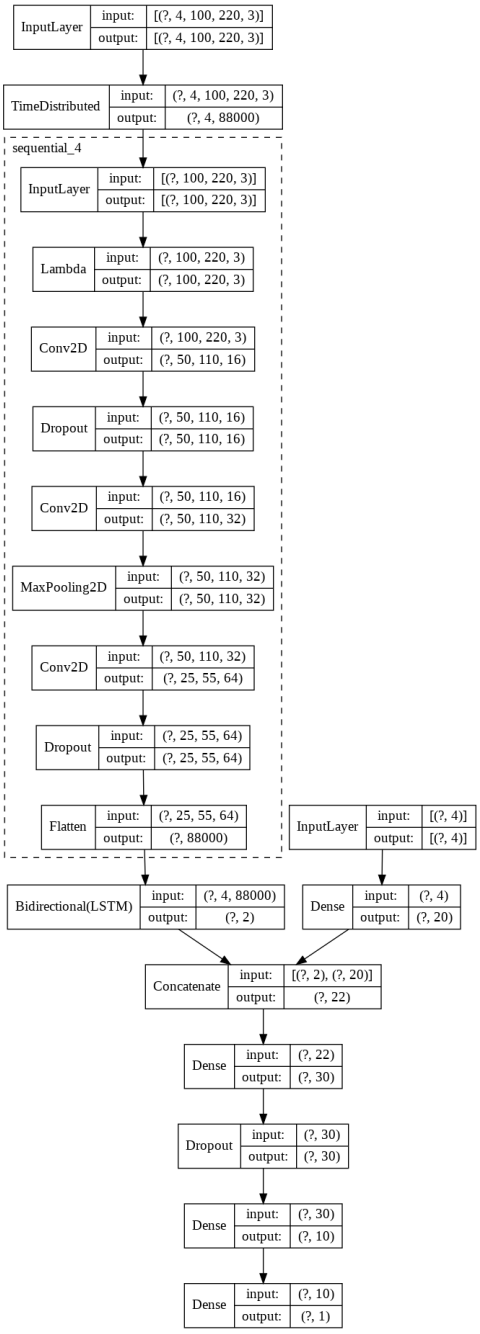


Figure 3: CNN-LSTM Model for speed estimation

2.3. OP-hao Model

Learning to predict is the backbone of a self-driving model. From a traffic perspective, a more accurate prediction on the future speed might help improve both safety and traffic efficiency, either for the individual vehicle or on the system level.

In this paper we attempt to train a custom end-to-end longitudinal control model based on the pretrained OP model and our own driving data. Note that to enhance the capability of navigating in heavy congestion in areas like Atlanta, we intentionally choose a great amount of congestion data with stop-and-go traffic.

2.3.1 Neural network structure

We use a pre-trained model from Openpilot(OP) that is open-sourced by Comma.ai. Since we only focus on the longitudinal prediction, we can safely ignore the outputs such as path, lead, etc. Note a partial neural network structure of the Openpilot network can be observed in figure 4.

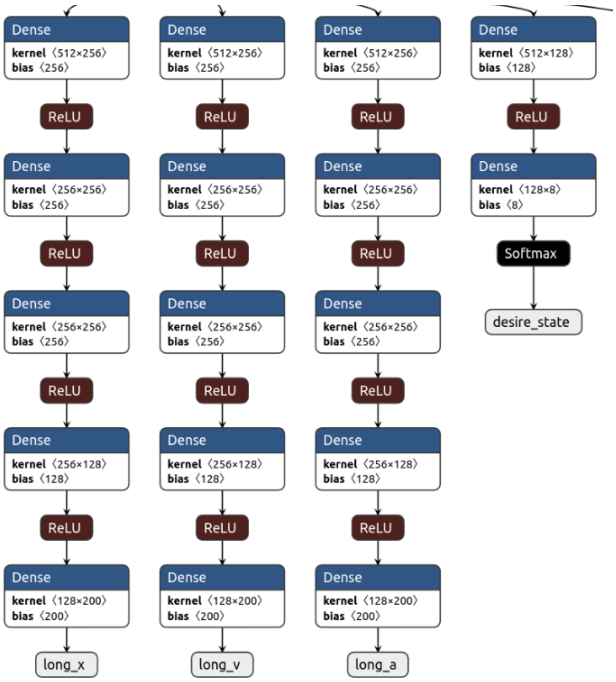


Figure 4: Part of the neural network architecture in the OP model

2.3.2 Data

To train a custom model, we hit on the road and collected some driving data from I-85/I-285 in Atlanta, which in total lasts about 1-hour. All the miles were driven by the author without any assistance, such as Adaptive Cruise Control and Lane Keep Assist Systems. The raw data are stored as individual 1-min driving footages which were recorded by a phone camera mounted on the windshild of a 2019 Honda Civic. Menawhile we also used a CAN-reader to log the ground-truth speed of the vehicle while driving.

To get a more sensible driving policy, we train the model using an evenly distributed dataset, where the driving data

points come from different speed levels ranging from $0 \sim 5, 5 \sim 10...25 \sim 30m/s$. This is to prevent our custom model from being biased to any low-speed or high-speed scenarios.

2.3.3 Training

Since we are focusing merely on the prediction of future speeds, it is reasonable to freeze all other layers in the pre-trained OP model except the three last linear layers that directly outputs the long_v in Fig.4. Note that long_v is a vector of 200 elements which correspond to the speed values in the next 2 seconds. As we notice the speed prediction between $1 \sim 2$ is usually inaccurate, we only chose the future 1 second for simplicity.

The loss of model prediction at time t,l_t, is defined as the discrepancy between the prediction v_{pred} and the ground-truth speed v_{truth}.

$$l_t = \Sigma_{i=1}^{10} |v_{pred}(t+i) - v_{truth}(t+i)| \quad (1)$$

3. Experiments

This section introduces the dataset we used for training and a discussion of the results. Three models are all implemented using Tensorflow Keras [3].

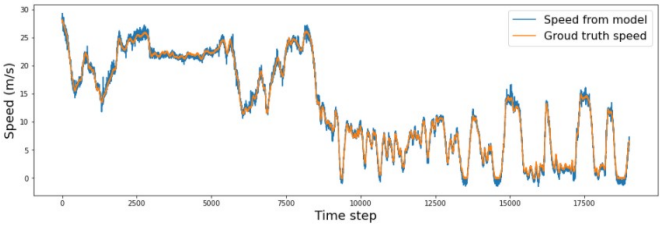
3.1. Training and results of CNN-BGRU

The hyper-parameter settings of CNN-BGRU training is listed in Table 1. We select these values based on trial and error, and the values utilized in references. The optimizer applied for CNN-BGRU is Adam [8], the learning decays by a factor of 0.5 after every 10 epochs. The loss function we apply in the training process is the mean square error (MSE) between the estimated value and ground-truth value, which is a common measure in the regression curve-fitting. Note that to prevent overfitting, we include 'Dropout' layer after the third and forth Conv2D layers, BGRU layer, and the first Dense layer, with dropout rate of 0.5.

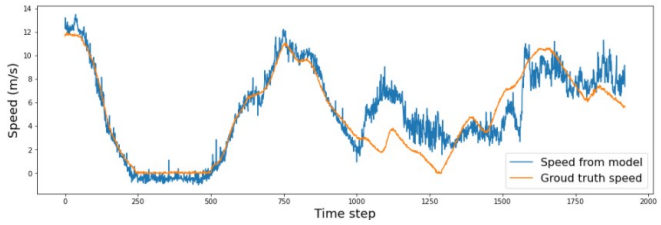
Parameter	Value
Batch size	32
Time step	4
Initial learning rate	0.0001
Epochs	30

Table 1: Hyper-parameters for CNN-BGRU

After the training for the speed estimation, the training loss is 3.3644, while the validation loss is 3.9350; for the acceleration estimation, the training loss is 0.0756, the validation loss is 0.2112. The speed estimation performance of CNN-BGRU is illustrated through in Figure 5. We can

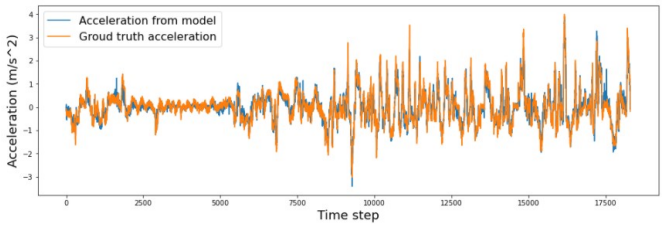


(a) Training dataset

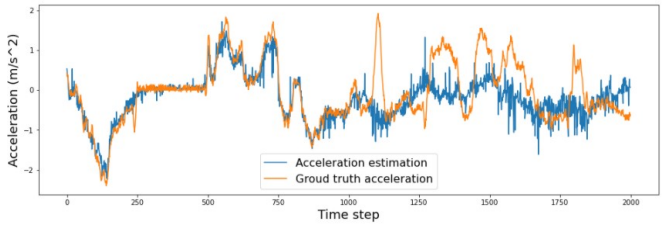


(b) Testing dataset

Figure 5: Performance of speed estimation



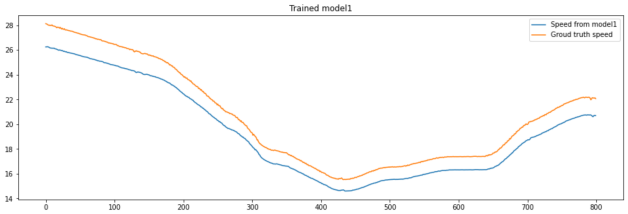
(a) Training dataset



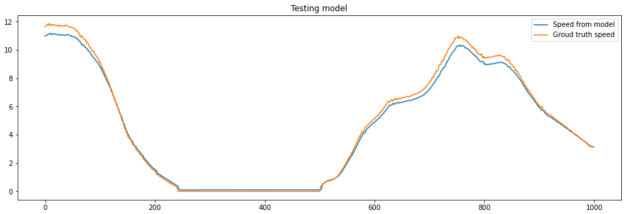
(b) Testing dataset

Figure 6: Performance of acceleration estimation

notice that for the training data, the estimated speed profile (blue line) keeps up with the ground-truth speed (orange line), although the estimated speed is a little more choppy. However, for the testing data, we can notice that the estimated speed starts to drift away from the ground-truth speed, which means the model fails to generalize. Our explanation is as follows: in the training data, the vehicle is driving in the highway scenario, while in the testing data, the vehicle runs into some urban area and local roads, which is much different from the situation of highway driving, thus, the model fails to handle this type of new scenario and generates inaccurate speed estimation. To alleviate this, we need to enrich our dataset to include some urban-driving



(a) CNN-LSTM model output on training data



(b) CNN-LSTM model output on test data

Figure 7: CNN-LSTM model results

scenario and re-train the CNN-BGRU model with more data to help it generalize better. Similar circumstances occur for the acceleration estimation (as visulized in 6, the CNN-BGRU model can capture the variations of choppy vehicle acceleration in the training data which mainly involves highway driving scenario, while fails to estimation accurate acceleration in the testing dataset that consists of urban-driving scenario.

3.2. Results of the CNN-LSTM Model

After 3 epochs of training, a learning rate of 1.05×10^{-4}, and no decay in the learning rate, the CNN-LSTM model was able to accurately train 762247 parameters to predict the speed of an AV at a given time. This accuracy was so precise that the final mean square loss on the test data in ifgure 7 was 0.0973 for 1000 test cases, meaning that the structure of the neural network was sufficient in determining the speed of the AV. It is, however, important to note this networks use of a second input providing the vehicle's speed in the previous image frames. Based on the nature of this input, it is reasonable to say that it could have a very high correlation with the output, meaning that it could be dominating the output of the network.

To consider this claim, a second network was created with the same structure as the CNN-LSTM model in figure 3, but it did not take in this second input. Therefore, the only input to this network was the 4 images passed into the convolutional layers. The results of this network would give an idea as to how much the previous velocity values were influencing the output of the CNN-LSTM model. Unfortunately, the results of this network was significant, with a total loss of 17.185 over 1000 test cases. This would seem to suggest that the previous vehicle's speeds played a significant role in the output for the CNN-LSTM model, and that an end-to-end neural network that relied solely on image frames to determine the longitudinal velocity of a car would require either a more complicated network fine-tuned to discover more key features in the images or more time to train with more epochs.

3.3. Results of the OP-hao model

First we show the training process of our customized OP-hao model, where the loss histories for both training and validation were shown in the Fig.8

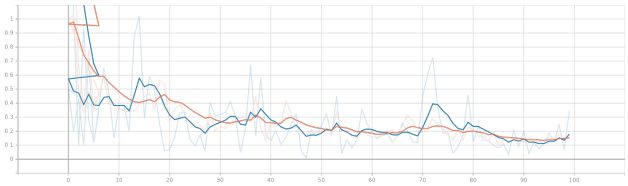


Figure 8: Loss history in 100 epochs of the training and validation process: orange line corresponds to the training loss and blue curve represents the validataion loss. The graph was automatically generated by tensorboard

To see how the OP-hao model outperforms the OP model, we can compare their performance using the following examples:

First, we investigate the full-stop scenario where the vehicles are waiting the downstream congestion to dissipate. The OP model cannot tell the difference and will always predict an increasing pattern as the future speed.

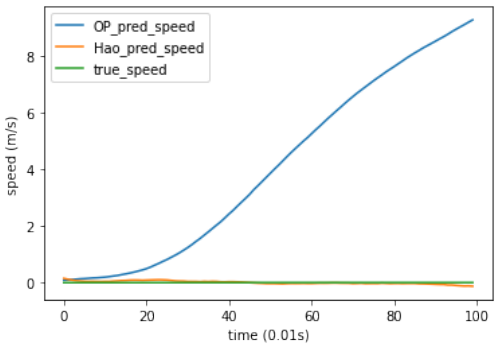


Figure 9: Comparison of the OP model and OP-hao model on predicting future speed in stop and go traffic

For the second case, we show a common following scenario in stop-and-go traffic, where vehicles typically crawl with very low speeds. As Figure 10 indicates, the OP model mistakenly predicts an increasing pattern while the actually speed declines. The OP-hao model is able to predict

a more faithful future trajectory, but we can see an almost constant discrepancy between the tow. Unfortunately the current speed seems to deviate after the training.

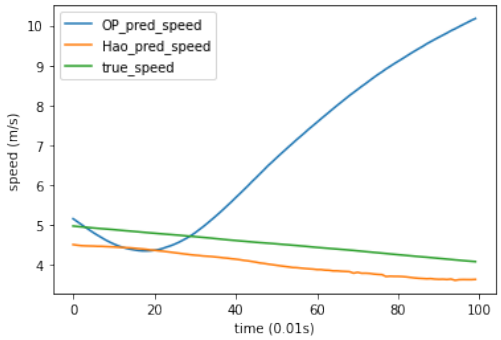


Figure 10: Comparison of the OP model and OP-hao model on predicting future speed in stop and go traffic

4. Discussion and future work

This project presents three deep neural network models for retrofitting AV longitudinal driving behavior. The CNN-BGRU model demonstrates that the combination CNN and RNN is an effective structure for retrofitting the longitudinal behavior. Meanwhile, the CNN-LSTM model showcases that including extra kinematic information from previous time step can help boost the model performance, and the training effort (e.g., less epochs) is reduced compared to the CNN-BGRU model. Moreover, we train a custom model OP-hao using our own driving data. The model elaborates that the besides retrofitting, the prediction of the speed and acceleration is another promising functionality the model can be extended.

Future directions include the following: (i) using more driving data for model training to improve the generalization; (ii) applying more advanced image processing techniques (e.g., semantic segmentation) to exclude the disturbance in the environment; and (iii) using the extracted features (e.g., spacing and speed of ambient vehicles) to retrofit the longitudinal behavior, comparing this method to end-to-end model.

References

[1] M. Bojarski, D. Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, L. Jackel, Mathew Monfort, U. Muller, Jiakai Zhang, X. Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *ArXiv*, abs/1604.07316, 2016.

[2] Zhilu Chen and Xinming Huang. End-To-end learning for lane keeping of self-driving cars. *IEEE Intelligent Vehicles Symposium, Proceedings*, (Iv):1856–1860, 2017.

[3] François Chollet. Keras: The python deep learning library. 2018.

[4] Hesham M. Eraqi, Mohamed N. Moustafa, and Jens Honer. End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies. (Nips):1–8, 2017.

[5] Laurent George, Thibault Buhet, Emilie Wirbel, Gaetan Le-Gall, and Xavier Perrotton. Imitation learning for end to end vehicle longitudinal control with forward camera. *arXiv preprint arXiv:1812.05841*, 2018.

[6] Simon Hecker, Dengxin Dai, and Luc Van Gool. End-to-end learning of driving models with surround-view cameras and route planners. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 435–453, 2018.

[7] Tsung Ming Hsu, Cheng Hsien Wang, and Yu Rui Chen. End-to-end deep learning for autonomous longitudinal and lateral control based on vehicle dynamics. *ACM International Conference Proceeding Series*, pages 111–114, 2018.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[9] Zhicheng Li, Zhihao Gu, Xuan Di, and Rongye Shi. An lstm-based autonomous driving model using waymo open dataset. *ArXiv*, abs/2002.05878, 2020.

[10] A. Savitzky and M. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36:1627–1639, 1964.

[11] Harald Schafer, Eder Santana, Andrew Haden, and Riccardo Biasini. A commute in data: The comma2k19 dataset, 2018.

[12] Shobit Sharma, Girma Tewolde, and Jaerock Kwon. Lateral and longitudinal motion control of autonomous vehicles using deep learning. In *2019 IEEE International Conference on Electro Information Technology (EIT)*, pages 1–5. IEEE, 2019.

[13] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017.

[14] Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end Multi-Modal Multi-Task Vehicle Control for Self-Driving Cars with Visual Perceptions. *Proceedings - International Conference on Pattern Recognition*, 2018-August:2289–2294, 2018.

[15] Mofan Zhou, Xiaobo Qu, and Xiaopeng Li. A recurrent neural network based microscopic car following model to predict traffic oscillation. *Transportation Research Part C: Emerging Technologies*, 84:245–264, 2017.