CNN-LSTM Based Learning for Self-Driving Cars

Nazmus Sakib
University of Alberta
Edmonton, Canada
nazmus@ualberta.ca

Nilanjan Ray
University of Alberta
Edmonton, Canada
nray1@ualberta.ca

Abstract

End to end approach of learning for self-driving car is showing immense potential due to superior performance of some deep models. We constructed a model with Convolutional neural network (CNN) and Long short-term memory(LSTM) which takes raw color images and outputs steering commands. Whenever we drive in highway, our steering motions do not variate much, and next motion command follows the previous ones. Inspired by this basic intuition, we argue that the CNN model followed by a recurrent neural network namely LSTM will be able to learn this behavior of driving. Our model implemented on Keras and tested in Udacity opensource simulator runs real-time with any standard Nvidia-GPU.

1. Introduction

Self-driving car is going to be the only large-scale consumer robot product in upcoming decades. This industry is expanding at a high acceleration compared to other robots such as social robots or assistance robot. The current car industry has a huge target people who seek for newer feature for newer models of the cars. Since driving is a risky task, it requires to be highly reliable and massively reactive agent. The problem of self-driving car has always drawn the attention of the researchers specially from the vision and robotics enthusiasts. Recently Nvidia has released their idea of driving the car in an end to end approach [1], which has drawn massive attention to the community. It is an interesting idea and very much different than other popular method such as SLAM (Simultaneous Localization and Mapping) [2] which is a modular based approach of acquiring sensor data, filtering, recognition and control.

The core idea of Nvidia's work is mostly inspired by the recent outbreak of using Deep Learning in solving multivariate problems. Availability of GPU (Graphics Processing Unit), labeled dataset and different open source libraries such tensorflow, keras, caffe [3] etc. have popularized designing CNN models and its different kinds. With the deployment of theses deep models researchers are already outperforming the previous baselines. The availability of large scale labeled data such as Large Scale Visual Recognition Challenge (ILSVRC) [4] and powerful variations of CNN, e.g. VGG [5], Resnet [6], LSTM [7] have successfully solved many complex problems in an end to end manner.

In this project we mostly follow the Nvidia's strategy of level-3 driving [8] on highways except their CNN model. The deep model is trained on RGB images taken from front side of the car and corresponding steering wheel is recorded. Then these training data acquired from driving is feed into the CNN and trained as a regression problem which after training can predict the steering command when given a single image. The method does not explicitly define the hand-crafted feature extraction e.g. road detection, lane detection, marking drivable free space etc.

Our motivation is taken from general intuition of driving. When controlling the steering we do not change it suddenly in a discrete manner unless the case of emergency. Our driving sequence is followed by previous steering commands. Therefore, we train our model with a CNN, followed by an LSTM in an end to end fashion. We argue that our CNN models learn to extract features with different kernels, and the fully connected layers map the extracted feature to control the vehicle. The last single neuron of the CNN outputting the control is followed by LSTM neurons which learn the sequence of driving over a significant time frame. Our model is capable of producing steering commands at 25 fps on a Nvidia 1080 GPU and outperforms Nvidia's method in many aspects.

2. Related Works

The end to end approach for self-driving is a comparatively newer concept. Although the earlier work (1989) of ALVINN (Autonomous Land Vehicle in a neural network) includes using a fully connected network to drive. It used the pixels values mapped to actuation. It was able to drive despite few obstacles. Although the network is simpler but carried huge significance for the later modern works. The online course company Udacity have partnered with Nvidia and released dataset for a self-

driving Challenge. The work of Stanford students [9] for the competition is remarkable. They were also inspired by Nvidia's work and replaced the model with 3D CNN, LSTM and ResNet. Their performance in the competition is noteworthy. They used the 3D convolution followed by a Recurrent network to include the temporal information of the video/sampled images. Their assumption is the temporal difference produce a better control although there is no proof of the fact other than the performance. They also used a pretrained ResNet50, added few layers of fully connected at the end for training. The model was pretrained on ImageNet. They evaluated their performance with RMSE between the actual and predicted steering values after partitioning the dataset into training and validation set from the Udacity released data.

The work of Huazhe Xu et.al [10] from UCB addresses the driving problem as a language model where the driving behavior is highly likely of the previously observed ones. They used a FCN-LSTM for the multimodal predictions. The loss function for training was combination of driving loss and segmentation loss. Although the model is capable of running with a RGB image but during learning process they train with the mentioned losses to make it more robust. They named it as privileged learning, but how the segmentation data was labeled is not clear from their discussion.

The combination of Generative Adversarial Network (GAN) [11] and Variational Auto-encoder (VAE) [12] for video prediction by Comma.ai [13] is also an interesting concept. Predicating realistic video frames ahead largely helps control specially in highway which is similar to the interpolating lane marking ahead based on current observation.

3. Network Architecture

The CNN part of our proposed CNN-LSTM model is same as the Nvidia [1] except that we excluded the layer having 1164 fully connected neurons. Since the model successfully extracts feature and produce steering commands. We added LSTM layers at the end which takes input from single final output of the CNN thus forcing the LSTM layers to learn only steering command sequences. We assume that we can draw analogy of our design with proportional P and differential gain D from the concept of PID [14] control theory. Our argument for the analogy is the CNN part is acting like the P part which is directly mapping the error to correct steering commands and the LSTM is using the previously observed images to act like D component of the system not to react much. This phenomenon creates a smoothing driving effect on our dataset which will be elaborated in the result section below.The model shown in Figure 2, takes 3 channel RGB images of size 66x200 and then it is normalized. Then

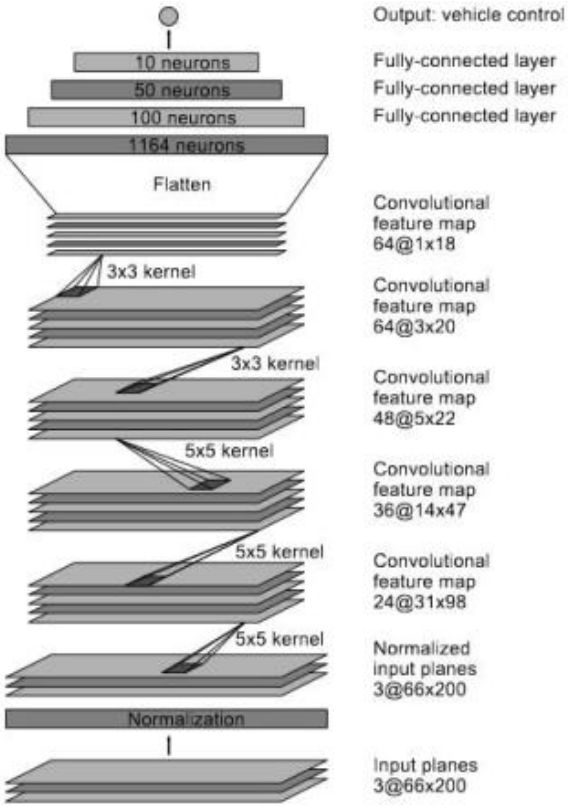

Figure 1: Nvidia's CNN architecture for end to end learning.

there is 5 convolutional layers and 2 fully connected layers (Nvidia has 3 fully connected layers).24 kernels of 5x5 is applied on the 1st layer, 36 kernels of 5x5 on 2nd layer and 48 kernels of 5x5 on the 3rd layer. All of them have 'ELU' activation and 2x2 stride. The last 2 layers of the CNN has 64, 64 kernels of each of size 3x3 respectively. These two layers do not have any activation function or strides. Then the outputs are flattened and connected to a layer of 50 fully connected neurons(FCN), followed by 10 FCN in the next layer. The final output of the CNN comes from a single neuron which learns to predict the steering control.

The single neuron output is then fed into LSTM layer in a time distributed manner over 5 frames. The 1st LSTM layer has 32 LSTM neurons and 2nd layer has 16 LSTM neurons which yield to a single steering command output. Here we can see the model looks like the ancient sand clock called Hourglass. Where we designed it to narrow to a single neuron and let the LSTM learn only the steering command output from multiple time frame.

The model has 197,024 parameters whereas Nvidia has 252,219 which is due to absence on a FCN layer of 1164 neuron. Dropping the layer did not affect the performance in our dataset considering both the environment being trained for same number of epochs. The regression is run on the steering commands of our dataset produced from the Udacity open source simulator [15].

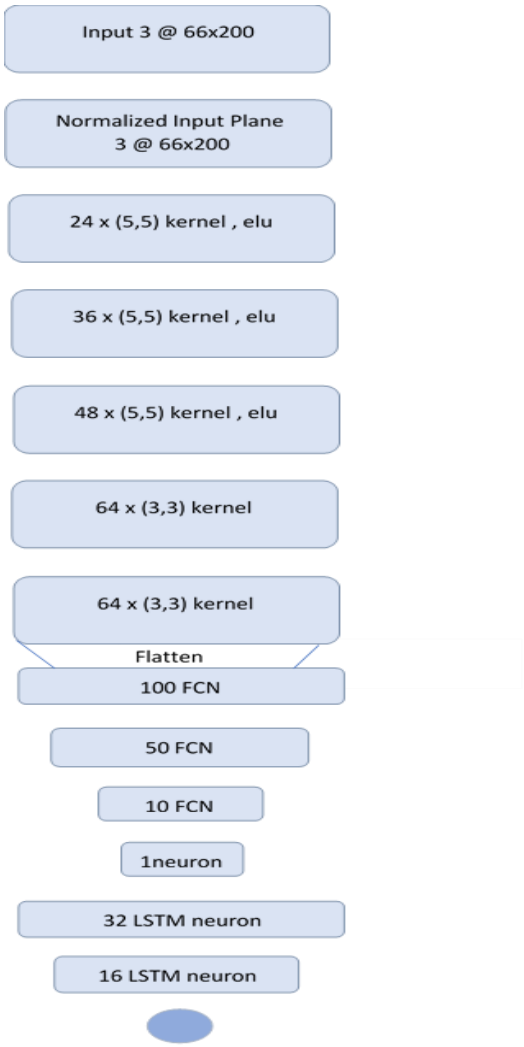

Figure 2: Proposed CNN-LSTM architecture

4. Dataset Collection and Properties

The training data was collected from Udacity open source simulator. The simulator has two modes training and autonomous test. When training is selected it let us drive using the keyboard or any joystick. It records 3 images the same way Nvidia captures from the car from its three ends. The corresponding keystroke is saved in terms of steering angle i.e. if the key stroke is pressed for longer times generate larger value of steering angle. The throttle, reverse and speed are also calculated. The simulator samples the video at 10 FPS.

It has been found that running the car following the lane or centering it on road does not help the model to learn to recover from off the lane conditions. So careful data set was created having instances of all type of forward driving. The simulator allows us to pause. So, the car position was changed several times to record good recovering instances. There are 2 tracks one is almost planner and the other one is hilly track we named them track 1 and track 2 , (Figure 3) respectively.

The track does not have any other car, pedestrian, signals etc. Hence it is a very simplified one. It can be finished by driving only for 2 to 3 minutes. We drove the car for more than 3 minutes in both the directions to avoid biasness on one type of turns. The 3 types of images took around 56MB of sizes which were randomly shifted and flipped to create as much as 3 times the dataset as described in the Nvidia's work. It helped to increase the data size to make sufficient for any generalized deep model. As the steering angle is relevant only for the center image, a constant offset was added and subtracted for right and left images respectively. Before feeding to the network some preprocessing were done specially to crop out the unnecessary image information such as sky or hills beyond the track and to match the input size 66x200 for the model.

5. Experiments

The model was trained on the mean squared error (MSE) loss function between the recorded steering commands and the ones generated from the model as output at the end of LSTM layers. Dropout probability of 0.1 was used for each of the LSTM layers to make the model more generalized. Both the Nvidia and our proposed CNN-LSTM model were implemented in Keras. Both models were trained for similar number of epochs. For CNN LSTM each epoch took around 1000 secs on a single TI 1080 and around 600 secs on 2 TI 1080 GPUs.

The Adam optimizer was used with the learning rate of 0.0001, batch size of 40. 20 epochs took around 6 hours.

Figure 3: Challenging Hilly Track 2, Udacity simulator

6. Evaluation and Discussion

The evaluation part of these of system is tricky. Most of the evaluations in others work are done with RMSE. We divided the 2 tracks into subset of T1, T2 and T3 for different segments of the road. For the evaluation we drove

the car carefully on the middle of the road. Then the evaluation set was fed to both the network to predict the steering angles and were compared with the original ones. We observe the following RMSE results in Table 1.

Track	Nvidia CNN	CNN-LSTM
T1	0.71212	**0.29445**
T2	1.44381	**0.22438**
T3	1.44486	**0.22519**

It reveals that our method performed well on the evaluation set after training on the same data. This performance can also be observed visually on the video [16]. Its clear that Nvidia drive model was shaking a lot on the road whereas our model looked very firm and less oscillating on the road. Although Nvidia model runs faster than our model. The paper states that it runs at 30 fps while our model runs on 25 fps on a single 1080 TI GPU. From the video it is clear that their model runs faster than ours for the same track.

On the other hand, when both the models were driven on the challenging track Figure 3, the Nvidia's model performs better than ours. In fact, out of 10 trials it was able to finish 5 times whereas our method struggles to finish only 3 times. It is evident that the CNN based model is very responsive and it mimics the training dataset a lot. Whereas, CNN-LSTM drive is smooth and seems more reliable.

The evaluation metrics used here is not fair in the sense that both the models were run for same number of epochs. Different model might get better at learning at different higher number of epochs. It might be the reason why the original paper did a simplified autonomy metrics rather than discussing the training iterations. If we measure in terms of autonomy discussed in [1], on track 1, our model will have higher autonomy, but it will be lower on the other track compared to the CNN model.

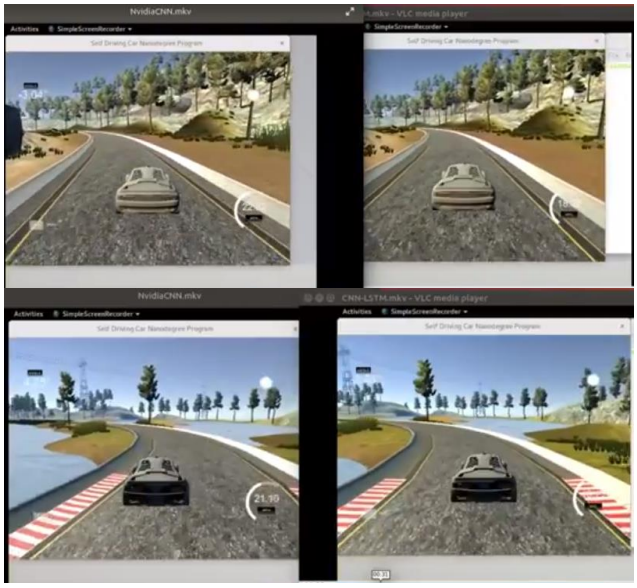

Figure 4: Driving comparison on the same track.

From the visual inference of the evaluation video we can conclude that our design approach to include time distributed steering commands is kind of working. Because, the system is not much reactive and behaving very stable, resulting in a better driving than the previous model.

7. Conclusion

The end to end driving is a newer concept but very much promising. It is easy to deploy. It can work without any depth sensor values. We can interpret this approach as converting hours of driving data compressed to a single model. This type of model gets rid of a lot of hyper-parameter that are found on the state of art methods. Tuning the threshold for different modular makes the system more complicated. Here comes the superiority and strength of the end to end deep learning-based approach. They can exploit the availability of recent large scale labeled data. There are some apps which can record the driving data from crowd-sourcing. Access to those diverse real-life driving can make the models more robust and reliable.

In future, a strong evaluation procedure has to be undertaken. On possible way could be using the steering command and velocity and derive a motion model to find the trajectory of the car. Because in driving, there are many possible steering options but the trajectory for safe and comfortable driving will mostly align with the road pattern. Next, the models can be trained and tested on multiple dataset. They should be trained for longer epochs to retrieve the change of performance.

It is very evident that driving it self is a complex problem and, in the end to end approach we are doing regression on a one particular control factor. There are other factors such as obstacle avoidance, car or pedestrian detection, switching lane, reacting during the time of emergency. In order to achieve an overall performance, the end to end model has to be violated. Instead, it has to be a hybrid system. Use of more advanced concepts such as first or second order derivative at the loss function might help to reach multiple goal for the driving agent.

8. Acknowledgement

Thanks to Prof. Nilanjan for the course 617 and the supplied useful materials including papers, video tutorials and notes, helped the project finished within short period of time. Thanks to Prof. Hong Zhang for GPU access in the Robotics-Vision Lab, UAlberta, which accelerated the project progress. Thanks to Martin Humphrey for

configuring the necessary alternate libraries on Notecc OS in the lab machines to run the project source codes.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] Stachniss C., Leonard J.J., Thrun S. (2016) Simultaneous Localization and Mapping. In: Siciliano B., Khatib O. (eds) Springer Handbook of Robotics. Springer, Cham.

[3] Tensorflow, Keras, Caffe. https://www.pyimagesearch.com/2016/06/27/my-top-9-favorite-python-deep-learning-libraries/

[4] Large scale visual recognition challenge (ILSVRC). URL: http://www.image-net.org/challenges/LSVRC/.

[5] Karen Simonyan, Andrew Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV].

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV].

[7] Klaus Greff, Rupesh K. Srivastava, Jan Koutn´ık, Bas R. Steunebrink, Jurgen Schmidhuber. LSTM: A Search Space Odyssey.

[8] Driving level. https://en.wikipedia.org/wiki/Autonomous_car .

[9] Stanford Report, Self-Driving Car Steering Angle Prediction Based on Image Recognition. http://cs231n.stanford.edu/reports/2017/pdfs/626.pdf

[10] Huazhe Xu, Yang Gao, Fisher Yu, Trevor Darrell, End-to-end learning of Driving Models from Large-scale Video dataset. arXiv:1612.01079 [cs.CV]

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014

[12] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013

[13] E. Santana and G. Hotz. Learning a driving simulator. *preprint arXiv:1608.01230*, 2016

[14] PID controller. https://en.wikipedia.org/wiki/PID_controller

[15] Udacity Simulator. https://github.com/udacity/self-driving-car-sim

[16] Evaluation video. https://www.youtube.com/watch?v=iOIdY-R79GM