CS230

Deep learning approaches for the Lyft Motion Prediction for Autonomous Vehicles challenge

Augustine Chemparathy
Department of Management Science & Engineering
Stanford University
agchempa@stanford.edu

Hugo Vergnes
Department of Statistics
Stanford University
hugo.vergnes@stanford.edu

Abstract

Our project predicts the trajectory of "agents," including pedestrians, bikes, and cars, in the vicinity of a moving self-driving vehicle, termed the "ego". This project is part of an entry in the Kaggle Competition "Lyft Motion Prediction for Autonomous vehicles." We are provided a 20GB dataset of over 40 million frames of motion of Lyft self-driving vehicles and surrounding agents. We are also provided a rasterizer to create series of bird's-eye images of these recorded scenes. To this day, we have tried different approaches in PyTorch[4]: (1) generating images with the rasterizer to feed into a ResNet, (2) using a sequence model to forecast trajectories, and (3) using geometric approaches with PointNet and GNN. In this final report, we will describe each approach as well as the results we obtained throughout our project.

1 Introduction

Automated vehicles are the ultimate goal of a race between major tech companies (Uber, Lyft, Tesla etc). In the last decade, deep learning has enabled major breakthroughs in this industry, but at the state of the art the performances are still not sufficient to allow productions of AV's. To increase performance and reduce the risk of bad maneuvering, the AV needs to be able to predict the motion of surrounding agents just as a human operator would. In this project, we are tasked with predicting the trajectory of a given agent over a sequence future frames given historical data.

2 Dataset and Features

The Kaggle competition provides a 21GB dataset consisting of over 16,000 scenes recorded from a Lyft vehicle, in which each scene consists of several hundred frames [2]. In total, we are provided over 21 million frames for each of the train and the validation dataset. The data is previously split into train, validation and test sets by the host. The test set has hidden ground-truth values and is used to assess submissions to the Kaggle competition. The dataset is available in a compressed format in .zarr which allow for the same operations as numpy. The competition host also provides a Python library (L5Kit) to help load, train on, and evaluate on the data. To describe each scene, we have 15 different features. The most important includes frame-by-frame data such as the centroid coordinates and yaw of ego and the agents, as well as metadata describing each agent such as probability distribution of agent types (e.g. pedestrian, car, bicycle). A rasterizer is provided to generate bird's-eye view images of the scenes. An example of such an image is provided in Figure 1.

CS230: Deep Learning, Fall 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

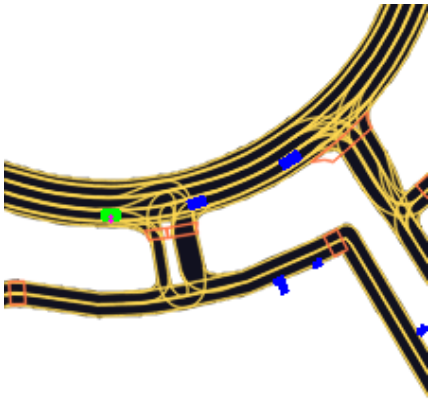

Figure 1: Example of the output of the rasterizer

3 Methods

We describe below the deep learning approaches we have tested for this problem. We initially used average MSE loss between the predicted centroids and ground-truth centroids. After our Milestone report, we moved all models to output multi-mode predictions, in which k hypothesized trajectories and k associated confidences (summing to 1) are reported. The Kaggle competition suggests using multi-mode prediction because it better captures the uncertainty in agents' movements. We typically predicted 3 possible trajectories.

The following loss function, provided by Lyft, is used for multi-mode prediction, where c_k is the confidence of the k'th trajectory, $\bar{x}_t^k$ is the predicted x-coordinate of the k'th trajectory at frame t, and x_t^k is the true x-coordinate at frame t. In line with a suggestion from Kaggle user KKiller, we multiply the loss by three to more closely match the loss on Kaggle:

$$L = -3 \times \log \sum_k e^{\log c^k - \frac{1}{2}\sum_t (\bar{x}_t^k - x_t)^2 + (\bar{y}_t^k - y_t)^2}$$

3.1 ResNet Baseline

Lyft trained a ResNet50 model [1] with MSE loss to serve as a baseline for this competition. This approach takes as input 10 sequential images of the scene (produced by rasterization) and predicts the x and y coordinates of an agent's position over the following 50 frames. In order to recapitulate the Lyft model as the baseline for our project, we downloaded a ResNet50 model pretrained on ImageNet, modified the input and output layers to be dimensionally compatible with our given task, and trained the network with the multi-mode prediction loss function. This allowed for results close to their baseline.

3.2 Geometric Approaches

Our initial intuition was that this problem was geometric. The standard approach and the most successful one was to use a ResNet to segment the images fed in and then compute a predictive trajectory with respect to the other agents identified on the map. However, this method is necessarily suboptimal since we are generating image and then trying to segment them. There is necessarily an error in the segmentation that we could bypass since we already have that information in the metadata.

Geometric approaches have the tremendous advantage that we don't need to rasterize images of the scene (which requires substantial computational time) and we can learn directly on numeric variables in the scene metadata, such as the centroids of all agents. As an example of comparison, running a ResNet of the full dataset takes a 8 days whereas running a GNN takes 18 hours and a PointNet takes less than 8 hours.

3.2.1 PointNet

PointNet [5] is a deep-Learning algorithm that takes as input an unordered point cloud. This approach is really different from the main approaches. It was initially proposed by KKiller, a competitor within the competition. We can feed an unordered set (PointCloud) in this model made of the different features of the scene. As an arbitrary choice recommended by the competition host, we chose 8 of the most significant agent related features to train our model (Position, yaws, type etc). Our PointNet had 21 milion parameters, allowing us to feed in a Batch size of 115 points in a GPU of size 11GB.

3.2.2 Graph Neural Network

PointNet is a great layer for a Neural Network as it allows to run 1D convolution on an unordered Point Cloud. However, the type of convolution is pretty simple and it clusters points in centroids, thus running convolution only on the closest neighbors in the latent space.

To run convolutions on the entire Point Cloud we designed a Graph Neural Network. We used a maximum of 150 agents per frames and we connected them all together. We then designed a Graph Neural network for multi-modal prediction to run on this fully connected graph using Sage Convolution. The update rule of a Sage-Convolution [6] is detailed below. The message passing step agglomerates of each agents. The matrix W_1 and $W2$ allows to put more weights on the relevant nodes.

$$\mathbf{x}_i' = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W_2} \cdot \text{mean}_{j \in \mathcal{N}())} \mathbf{x}_j$$

There is many type of graph convolution that are featured in *torch geometric*, the sub module of Pytorch we used to generate our graph. The go to type for trying would be *Graph Convolutions* [7]. We also tried GIN convolution which are often used as a very powerful type of convolution [3]. However we opted out for Sage Convolutions which don't require a fully connected layer between each Graph Convolution Layer. Indeed the use of GIN induced a lot of additional parameters that were overloading our GPU. We obtained better result using a larger batch size permitted by the fewer number of parameters of *SageConv*. We had 2M parameters in our final 7 layers network.

3.3 Encoder-Decoder seq-seq translation

Because the problem we are given is intrinsically a sequence prediction problem, we also evaluated a sequence-based approach. This neural network is a encoder-decoder model using two LSTM's and is adapted from an implementation by Kaggle user Kramadhari. We modified the encoder-decoder implementation to add the historical yaws as well as centroids to the feature vectors and to output multi-mode predictions. To further optimize this model, we tuned the size of the encoder and decoder hidden layers as well as the size of the first fully connected layer.

The original model had encoder and decoder with hidden layer size 128 and fully-connected layer size 256. Because multi-mode prediction results in an output of size 303, we experimented with increasing the size of these layers. We report the results for two high-performing models; the "large hidden layer" model has hidden layer of size 512, and the "large fc layer" model has fully connected layer of size 512.

3.4 Encoder-Decoder seq-seq translation with ResNet

We also designed a model that uses ResNet to generate embeddings of the ten historical frames, which are inputted as a sequence to the LSTM. The final output of the LSTM is fed into a fully connected layer with ReLu activation, and then a fully connected layer with no activation. The standard dataloader creates 2 separate images for each historical frame — 1 containing the ego and 1 containing all other agents. These two images were stacked and input to the ResNet to compose a 2-channel "image" for each historical frame.

Because of the very high time and computational cost of training a model incorporating ResNet, we used ResNet18 instead of ResNet50, which is the larger architecture used by the baseline model. We also trained a "large hidden layer; large fc layer" variant of this network with hidden layer of size 1024 and fully connected layer of 512.

4 Results and Analysis

Approach	Minimum validation loss
Resnet50, Lyft baseline	82.246
Resnet18	40.93
LSTM-LSTM encoder-decoder	77.29
LSTM-LSTM encoder-decoder; large hidden layer	105.23
LSTM-LSTM encoder-decoder; large fc layer	85.93
ResNet18-LSTM encoder-decoder	878.63
ResNet18-LSTM encoder-decoder; large hidden and fc layers	1494.16
PointNet	64.95
GNN	97.14

The ResNet model was very time-consuming to train, because it requires the rasterization of the images, that needs to be performed on a CPU. Because of the large size of the training set, each iteration takes a lot of time to process, so each experiment had to be carefully evaluated before running. We estimate that training ResNet on the full train dataset (crucial for performance) will take weeks if not over a month.

Our PointNet has allowed us to pass Lyft's baseline model on Kaggle, with a maximum score of 64 against the baseline score of 82. For the PointNet, we are taking batches of 110 sample points (as big as the GPU can hold) and we are running the model between 20 and 40 epochs with a cosine annealing scheduler (initially proposed). We tried to change the scheduler with a OneCycle scheduler to force the learning rate to get bigger values at the beginning of the training but the results where significantly worse than with the initial cosine annealing. Results are shown in Figure 2. Our interpretation of this result is that the cosine annealing allows to increase the learning rate several time, whereas it happens only once with the One Cycle scheduler, thus reducing the chances to be stuck in a local minimum. During our experimentations, we realized that our PointNet was relatively robust. In the sense that different choice of hyper-parameters had small effects. And the performance quickly stabilized after 20 epochs for roughly 7 hours of training. Also, as seen in figure **??**, we identified a learning rate to be optimal between 0.001 and 0.003. The blue curve is the under-fitting learning curve for a learning rate of 0.0003.

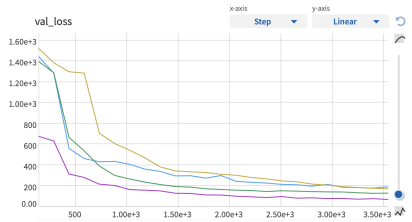

Figure 2: The purple curves is the validation loss for a cosine annealing scheduler, the three curves above are the results for different experimentation of a One Cycle Scheduler. In x we can see the number of the batch fed in the networks, and in y it is the value of the validation loss

We also implemented a Graph Neural network, which proved to have a similar efficiency. Since we are using a fully connected GNN, the "perceptive field" of each agent is every other agent in the scene whereas in the PointNet it was only the agents that were close to it. Naturally, we are processing a bigger representation of the scene since we need to store as well the edges. It forced us to reduce the batch size to 15 scenes and augmented the time per epoch from 30 minutes for the PointNet to an hour and a half for the GNN. However, our Graph Neural Network showed interesting expressiveness. The loss were still significantly reducing after 15 epochs whereas for our PointNet it was roughly stabilized at 15 epochs. To improve both of our model, we would advise our reader to fine-tune the size of the spheres used the Furthest-sampling-point algorithm for the PointNet [5] as it is essential to compete significant information around each agents. Since the GNN was particularly long to train, we couldn't experiment as much on it. But we would advise the reader to work on reducing the number of edged in the graph since it is what take the most space in our GPU.

A drawback of the sequence-to-sequence LSTM approach is that it fails to capture data about the scene that can only be derived from images, such as positions of other vehicles, lane markings, and

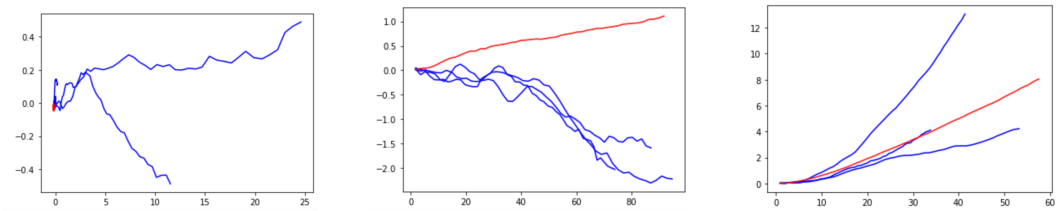

Figure 3: Three sample trajectories produced by the LSTM-LSTM encoder-decoder. Ground truth in red; predicted trajectories in blue

so on. Nevertheless, this model achieved comparable minimum average validation loss to the ResNet baseline. This may be because most ego's tend to follow fairly predictable trajectories (such as moving forward in a straight line, or moving in a parabola when turning a corner). It is likely that the sequence-to-sequence LSTM approach successfully predicts many of the most typical trajectories, but cannot predict complex trajectories that depend on the trajectories of multiple agents in the scene — for example, when a car in the left turn lane waits before turning because of traffic in the other lane.

To visualize the results, we provide three sample trajectories output by our best-performing LSTM-LSTM encoder-decoder model in Figure 3. The trajectories represent predictions with (1) high, (2) intermediate, and (3) low loss. These trajectories are usefully for gaining intuition about this model — in general, this model seems to prefer long, consistent trajectories. In case (1), we see that the ego remained stationary but 2 out of 3 trajectories predicted by the model show substantial motion. In case (2), the model simply predicts movement in the incorrect direction. In case (3), the model captures the groundtruth trajectory well.

In order to capture some of these high-level relationships between elements of the scene, we experimented with using a ResNet to generate embeddings of historical frames, which are fed into the LSTM. Because of the very long training time for the ResNet, we were not able to train this model for long enough to reduce the validation loss substantially. This approach is especially challenging because of the relatively large number of images that must be trained on in each iteration — for example, if the batch size is 8, and each example has 10 historical frame and 1 current frame, then the ResNet encoder must be trained on 88 different images, instead of 8. This greatly increases the computational demand of this approach, and led us to use the smaller ResNet18 model instead of ResNet50. It is likely that with more training time and resources, this approach could be optimized to produce better results than the sequence-to-sequence LSTM because it captures higher-level relationships between elements of the scene.

5 Conclusions

Of the models that we trained, three achieved lower validation error than the Lyft baseline, and one achieved comparable validation error to the Lyft baseline. Therefore, we believe that our project was successful in its goal of creating competitive models for the Lyft competition. Despite being architecturally simple and also smaller than ResNet50, ResNet18 achieved the best performance out of all of the models that we tested.

6 Contributions

Augustine Chemparathy ran experiments on the LSTM-based encoder-decoder models and the resnet-LSTM encoder-decoder model. Hugo Vergnes ran experiments on the PointNet and GNN approaches. Both authors wrote the paper and prepared the video. For further information of our project, we would advise the reader to look our github repo that can be found here: https://github.com/BowenRaymone/KaggleLyftCompetition

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[2] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. One thousand and one hours: Self-driving motion prediction dataset. https://level5.lyft.com/dataset/, 2020.

[3] Jure Leskovec Stefanie Jegelka Keyulu Xu, Weihua Hu. How powerful are graph neural networks? 2019.

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[5] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017.

[6] Jure Leskovec William L. Hamilton, Rex Ying. Inductive representation learning on large graphs. 2019.

[7] Tong H. Xu J. et al. Zhang, S. Graph convolutional networks: a comprehensive review. 2019.