Motion Prediction for Autonomous Vehicles

Project for E599: High Performance Big Data Systems

Madhura Bartakke
Masters in Data Science
Indiana University, Bloomington
mabartak@iu.edu

Neha Tayade
Masters in Data Science
Indiana University, Bloomington,
ntayade@iu.edu

KEYWORDS

Motion prediction, Distributed Computing, ResNet, LSTM, PyTorch, RNN, CNN, GPU

1. INTRODUCTION

Highly automated technology, self-driving cars, promise a range of potential benefits like greater road safely, enhanced independence for seniors and disables, ability to recapture driving time, fewer roadway crashes and reduced traffic congestion. Dynamic ability of these systems to perceive, understand and anticipate surrounding agents' behavior for predicting future positions using deep neural networks facilitates motion prediction using surrounding agents by planning its trajectory.

This project aims to embed Level 1 motion planning[1] within vehicles using various algorithms to predict best trajectories and paths of advancement by critically evaluating movements of other traffic agents. To achieve this, we plan on designing a Residual Neural Network (ResNet) as the baseline model and extending analysis to other deep neural network algorithms like Long short-term memory (LSTM) and MotionNet to predict motion of objects and their next locations. Finally, we aim to compare and contrast train, validate and test losses across these models and conclude with the best model.

Driving decisions are computed in a modular perception-planning-action pipeline. The components of the modular pipeline[2] can be designed based on AI and deep learning methodologies and a safety monitor is designed to assure the safety of each module.

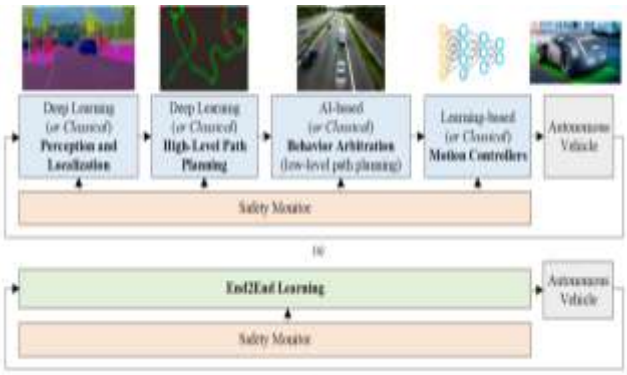

Fig 1: Modular self-driving pipeline

Given a route planned through the road network, the first task of an autonomous car is to understand and localize itself in the surrounding environment. Based on this representation, a continuous path is planned, and the future actions of the car are determined by the behavior arbitration system. Finally, a motion control system reactively corrects errors generated in the execution of the planned motion. Given the huge dataset at-hand and the deep learning-AI paradigm for hierarchical decision-making process, we use data parallelism and model parallelism in a virtual environment set-up to distribute computational load on a single or multiple GPUs to increase the overall efficiency and decrease latency.

2. RELATED WORK

Much related work has been done around perception tasks which are supervised tasks of estimating 3D position of objects around the autonomous vehicle. For most problems, deep learning approaches are now state of art such as 3D object detection and semantic segmentation. For prediction tasks, deep learning approaches leveraging the birds-eye-view representation of the scene or graph neural networks have proved themselves as leading solutions for this task. We intend to gauge above work to improve comprehensiveness of datasets and methodologies.

2.1. NEURAL NETWORK

One of the earliest reported use of a neural network for autonomous motion comes from the Autonomous Land Vehicle in a Neural Network (ALVINN) system[3]. It was a straightforward architecture consisting of fully connected layers. The network predicted actions from pixel inputs applied to simple driving scenarios with few obstacles. This research showcased the untapped potential of neural networks for autonomous navigation.

2.2. CONVOLUTIONAL NEURAL NETWORK

In 2016, NVIDIA used a CNN architecture[3] to extract features from the driving frames. The network was trained using augmented data, which was found to improve the model's performance. Shifted and rotated images were generated from the training set with corresponding modified steering angles. This approach was found to work well in simple real-world scenarios, such as highway lane-following and driving in flat, obstacle-free courses.

2.3. GENERATIVE ADVERSARIAL NETWORK

Comma.ai has proposed to consider autonomous navigation as equivalent to predicting the next frame of a video. It combines Variational Auto-encoder (VAE) and a Generative Adversarial Network[3] (GAN) to keep predicting realistic looking video for several frames based on previous frames despite the transition model optimized without a cost function in the pixel space. In many scenarios, video prediction is ill-constrained as preceding actions are not given. The model addresses this by conditioning the prediction on its previous actions.

2.4. DEEP REINFORCEMENT LEARNING

Deep reinforcement learning's[3] (RL) success in the field of autonomous navigation is demonstrated by learning of games like Atari and Go by Google DeepMind. In this method, models are trained using a predefined reward function such that it can learn and correct itself from past experiences. In these works, the main aim is to learn purely from experience and discover hierarchical structure automatically. This is hard, time intensive, and is, in

general, an open problem, particularly for sensorimotor skills with the complexity found in self-driving cars.

3. ARCHITECTURE AND IMPLEMENTATION

3.1. ARCHITECTURE

Here, we first implement the forecasting process in a shared virtual environment using GPU tensors through PyTorch library. Then, we optimize the neural network performance on different models. Distributed method of sharing the computation resources on multiple worker nodes is adopted to parallelize the process. A python supporting package called L5kit is used for rasterization, visualization, and modeling by fetching .zarr file contents into arrays and process them in batches. We worked on our baseline model ResNet18 in the initial phase and later extended this work to ResNet50 and LSTM.

3.1.1. L5KIT

L5KIT library is developed by Lyft Level 5 self-driving division and is open to external contributors. It supports functionality from reading data to framing and visualizing the simulation problems. The purpose of this framework is to enable engineers and researchers to experiment with data-driven approaches to execute plan and simulate tasks using any real-world driving data and contribute to state-of-art solutions. This can be used to build systems which:

- Make prediction, planning and simulation problems turn into data problems. Train them on real data.
- To model key aspects of Autonomous Vehicle stack using neural networks
- To predict future movements of cars around the Autonomous Vehicles environment using historical observations.
- To plan behavior of an Autonomous Vehicle to imitate human driving.

3.1.2 RESNET

Residual Network (ResNet) is a Convolutional Neural Network (CNN) architecture which was designed to enable hundreds or thousands of convolutional layers. Previous CNN architectures had a drop off in the effectiveness of additional layers. ResNet stand above these architectures which can add many layers that result in strong performance.

Problem - Vanishing gradient: ResNet is a solution to the vanishing the gradient problem. ResNet trains neural network by backpropagation that relies on a process called gradient descent which is moving down the loss function to find the optimal weights to minimize it. For too many layers, repetitive multiplication makes the gradient smaller which results in disappearance of gradients causing performance to degrade with each additional layer.

Solution - Skip connections: These identity shortcut connections also known as skip connections. ResNets stack up identity mappings on layers that initially did not contribute anything and reuses the activations from previous layers. Thus, skips result into compression of few layers which results in faster learning. The residual part of the network explores more and more feature space of the source image.

Architecture - ResNet captures the spatial resolution across the sequence of frames from scenes of any agent to capture the contextual information of the vehicle motion. This includes 5 convolutional layers where a filter with learned weights from pre-trained baseline model is convolved over the input. This is followed by a pooling layer that reduces the spatial size by sub-sampling the semantic image vectors. Finally, the fully-connected layers map input to the desired output dimensions.

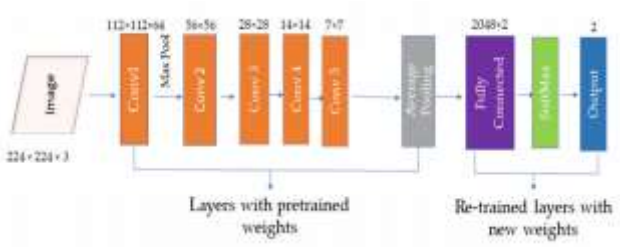

Fig 2: ResNet architecture

3.1.3. LSTM

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems.

Problem - Long term dependency: Some RNNs fail to capture long-term dependencies due to exponentially smaller weights through the layers. Gradients propagated over many stages tend to either vanish (most of the time) or explode (damaging optimization)[5].

Solution - LSTMs have shown their effectiveness in modeling long-range dependencies where the influence by the distant states on the present and future states can be adaptively adjusted.

An has LSTM cell blocks in place of our standard neural network layers. These cells have various components called the input gate, the forget gate and the output gate.

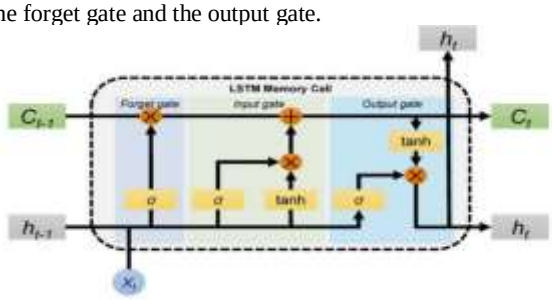

Fig 3: LSTM Cell architecture

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It is very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through”. An LSTM has three of these gates, to protect and control the cell state.

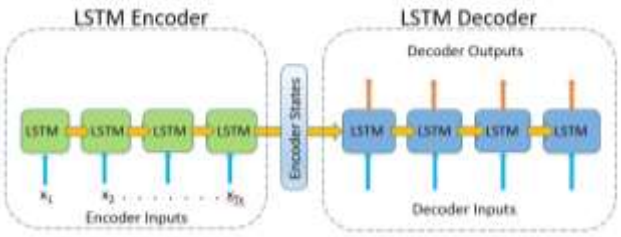

Fig 4: LSTM architecture

Architecture - LSTM, like any other RNN, captures the long-term dependencies of temporal information from the image sequences with the help of input, latched and output gates. This generates hidden and cell state for every frame in the sequence. The Encoder with hidden layer summarizes these historic predictions[7] into a prediction-based trajectory followed by a Decoder, with same configuration, uses weighted Gaussian Mixture Model to generate prediction intention and map it onto an initial future trajectory[5]. The 4-layer model reads entire target sequence, offsets it by 1 timestamp and predicts the next timestamps with highest probabilities.

3.2. IMPLEMENTATION

The intention of this project is to set a baseline using ResNet18 and then compare performance between ResNet50 (CNN) and LSTM (RNN) on big datasets that involve spatial and temporal features. However, considering the data, there is a dependency on the L5Kit to create and process datasets and visualize the predicted results. The dataset has instances from 25 seconds long video streams which are sequences of frames in a scene for unique agent ids. Size of the dataset being 22GB makes it necessary to use a distributed environment for computation and cut down on execution time. The two approaches used are Data Parallelization and Model Parallelization that evaluate the processing time and learning curves to infer single and multiple GPU processing. The implementation steps are as follows:

1. To prepare the train, valid and test chunked datasets, we rasterized the three datasets to get visual data as multi-channel tensors and turn them into RGB images.
2. Based on the configurations to fit, such as batch sizes, raster dimensions, historic frames batches for calculations, future frame lengths, number of worker nodes, we drafted a configuration dictionary to initialize them.
3. Then we extracted the arrays for image frames, agent information and scenes into pandas dataframes.
4. First constructed the ResNet Baseline model following with ResNet50 and LSTM.
5. Optimized the network using Adam (RMSProp + Adaptive learning) method and MSE loss function.
6. Compared the losses of train, valid sets to those obtained from the test set for evaluation of correctness of the model. Recorded time stamps, agents ids and coordinates for 224 raster points.

3.2.1. DATA PREPROCESSING

1. Many machine learning models operate on images, but ignore the fact that images are 2D projections formed by 3D geometry interacting with light, in a process called rasterization. Enabling ML models to understand image formation is the key for generalization. This achieved by the rasterization[4] which is the step to transform images in constructor, map them into (x,y) coordinates and convert into a series of vectors. The raster parameters are as follows:

- Raster size: [224, 224]
- Pixel size: [0.5, 0.5]
- Map type: 'py_semantic'

2. Agent Dataset iterates over valid agents indices to match the first valid agent in the dataset to the index of the corresponding scene. This assigns the scenes and frames with the same index to the respective Agent Id.
3. Chunked Dataset makes the dataset active on the disk in the form of compressed chunks which makes data loading and writing interfaces easy to use as it involves NumPy-like slices. This interface between raw data on the disk and Python accessible information provides the arrays mapped from the disk. When one of these arrays is indexed (or sliced), zarr identifies the chunk(s) to be loaded. The chunk is then decompressed on the fly after which the NumPy array copy is returned.

3.2.2. CONFIGURATIONS

1. Batch configurations

For every batch in data loader, the following configurations were chosen to compute gradients for every epoch during training and validation.

- Epochs: 10
- History frames: 10
- Future frames: 50
- Channels: (history frames + 1)*(number of coordinates)

2. ResNet18

For the baseline model, we choose ResNet18 with 4 layers having 2 Basic Blocks each. Followed this with a fully connected linear layer having 100 output features, filtered out of the 512 input features. The following model parameters are used:

- Batch size (Train, Valid, Test): 10, 8, 10
- Learning rate: 0.001
- Criterion: Mean Squared Error
- Optimizer: Adam
- Number of steps: 100

3. ResNet50

ResNet50 model consists of 5 convolution layers followed by a fully connected layer. This model has 2048 input features and 100 output features. The following model parameter are used:

- Batch size (Train, Valid, Test): 16, 32, 8
- Learning rate: 0.001
- Criterion: Mean Squared Error
- Optimizer: AdamW
- Number of steps: 10000

4. LSTM

LSTM model consists of Encoder and Decoder[6] followed by a fully connected layer. The model has 128 input features and 100 output features. The following model parameters are used:

- Batch size (Train, Valid, Test): 16, 32, 8
- Learning rate: 0.001
- Criterion: Mean Squared Error
- Optimizer: AdamW
- Number of steps: 10000
- Schedular: LR Schedular

After training both models, we froze the best checkpoint and used it to run inference on single GPU and attempted on multiple GPUs to evaluate the best model.

4. EXPERIMENTS

4.1. ENVIRONMENT SETUP

1. Single GPU environment
- Kaggle kernel: NVIDIA TESLA P100 GPU
- Cuda version: 9.2
- Single computation runtime: GPU- 994s, CPU-13419s

2. Distributed environment
- Future Systems, Juliet Server, Romeo nodes
- Kernel: 6 NVIDIA TESLA K80
- Cores: 136 CPU, 161792 GPU
- Storage: 2.4 SSD, 48 HDD

4.2. DATASET

The full data was collected by a fleet of 20 autonomous vehicles over a four-month period which consists of around 1000 hours[1] of data by Lyft's level 5 self-driving division. It consists of approximately 44,000 scenes, where each scene is 25 seconds long. The scenes capture the perception output of the self-driving system, which encodes the precise positions and motions of nearby vehicles, cyclists, and pedestrians over time. On top of this, the dataset contains a high-definition semantic map with 15,242 labelled elements and a high-definition aerial view over the area. The data is packed in .zarr files and are loaded using the zarr Python module and are also loaded natively by l5kit. .zarr file contains a set of:
- Scenes: Driving episodes acquired from a given vehicle for capturing aerial and semantic views.
- Frames: Snapshots in time of the pose of the vehicle in the form of time frames.
- Agents: Four agent label probabilities captured by the vehicle's sensors.

Files in the dataset:
- Aerial_map - an aerial map used when rasterization is performed
- Semantic_map - a high-definition semantic map used for rasterization
- Sample.zarr - a small sample set, designed for exploration
- Train.zarr - the training set, in .zarr format
- Validate.zarr - a validation set
- Test.zarr - test set, in .zarr format

The analysis of train, test and validate frame counts over time present in ratio 83:7:10 is given in the diagram below:

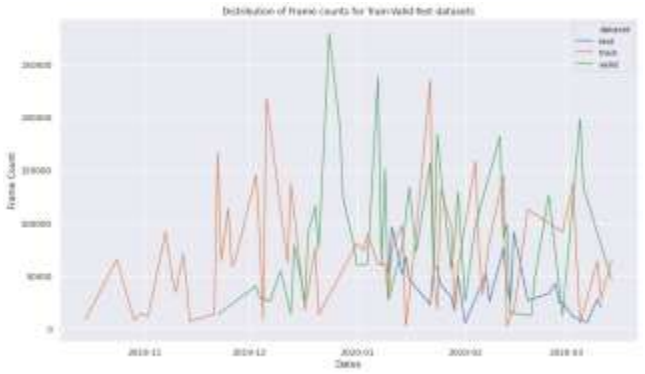

Fig 5: Frame count distribution for Train, Test and Validation sets

4.3. RESULTS

1. Learning loss

Loss curves for the three models were recorded for 10000 steps for above mentioned batch configurations.

No.	Model	Split	Loss
1.	ResNet 18	Train	10.172
		Validation	12.817
2.	ResNet 50	Train	6.446
		Validation	9.056
3.	LSTM	Train	5.425
		Validation	5.097

2. Time

No.	Model	Split	Time/iteration	Total time
1.	ResNet18	Train	1.06s	14580s
		Validation	1.13s	
2.	ResNet50	Train	1.69s	15120s
		Validation	1.91s	
3.	LSTM	Train	1.45s	13680s
		Validation	1.62s	

3. Distributed Computing

The distributed computing environment was hosted on FutureSystems computing server called Juliet. To do distributed training, the model would just have to be wrapped using DistributedDataParallel and the training script would just have to be launched using dist.init_process_group(). The data was sampled and distributed using DistributedSampler. The node communication bandwidth is extremely important for multi-node distributed training. This is achieved using --local_rank for argparse to specify the different parallel processes of the same load to be processed on 6 nodes for 10 epochs of train and validation. The host of all nodes use nccl backend during distributed training.

For each epoch, the data load was distributed along 6 nodes. Training for given batches across all the nodes were done simultaneously. The results from all the nodes were aggregated after training.

```
Local Rank: 0, Epoch: 1, Training...
Training Resnet...
Local Rank: 1, Epoch: 1, Training...
Training Resnet...
Local Rank: 2, Epoch: 1, Training...
Training Resnet...
Local Rank: 3, Epoch: 1, Training...
Training Resnet...
Local Rank: 5, Epoch: 1, Training...
Training Resnet...
Local Rank: 4, Epoch: 1, Training...
Training Resnet...
```

3. Image Results

In the below result images, following elements infer:

- Autonomous vehicle represented by green rectangles
- Blue rectangles represent agents. Some agents like vehicles are spotted on the road. Taking a closer look, we can spot few blue dots which represents other vehicles, pedestrians, and bicycles.
- The semantic colored yellow lines represent paths along which we must predict the motions of external agents.
- The green dotted line depicts predicted agent's motion

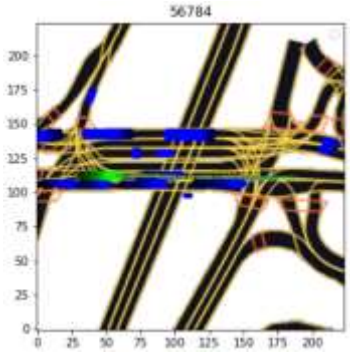

Fig 7: Single frame from a scene

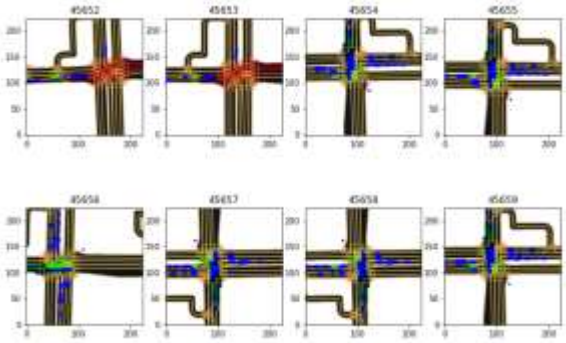

Fig 8: 6 consecutive frames from a single scene

4. Analysis of results

The performance was evaluated based on learning curves, time efficiency and visualization. Loss curves are evaluated on the metric by which the parameters of the model are optimized. For ResNet50 and LSTM, train and validation losses are decreased to a point of stability with a minimal gap between the two final loss values. Time efficiency is the efficiency of the algorithms used to measure how many computations are needed to find the solution across different input sizes. Visualizations are used to plot the predicted trajectory on semantic map for next 50-time frames.

The training losses for 10000 steps across all the three models signifies that gradual decrease in losses. Despite having a decent training loss for ResNet50, LSTM outperforms the other two models across all levels (learning curves, losses, training time, visualizations). LSTM took 1.45s per iteration which is a reasonable amount of training time as against ResNet18 and ResNet50 which took 1.06s and 1.69s, respectively.

6. CONCLUSION

1. A summary of achievements

In this project, our goal was to choose and compare different models for predicting motion in future time frames for an autonomous vehicle. We focused mainly on three models: ResNet18 (baseline), ResNet50 and LSTM. The baseline was trained from scratch. The other two models were trained on pre-trained weights from ResNet18. The results from all the three models were fairly good, however LSTM outperformed the other two models on evaluation parameters (learning curves, time efficiency and visualizations). Compared to the other models LSTM had lowest processing time per iteration. The train and valid curves too closely followed each other. Since the dataset was huge, we could achieve decent efficiency using model parallelization.

2. Findings

One of the biggest challenges of the project was to use the L5Kit which uses functions and parameters in a rasterize manner for a map like input images. Processing and visualizing data specific to our goals required a lot of research. The other major challenge was implementing distributed GPU computing which required familiarity with PyTorch Distributed.

Lot of computational overhead was generated to completely utilize the available resources and uniformly distribute the load over the available GPUs. The training performed on multiple GPUs but test loop could be assigned to only one default GPU.

3. Recommendations

Further work can be done by applying Reinforcement learning[2] techniques. Reinforcement learning allows you to learn from mistakes generated by reward system instead learning from a labelled or unlabeled dataset. Development platforms for reinforcement learning in self-driving cars:

- Voyage Deep Drive - Simulation platform where you can build reinforcement learning algorithms in a realistic simulation.
- Deep Traffic - A course project launched by MIT where you can try and beat traffic using Deep Reinforcement Learning algorithms and a simple simulator.

By taking benefits of ResNet and LSTM in a single model we can use both, temporal and spatial features to predict motion of autonomous vehicles.

7. ACKNOWLEGDEMENT

To Professor Judy Qiu and Teaching Assistant Selahattin Akkas who were always present to help during the course of this project. They provided us with computing resources, guidance, and advice which helped us solve the issues we confronted. To all the Github, Kaggle and StackOverflow communities without whom this project would not have been a success.

6.REFERENCES

[1] John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, Peter Ondruska. One thousand and one hours: Self driving motion prediction dataset. Link

[2] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, Gigel Macesanu: A Survey of Deep Learning Techniques for Autonomous Driving. Link

[3] XQ: Literature Review on the Application of Deep Learning in Building Self Driving Cars. Link

[4] Siddhesh Khandelwal, William Qi, Jagjeet Singh, Andrew Hartnett, Deva Ramanan. What-if Motion Prediction for Autonomous Driving. Link

[5] Nachiket Deo, Mohan Trivedi. Multi-Modal Trajectory Prediction of Surrounding Vehicles with Maneuver based LSTMs. Link

[6] Yeping Hu, Wei Zhan, Masayoshi Tomizuka. Probabilistic Prediction of Vehicle Semantic Intention and Motion. Link

[7] Sajjad Mozaffari, Omar Y. Al-Jarragh, Mehrdad Dianati, Paul Jennings, and Alexandros Mouzakitis. Deep Learning-based Vehicle Behaviour Prediction for Autonomous Driving Applications: a Review Link