Trajectory prediction of dynamic agents around an autonomous vehicle

Report submitted in fulfillment of the requirements
for the Undergraduate Project of

Fourth Year B.Tech

by

Anwoy Chatterjee (Roll No.: 18075075)

Under the guidance of

Prof. Rajeev Srivastava

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI
Varanasi 221005, India

Dedicated to

My parents, teachers, ...

Declaration

I certify that

1. The work contained in this report is original and has been done by myself and the general supervision of my supervisor.
2. The work has not been submitted for any project.
3. Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references.

Place: IIT (BHU) Varanasi

Date: 28.11.2021

Anwoy Chatterjee

B.Tech. Student
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Certificate

*This is to certify that the work contained in this report entitled "**Trajectory prediction of dynamic agents around an autonomous vehicle**" being submitted by **Anwoy Chatterjee (Roll No. 18075075)** carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of our supervision.*

Place: IIT (BHU) Varanasi
Date: 28.11.2021

Prof. Rajeev Srivastava
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Acknowledgments

I would like to express my sincere gratitude to Prof. Rajeev Srivastava Sir and Mrs. Divya Singh Ma'am for their able guidance and support in carrying out the project. I would also like to thank all the Professors of our department, my parents and my friends for their constant support in these tough times.

Place: IIT (BHU) Varanasi
Date: 28.11.2021

Anwoy Chatterjee

Abstract

As autonomous vehicles navigate in highly-uncertain and interactive environments shared with other dynamic agents, like other vehicles or pedestrians, predicting the trajectories of the surrounding agents is essential for autonomous vehicles in order to plan safe and comfortable maneuvers. In this report, an approach has been presented for long-term trajectory prediction of the road-agents using a combination of graph temporal convolution layers along with LSTM-based recurrent neural networks. Our approach have been evaluated on the Apolloscape, Lyft and Argoverse datasets and the benefits over the prior trajectory prediction methods have been highlighted. In practice, our approach outperforms the prior methods on these datasets with respect to long-term prediction of the trajectories. The improvement in performance of our approach with respect to the prior methods is more for the sparse datasets compared to that for the dense datasets.

Contents

Chapter 1

Introduction

1.1 Overview and Motivation of the Research Work

Autonomous driving is currently an active area of research and it includes many problems related to trajectory prediction and understanding of the behavior of the surrounding dynamic agents. The safety of a self-driving car is a challenging problem which needs to be given more attention to facilitate extensive use of autonomous vehicles. For avoiding accidents, it is necessary for the autonomous vehicles to accurately predict the trajectories of the surrounding road-agents. Trajectory prediction is the problem of predicting the short-term (1 to 3 seconds) or long-term (3 to 5 seconds) spatial coordinates [1] of various road-agents such as cars, buses, bicycles, pedestrians, and even animals, etc. Thus, trajectory prediction problem is crucial for safe navigation of the autonomous vehicle.

A major challenge in the trajectory prediction is ensuring accurate long-term prediction (3 to 5 seconds). As the prediction horizon increases, error in the predictions also increases, often due to the diminished influence of the temporal correlation. We have tried to resolve this problem in the long-term prediction of trajectories.

1.2 Problem Statement

Let us first formally define the trajectory of a road-agent.

Trajectory of a road-agent: The trajectory for the i^{th} road agent is defined as a sequence $P_i(a,b) \in \{R^2\}$, where $P_i(a,b) = \{[x_t, y_t] \mid t \in [a,b]\}$. $[x,y] \in R^2$ denotes the spatial coordinates of the road-agent in meters according to the world coordinate frame and t denotes the time instance.

Our aim is to design a model to predict the future trajectory of the road-agents for a certain interval using the information about their trajectory in the past few seconds.

The *Trajectory Prediction Problem* can thus be defined as:

Trajectory Prediction Problem: In a set of image frames each with N road agents, given the trajectory $P_i(0,\tau)$, predict $P_i(\tau^+, T)$ for each road-agent v_i , $i \in [0,N]$.

1.3 Organisation of the Report

The rest of the report is organised in the following format:

- **Chapter 2** discusses briefly about the recent works done in trajectory prediction.
- **Chapter 3** focuses on the pre-processing methods for each dataset, and the steps involved in generating the graphs from the pre-processed datasets.
- **Chapter 4** explains the architecture and working of our proposed model for the trajectory prediction problem, and mentions the tools or methods used for training the model.
- **Chapter 5** lists all the results and conclusions obtained in the project.

Chapter 2

Related Works

A lot of research has been done recently on Trajectory Prediction. These are not limited to only trajectory prediction of the road agents around an autonomous vehicle but also include the trajectory prediction of pedestrians in crowd and many are dedicated to trajectory prediction of only humans.

Paparusso et al. [2] designed a deep RNN based on Long Short-Term Memory autoencoders. This model fuses the information on the road geometry and the past driver-vehicle system dynamics to produce context-aware predictions. But it has certain limitations like it does not define a minimal set of drivers and tracks and thus has less generalisation capabilities.

Park et al. in [3] hypothesizes that future trajectories of human drivers should follow distributions of multiple modes, conditioned on the scene context and social behaviors of agents. The model is designed to explicitly capture both agent-to-scene interactions and agent-to-agent interactions with respect to each agent of interest. Since this model learns agent-to-agent interactions and agent-to-scene interactions it has better precision and better prediction but it is less time efficient.

For trajectory prediction of humans, Cheng et al. [4] trained two LSTM encoders to learn temporal context between steps by taking the observed trajectories and the

extracted dynamic spatial context as input, respectively. But this model does not have the method for learning the impact from the static context to further enhance the performance of trajectory prediction.

Styles et al. proposed a method [5] in which the future trajectory of an object is predicted in a network of cameras. It not only considers forecasting trajectories in a single camera view but also this work is the first to consider the challenging scenario of forecasting across multiple non-overlapping camera views. But it focuses on only next-camera prediction problem and not on full trajectory information.

Liu et al. [6] introduced a social contrastive loss that encourages the encoded motion representation to preserve sufficient information for distinguishing a positive future event from a set of negative ones. But in this method the random negative sampling is not able to provide any significant performance gain.

Ivanovic et al. [7] proposed Mixtures of Affine Time-varying Systems (MATS) as an output representation for trajectory forecasting that is more amenable to downstream planning and control use. Then multimodal planning methodology and significant computational improvements on a large-scale autonomous driving dataset are integrated. But this model feasibility is not guaranteed and it also does not currently guarantee safe behavior, given that we are reasoning about uncertain predictions of other agents.

Liang et al. [8] uses the hardest camera view to mix up with adversarial data from the original camera view in training, thus enabling the model to learn robust representations that can generalize to unseen camera views. But in case of fewer numbers of data and less diverse views, the performance drops. Weng et al. [9] unified the solutions for 3D multi-object tracking and trajectory forecasting, which also incorporates two additional novel computational units. First, a feature interaction technique by introducing Graph Neural Networks (GNNs) to capture the way in which multiple agents interact with one another. Second, a diversity sampling function to improve

the quality and diversity of the forecasted trajectories is used. It performs feature extraction to get impressive performance for each object independently, but pays less attention to the interaction between the objects.

Graber and Schwing [10] proposed a method which recovers interactions between entities at every point in time; this incorporates insights from sequential latent variable models to predict separate relation graphs for every time-step. But this method does not adapt additional methods used by recent sequential latent variable models, such as auxiliary loss functions, to further improve the performance. On the other hand, Deo et al. [11] proposed a method (CS-LSTM) which combines CNNs with LSTMs to perform trajectory prediction on U.S. highways, but this method does not consider the inter-object interactions. Chandra et al. [12] also used a CNN + LSTM approach (TraPHic) along with spatial attention-based pooling to perform trajectory prediction of road-agents in dense and heterogeneous traffic.

Gupta et al. [13] proposed a GAN-based trajectory prediction approach (Social-GAN) which is originally trained on pedestrian crowd datasets. This method uses the encoder-decoder architecture to act as the generator and trains an additional encoder as the discriminator.

Li et al. [14] (GRIP) proposed a graph-based trajectory prediction approach that replaces standard CNNs with graph convolutions and combines GCNs with an encoder-decoder framework.

Chandra et al. [1] presented an approach for traffic forecasting in urban traffic scenarios using a combination of spectral graph analysis and deep learning. Their formulation represents the proximity between the road agents using a weighted dynamic geometric graph (DGG), and uses a two-stream graph-LSTM network to perform traffic forecasting using these weighted DGGs.

Chapter 3

Pre-processing of datasets

We have used three datasets in this project:

- Apolloscape trajectory prediction dataset [15]
- Lyft level-5 Perception dataset [16]
- Argoverse Motion Forecasting dataset [17]

We first pre-processed each dataset to a text file where each row contains the columns: *Frame ID*, *Object ID*, *X coordinate*, *Y Coordinate*, and *Dataset ID*.

After pre-processing, the *Frame ID*s range between 1 to n, and *Object ID*s range from 1 to N. So, n is total number of frames and N is total number of objects/ road-agents. Each dataset uses a different convention to represent the *Frame ID*s (for example, few datasets use Timestamp as Frame ID), and these initial IDs are mapped to an integer between 1 to n. Similarly for *Object ID*s, there are also different initial representations (for example few datasets represent Object IDs using string of characters), and they are mapped to an integer between 1 to N. Even if the *Frame ID*s and *Object ID*s of any dataset already ranges between 1 to n, or 1 to N, it is confirmed that they are sequential and there are no missing IDs.

The *Dataset ID*s are used to differentiate different scenes of the same dataset, since the objects or, agents are not preserved across scenes.

The text files are then converted to .npy format. Based on the frame rate at which the trajectories of the vehicles are recorded in the concerned dataset, and the training sequence time and prediction sequence time interval, the training sequence length ($train\_seq\_len$) and prediction sequence length ($pred\_seq\_len$) for each dataset are determined. Further, it is ensured that: $frame\_length\_cap \geq (train\_seq\_len + pred\_seq\_len)$. We use this $frame\_length\_cap$ to enforce that an *Object ID* is present in at least $frame\_length\_cap$ number of frames.

If any dataset is too large, only few scenes from the whole data may be considered and the *Dataset ID*s list is used to tweak the values and reduce the amount of data. Finally, the pickle files are generated and these are used for training and evaluation of the models.

The algorithms for the pre-processing of each dataset are discussed separately in the following sections.

3.1 Pre-processing of the Apolloscape dataset

The Apolloscape trajectory prediction dataset [15] consists of 53 minutes of training sequences and 50 minutes of testing sequences captured at 2 frames per second. Each line of each text file in the dataset contains: *frame_id, object_id, object_type, position_x, position_y, position_z, object_length, object_width, object_height, heading*. Position is given in the world coordinate system. The unit for the position is *meter*.

Steps involved in pre-processing:

The creators of this dataset have done the processing of the image collected by the cameras associated with autonomous vehicles: the dynamic objects around are detected and their coordinates (in the world coordinate system) at each time-frame are

determined. The information available to us is in the form of several text files, and each line of the files contains information about a particular object at a particular time-frame. The steps involved in processing of these text files are:

1. First we generated a long numpy array from each text file. The shape of the array is initially $(l, 10)$, where l is the number of lines in the text file, and each line contains 10 columns.

2. We then mapped the $frame\_id$ in the first column to an integer between 1 to n, where n is the total number of frames.

3. Next, we mapped the *object_id* in the second column to an integer between 1 to N, where N is the total number of objects, or road-agents. It must also be ensured that the mapped id's are sequential and there are no missing id's.

4. Only those objects which are present at least t_h times (t_h is the no. of observation frames) are retained in the data, other objects are removed.

5. We then deleted the unnecessary columns in the data so that each row of the numpy arrays contains only four columns: $frame\_id$, *object_id*, *position_x*, *position_y*.

6. We then added an extra column for *dataset_id* to each row of the arrays for identifying the dataset in the future, as we are using multiple datasets.

7. Thus the final numpy array generated for each text file has shape $(l, 5)$, and each numpy array is saved as a file with *.npy* extension.

8. For the input to the model, we further divided the numpy arrays into batches, with each batch containing chunks of coordinates for *batch_size* number of objects during the observed time. Each chunk contains t_h pairs of coordinates, where t_h is the no. of observation frames.

3.2 Pre-processing of the Lyft dataset

The Lyft level-5 Perception dataset [16] is presented in the nuScenes format.

Steps involved in pre-processing:

Each file of the dataset, as shown in the schema above, is available to us is in the json format. So we first load the required json files. For our models, we will be requiring mainly the *sample.json*, *sample_data.json*, *sample_annotation.json* , and *scene.json* files. The steps involved in processing of these json files are:

1. First we extracted the list of *sample_token*-s and the corresponding *timestamp* (same as $frame\_id$) for any particular scene from the *sample.json* file. The list consists of several sub-lists, with each sub-list containing the *sample_token* and the corresponding *timestamp* to identify the time when the sample was recorded. This list is sorted based on the *timestamp* value.

2. We then created a large list containing all the data we require. This large list consists of many dictionaries. Each dictionary has four keys:

   - $'sample\_token'$ whose value is the same as the tokens in each sub-list within the list obtained in the previous step,
   - $'timestamp'$ whose value is value of timestamp corresponding to the *sample_token*,
   - $'sample\_data'$ whose value is collected from the *sample_data.json* file by using the corresponding value of *sample_token* as the foreign key, and
   - $'annotation\_data'$ whose value is collected from the *sample_annotation.json* file by using the corresponding value of *sample_token* as the foreign key.

3. We then collected the values of [timestamp, object_id, position] for each object present in each timestamp. For this, we again created a list containing many sub-lists, with each sub-list containing:

- timestamp,
- the *instance_token* (same as *object_id*), for uniquely identifying each object present in a frame, and
- a list containing the x, y, z-coordinates (in metres) of the corresponding object.

In this step, we also created an aggregated list containing all the *instance_token*s.

4. Next, we mapped the *instance_token* or *object_id* to an integer between 1 to N, where N is the total number of objects, or road-agents. For this, we created a dictionary, whose keys are the actual values of *instance_token*s as present in the aggregated list of instance_tokens obtained in the previous step, and the corresponding value is an integer between 1 to N.

5. We then created such a list each of whose sub-lists contain four values: $frame\_id$ (or, *timestamp*), *object_id* (or, *instance_token*), *position_x*, *position_y*. This can be done by combining efficiently the interconnected information stored in the lists and dictionaries obtained in the steps 2, 3 and 4.

6. We then mapped each *timestamp* or $frame\_id$ to an integer between 1 to n, where n is the total number of frames.

7. We now added an extra column for *dataset_id* to each sub-list of the list. In this case, we are assigning a different dataset id to each scene in the dataset, as the object id's are not preserved across scenes.

8. We converted our list to a numpy array. Thus the final numpy array generated has shape $(s, 5)$, where s is the total number of samples present in a set of data, and each numpy array is saved as a file with *.npy* extension.

9. For the input to the model, we further divided the numpy arrays into batches, with each batch containing chunks of coordinates for *batch_size* number of objects during the observed time. Each chunk contains t_h pairs of coordinates, where t_h is the no. of observation frames.

3.3 Pre-processing of the Argoverse dataset

The Argoverse Motion Forecasting dataset [17] is a curated collection of 324,557 scenarios, each 5 seconds long, for training and validation. Each scenario contains the information about the trajectories of each tracked object sampled at 10 Hz.

Steps involved in pre-processing:

The dataset consists of csv files, each file corresponding to a scenario. Each row of the csv file consists of the timestamp, object id, type of object, x and y coordinate corresponding to the position of the object, and the name of the place where the data was collected. The steps involved in processing of these csv files are:

1. In each csv file, the values of timestamp in the first column are mapped to an integer between 1 to n, where n is the total number of frames in the scene.

2. Then the *object_id* in the second column is mapped to an integer between 1 to N, where N is the total number of objects, or road-agents. It must also be ensured that the mapped id's are sequential and there are no missing id's.

3. Only those objects which are present in at least $(t_h + t_f)$ number of frames (t_h is the no. of training sequence frames and t_f is the no. of prediction sequence frames) are retained in the data, other objects are removed.

4. The unnecessary columns in the data are then deleted so that each row contains only four columns: $frame\_id$, $object\_id$, $position\_x$, $position\_y$

5. An extra column is then added for *dataset_id* to each row for identifying the different scenes of the dataset.

6. Finally, the data from all csv files are combined into a single large numpy array with shape $(l, 5)$ and the numpy array is saved as a file with *.npy* extension.

7. For the input to the model, the numpy array is further divided into batches, with each batch containing chunks of coordinates for the *batch_size* number of objects during the observed time. For each object, there are t_h pairs of coordinates, where t_h is the no. of observation frames.

3.4 Generation of the Graphs

We then generated graphs for each frame from the numpy arrays obtained after preprocessing from each of the datasets. The steps involved in this process are:

1. The data is first grouped based on the $frame\_id$. Each frame will have a separate graph.

2. The *object_id* of the objects in each frame are then encoded to a label between 0 to $N_f - 1$ for the creation of the nodes of the graph, so that the corresponding graph has N_f nodes.

3. Each node in a graph has two features, which are the x-coordinate and y-coordinate of the corresponding object or road-agent.

4. After this the edges of the graph are constructed. The edges are undirected. Out of the possible $\binom{N_f}{2}$ edges, only those are present for which $d(o_i, o_j) < \mu$, where o_i and o_j are the objects corresponding to the nodes connected by that edge, and $d(o_i, o_j)$ is the euclidean distance between the objects o_i and o_j. μ is a heuristically chosen threshold parameter, and here we have taken

$\mu = 10\ metres$, by considering the typical size of the road-agents and the width of the road [1].

5. The graph $G = (V, E)$ thus corresponds to a particular frame, where V is the set of objects in the frame and, E is the set of undirected edges generated based on the *step-4* as described above.

6. We have used the adjacency matrix for the representation of the graphs. To keep a fixed size for the input, the size of the adjacency matrix is made to be $N \times N$, where N is the maximum number of objects observed in a single frame for the corresponding dataset. If the number of objects in the current frame is less than N, all extra rows and columns are filled with zeros.

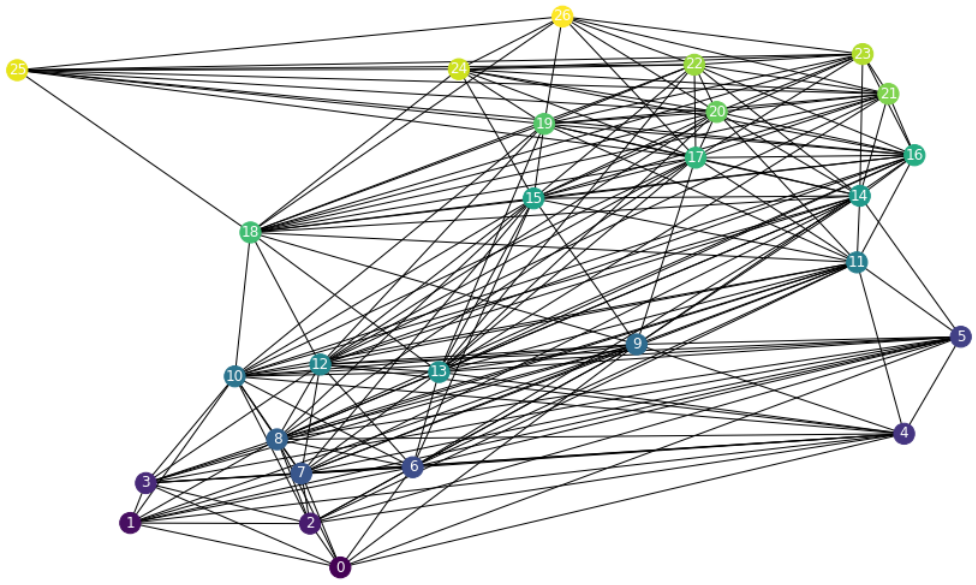

Figure 3.1 Visualization of the graph generated from the 511-th frame from our trainSet2 of ApolloScape Dataset. In this frame, no. of road agents = no. of nodes in the graph = 27, and, no. of edges = 197.

Chapter 4

Proposed Model

The general problem associated with the long-term prediction of the trajectories in most of the existing models lies in the fact that the relevance of the maneuver pattern of each object while making the future predictions, though captured initially, keeps on diminishing as the data progresses through the sequence-to-sequence model for more and more time-steps. Also most of these model do not give importance to inter-object interactions while predicting the trajectory of each object.

We explored a possible solution to this problem. We used three Graph Temporal Convolution layers in our model before giving the input to the encoder. These layers perform convolution operations as well as graph operations. The convolution operations are used to capture the temporal features and graph operations take into account the inter-object interactions. The details of the architecture, working, implementation and training of our model are elaborated in the successive sections.

4.1 Architecture of the model and its working

A schematic representation of the basic architecture of our model is shown below in Fig 4.1:

An algorithm for the working of the model is given below:

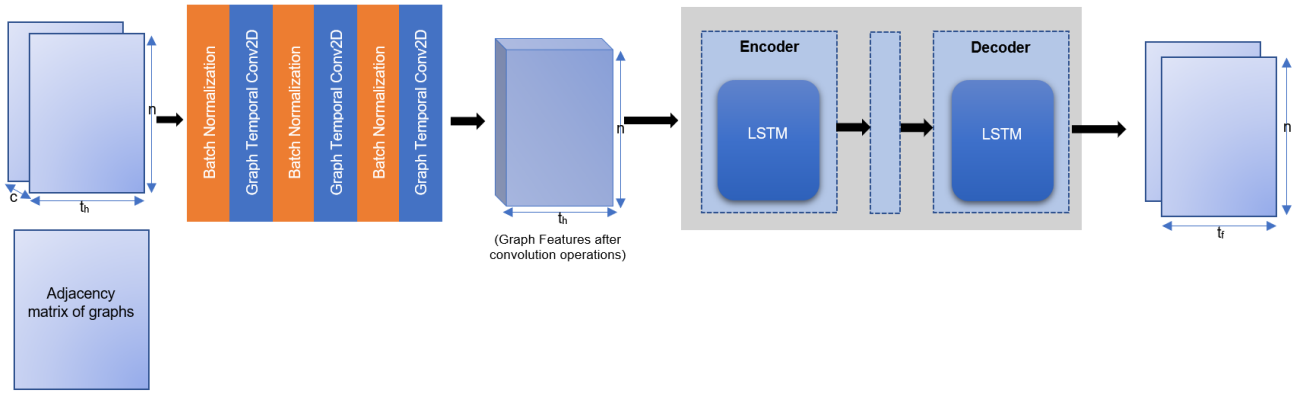

Figure 4.1 Architecture of our model

1. The pre-processed input data is of shape $(n \times t_h \times c)$, where c is the number of channels, which is two in this case, corresponding to the x and y-coordinates respectively. The input data is fed into a *Batch Normalization layer* to improve the training stability of the model. The shape of output from the Batch Normalization layer is the same as input.

2. Another part of the input is the Adjacency matrix of graphs, which help to take into account the interactions with other objects. As the Adjacency matrix consists of 0's and 1's only and don't need Batch Normalization, it is fed directly into the first *Graph Temporal Convolution layer*. Thus, the graph temporal convolution layer has two inputs - the input matrix with the coordinates of objects and the adjacency matrix for taking into account the interaction among the objects.

3. In the *Graph Temporal Convolution layer*, at first the convolution operation is applied using a kernel of size (1×3) to process the data along the temporal dimension which is the second dimension in our case. The stride is set to 1 and appropriate padding is done so that the shape of the feature matrix is maintained after the temporal convolution operation. The output of the convolution operation is a feature matrix, say $f_{temp-conv}$.

4. Once the temporal convolution is done, the interaction among the objects is now considered using the adjacency matrix of the graph. It involves multiplying normalized version of adjacency matrix A with $f_{temp-conv}$ using the following formula:

$$f_{final} = \sum_{j=0}^{1} \lambda_j^{-1/2} A_j \, \lambda_j^{-1/2} \, f_{temp-conv},$$

where λ_j can be computed as: $\lambda_j^{ii} = \sum_k (A_j^{ik}) + \alpha$. Here, $\lambda^{-1/2} A \, \lambda^{-1/2}$ is a normalized version of A to maintain the default range of the values of the coordinates in feature matrix, and $\alpha = 10^{-4}$ is used to avoid empty rows in A_j [14], [18]. It must be noted that as it is always not possible to perform the conventional matrix multiplication between *normalized* A and $f_{temp-conv}$ due to dimensional differences, we have used the `torch.einsum` function here to perform the operation according to the Einstein summation convention.

5. The output of the last Graph Temporal Convolution layer is fed into a Trajectory Prediction Model consisting of LSTM based Encoder-Decoder network.

6. The f_{final} generated by the last graph temporal convolution layer is given as input to the encoder LSTM at each time step.

7. Then, the hidden feature of the encoder LSTM as well as coordinates of objects at previous time-step are fed into a decoder LSTM to predict the position coordinates of the objects for the next time-step. Such a decoding process is repeated several times until the model predicts positions for all t_f time-steps.

8. The final output is the predicted future trajectories of the n objects for the next t_f time steps.

4.2 Training of the model

Loss Function:

We have used the *Mean Squared Error* loss function between the predicted trajectory and the corresponding ground-truth labels to train our model. The overall loss can be computed as:

$$Loss = \frac{1}{t_f}\sum_{t=1}^{t_f} loss_t = \frac{1}{t_f}\sum_{t=1}^{t_f} ||Y_{pred}^t - Y_{GT}^t||^2,$$

where, t_f is the number of time-steps in the future, $loss_t$ is the loss at time t, Y_{pred} and Y_{GT} are predicted positions and ground truth respectively. The model is trained to minimize the Loss.

Optimizer:

- *Adam* optimizer (`torch.optim.Adam`), with initial learning rate of 0.002 and other default parameters as in PyTorch, is used for training of the model.

The layers are trained by the optimizer with respect to the above mentioned loss function, using backpropagation.

Batch Size: 128

No. of epochs: 50

Chapter 5

Results and Conclusions

5.1 Graphical Results

Predicted trajectories and the originally provided groundtruth trajectories of some agents (two agents are randomly chosen from each dataset) are shown pictorially using graphs in this section.

Apolloscape dataset

In Fig. 5.1, the Ground-truth trajectory and the predicted trajectory of the object or road-agent which has been labelled with agent_ID=337 in the Apolloscape dataset is shown for the prediction time interval. The corresponding Average Displacement Error (ADE) and the Final Displacement Error (FDE), in metres, for this object in the considered interval are: **ADE=1.486**, and, **FDE=1.913**. Considering the length, width and height of the object, which are, 11.773 metres, 2.716 metres, and 2.86 metres respectively, the error in the predicted trajectory can be considered to be not much significant.

In Fig. 5.2, the Ground-truth trajectory and the predicted trajectory of the road-agent with agent_ID=659 in the Apolloscape dataset is shown. The corresponding Average Displacement Error (ADE) and the Final Displacement Error (FDE), in me-

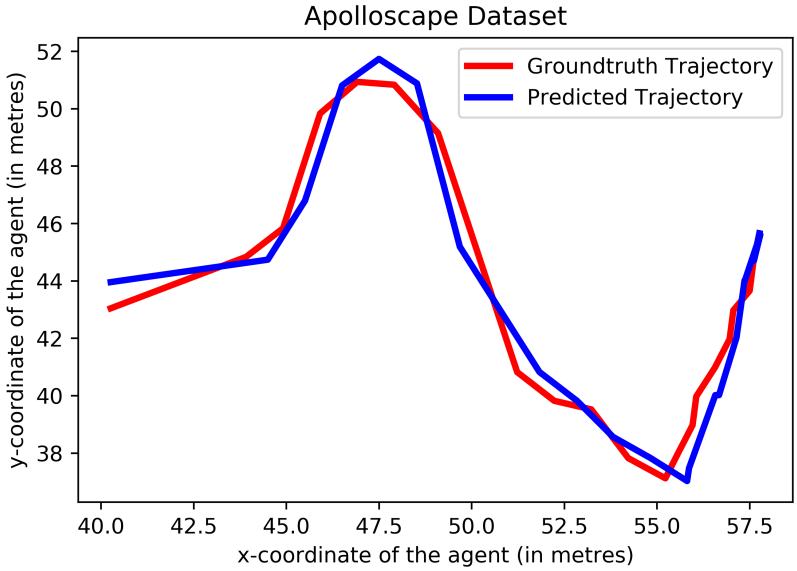

Figure 5.1 Groundtruth trajectory and predicted trajectory of the road-agent with *agent_ID = 337* of the Apolloscape Dataset

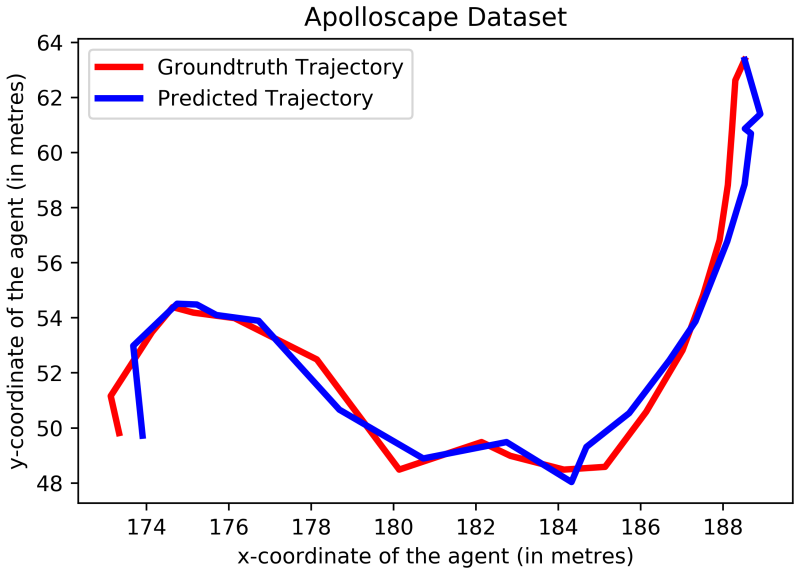

Figure 5.2 Groundtruth trajectory and predicted trajectory of the road-agent with *agent_ID = 659* of the Apolloscape Dataset

tres, for this object in the considered interval are: **ADE=1.727**, and, **FDE=1.945**. The length, width and height of the object being 9.835 metres, 2.764 metres, and 2.93 metres respectively, the error in the predicted trajectory can be considered to be relatively negligible.

Lyft dataset

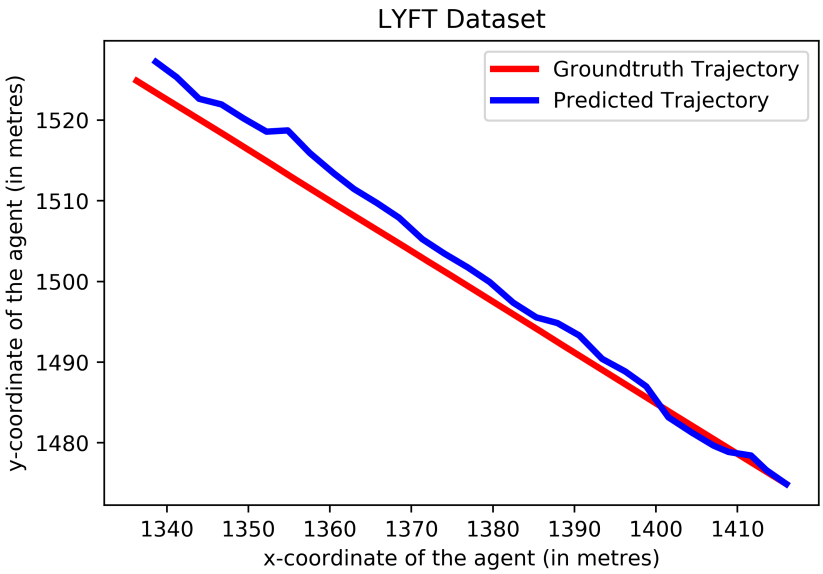

Figure 5.3 Groundtruth trajectory and predicted trajectory of the road-agent with *agent_ID = 2* of scene with *scene_ID = 127* of the Lyft Dataset

In Fig. 5.3, the Ground-truth trajectory and the predicted trajectory of the road-agent with agent_ID=2 in the scene labelled with scene_ID=127 of the Lyft (level-5) dataset is shown. The corresponding Average Displacement Error (ADE) and the Final Displacement Error (FDE), in metres, for this object in the considered interval are: **ADE=1.964**, and, **FDE=2.345**.

In Fig. 5.4, the Ground-truth trajectory and the predicted trajectory of the road-agent with agent_ID=16 in the scene labelled with scene_ID=141 of the Lyft (level-5) dataset is shown. The corresponding Average Displacement Error (ADE) and the

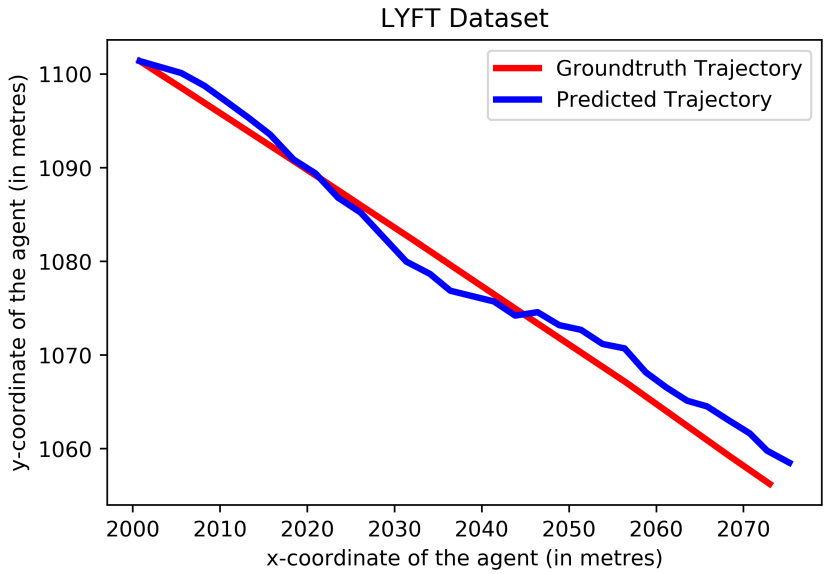

Figure 5.4 Groundtruth trajectory and predicted trajectory of the road-agent with *agent_ID = 16* of scene with *scene_ID = 141* of the Lyft Dataset

Final Displacement Error (FDE), in metres, for this object in the considered interval are: **ADE=2.047**, and, **FDE=2.258**.

Argoverse dataset

In Fig. 5.5, the Ground-truth trajectory and the predicted trajectory of the road-agent with agent_ID=11 of the Argoverse dataset is shown. The corresponding Average Displacement Error (ADE) and the Final Displacement Error (FDE), in metres, for this object in the considered interval are: **ADE=0.893**, and, **FDE=1.434**.

In Fig. 5.6, the Ground-truth trajectory and the predicted trajectory of the road-agent with agent_ID=27 of the Argoverse dataset is shown. The corresponding Average Displacement Error (ADE) and the Final Displacement Error (FDE), in metres, for this object in the considered interval are: **ADE=0.917**, and, **FDE=1.496**.

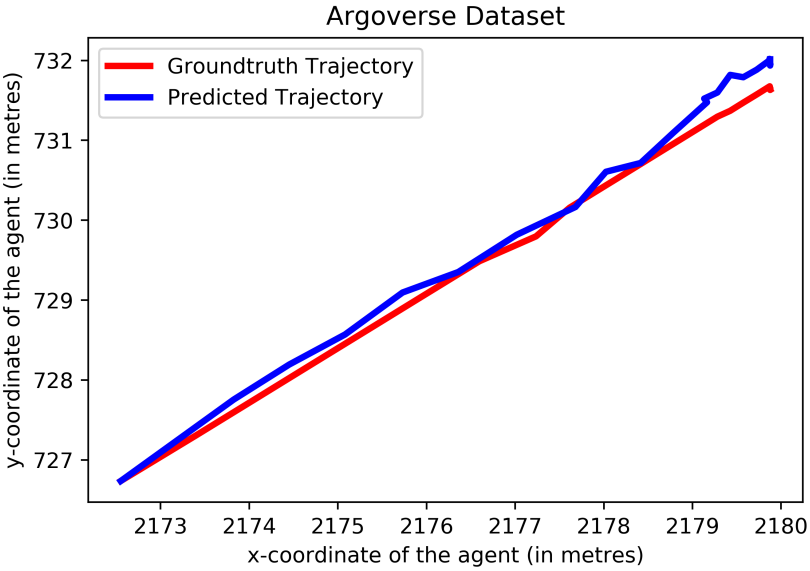

Figure 5.5 Groundtruth trajectory and predicted trajectory of the road-agent with *agent_ID = 11* of the Argoverse Dataset

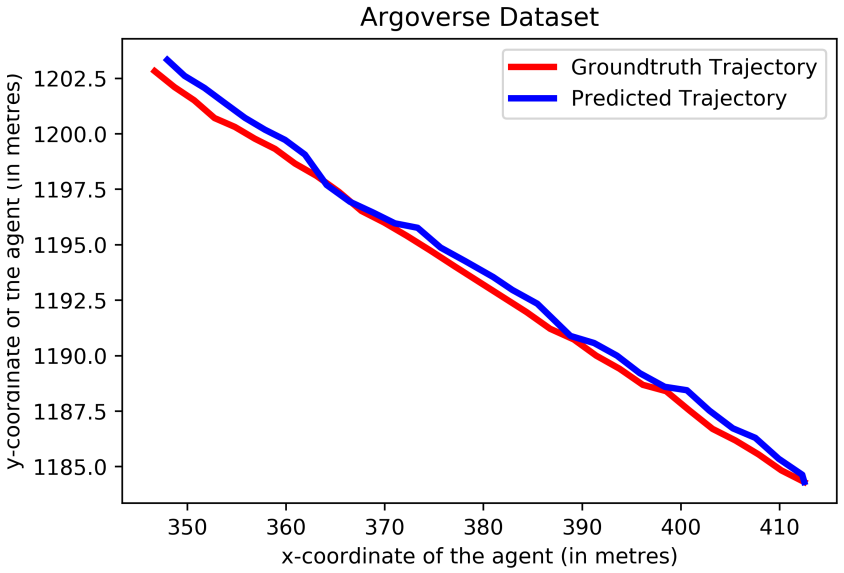

Figure 5.6 Groundtruth trajectory and predicted trajectory of the road-agent with *agent_ID = 27* of the Argoverse Dataset

5.2 Evaluation Metrics

For evaluating the accuracy of the trajectory predicted by the models, the metrics used are:

- **Average Displacement Error (ADE):** The root mean square error (RMSE) of all the predicted positions and real positions during the prediction window, i.e., the mean Euclidean distance over all the predicted positions and the ground truth positions during the prediction time. The ADE for the i^{th} object or, road-agent is given by:

$$ADE_i = \frac{1}{t_f} \sum_{t=1}^{t_f} \sqrt{(x^t_{i(pred)} - x^t_{i(GT)})^2 + (y^t_{i(pred)} - y^t_{i(GT)})^2},$$

  where, t_f is the number of time-steps or, frames in predicted sequence, $x^t_{i(pred)}$ and $y^t_{i(pred)}$ are the predicted x and y-coordinates of the i^{th} object at time t, and, $x^t_{i(GT)}$ and $y^t_{i(GT)}$ are the x and y-coordinates corresponding to the ground truth position of the object at time t.

  The overall ADE is the mean of the ADEs of all the concerned objects for which trajectories are predicted.

- **Final Displacement Error (FDE):** The RMSE distance between the final predicted positions at the end of the predicted trajectory and the corresponding true location, i.e., the mean Euclidean distance between the final predicted positions and the corresponding ground truth locations. The FDE for the i^{th} object or, road-agent is given by:

$$FDE_i = \sqrt{(x^{t_f}_{i(pred)} - x^{t_f}_{i(GT)})^2 + (y^{t_f}_{i(pred)} - y^{t_f}_{i(GT)})^2},$$

  where, t_f is the last time-step or, last frame in the predicted sequence, $x^{t_f}_{i(pred)}$

and $y^{t_f}_{i(pred)}$ are the predicted x and y-coordinates of the i^{th} object during the last time-step t_f, and, $x^{t_f}_{i(GT)}$ and $y^{t_f}_{i(GT)}$ are the x and y-coordinates corresponding to the ground truth position of the object at time-step t_f.

The overall FDE is the mean of the FDEs of all the concerned objects for which trajectories are predicted. *The FDE metric shows the long-term prediction ability of the model.*

5.3 Final Results and Comparison

Table 5.1 below summarizes the values of the metrics obtained for three datasets using our model and the other existing models on these datasets viz. Graph LSTM (stream-1) [1], GRIP [14], CS-LSTM [11], TraPHic [12], and Social-GAN [13].

Datasets →	**Apolloscape** (Pred. interval=3 sec.)		**Lyft** (Pred. interval=5 sec.)		**Argoverse** (Pred. interval=5 sec.)	
Models ↓	**ADE**	**FDE**	**ADE**	**FDE**	**ADE**	**FDE**
Our Model	**1.98**	**2.01**	**2.41**	**2.59**	**1.08**	**1.76**
Graph LSTM (S1)	3.59	3.71	5.32	5.79	2.40	3.09
GRIP	1.25	2.34	3.04	3.21	-	-
CS-LSTM	2.144	11.699	4.423	8.640	1.050	3.085
TraPHic	1.283	11.674	5.031	9.882	1.039	3.079
Social-GAN	3.980	6.750	7.860	14.340	3.610	5.390

Table 5.1 Final Results

Analysis of Results

The percentage of decrease of the errors (compared to the other existing models) is more in the sparse datasets like, Lyft and Argoverse dataset. It is because the average density (average no. of objects in each time-frame) is less in these datasets, which implies that owing to less traffic density, the objects will be more likely to follow their

own movement pattern than being influenced by other road-agents. This is less likely in datasets like Apolloscape, due to greater density.

The reduction in error is more for FDE compared to ADE, as in case of ADE the pattern was still captured in the initial time-steps, but for FDE it was diminished at the final time-steps of prediction. With the addition of the Temporal Convolution layers, the pattern being highlighted initially, the diminishing effect is certainly reduced. *The reduced FDE shows the better long-term prediction ability of our model.*

5.4 Future Work

The ways to improve the accuracy of our trajectory prediction model for the dense datasets can be explored. Also, the use of relative measurement of coordinates of the dynamic agents around by considering an ego-vehicle-based coordinate system can help to improve the prediction accuracy and this problem can be worked upon.

As an extension to this work, prediction of behaviors of the road-agents can also be done, which will help to forecast whether a road-agent is going to exhibit overspeeding, underspeeding or neutral behaviour.

Bibliography

[1] R. Chandra, T. Guan, S. Panuganti, T. Mittal, U. Bhattacharya, A. Bera, and D. Manocha, "Forecasting trajectory and behavior of road-agents using spectral clustering in graph-lstms," *IEEE Robotics and Automation Letters*, 2020.

[2] L. Paparusso, S. Melzi, and F. Braghin, "A deep-learning framework to predict the dynamics of a human-driven vehicle based on the road geometry," 2021.

[3] S. H. Park, G. Lee, M. Bhat, J. Seo, M. Kang, J. Francis, A. R. Jadhav, P. P. Liang, and L.-P. Morency, "Diverse and admissible trajectory forecasting through multimodal context understanding," 2020.

[4] H. Cheng, W. Liao, X. Tang, M. Y. Yang, M. Sester, and B. Rosenhahn, "Exploring dynamic context for multi-path trajectory prediction," 2021.

[5] O. Styles, T. Guha, V. Sanchez, and A. Kot, "Multi-camera trajectory forecasting: Pedestrian trajectory prediction in a network of cameras," 2020.

[6] Y. Liu, Q. Yan, and A. Alahi, "Social nce: Contrastive learning of socially-aware motion representations," 2021.

[7] B. Ivanovic, A. Elhafsi, G. Rosman, A. Gaidon, and M. Pavone, "Mats: An interpretable trajectory forecasting representation for planning and control," 2021.

[8] J. Liang, L. Jiang, and A. Hauptmann, "Simaug: Learning robust representations from simulation for trajectory prediction," 2020.

[9] X. Weng, Y. Yuan, and K. Kitani, "Ptp: Parallelized tracking and prediction with graph neural networks and diversity sampling," 2021.

[10] C. Graber and A. G. Schwing, "Dynamic neural relational inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[11] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," 2018.

[12] R. Chandra, U. Bhattacharya, A. Bera, and D. Manocha, "Traphic: Trajectory prediction in dense and heterogeneous traffic using weighted interactions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8483–8492.

[13] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks," 06 2018, pp. 2255–2264.

[14] X. Li, X. Ying, and M. C. Chuah, "Grip: Graph-based interaction-aware trajectory prediction," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 3960–3966.

[15] Y. Ma, X. Zhu, S. Zhang, R. Yang, W. Wang, and D. Manocha, "Trafficpredict: Trajectory prediction for heterogeneous traffic-agents," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 6120–6127, 07 2019.

[16] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet, "Lyft level 5 perception dataset 2020," https://level5.lyft.com/dataset/, 2019.

[17] M.-F. Chang, J. W. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, "Argoverse: 3d tracking and forecasting with rich maps," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[18] X. Li, X. Ying, and M. C. Chuah, "Grip++: Enhanced graph-based interaction-aware trajectory prediction for autonomous driving," *arXiv preprint arXiv:1907.07792*, 2020.