Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Statistics & Machine Learning
2022 | LIU-IDA/STAT-EX-A--2022/001--SE

AI-based prediction of road users' intents and reactions

–

Akshay Gurudath

Supervisor : Per Sidén
Examiner : Jose M Peña

External supervisor : Yury Tarakanov

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Different road users follow different behaviors and intentions in the trajectories that they traverse. Predicting the intent of these road users at intersections would not only help increase the comfort of drive in autonomous vehicles, but also help detect potential accidents. In this thesis, the research objective is to build models that predicts future positions of road users (pedestrians,cyclists and autonomous shuttles) by capturing behaviors endemic to different road users. Firstly, a constant velocity state space model is used as a benchmark for intent prediction, with a fresh approach to estimate parameters from the data through the EM algorithm. Then, a neural network based LSTM sequence modeling architecture is used to better capture the dynamics of road user movement and their dependence on the spatial area. Inspired by the recent success of transformers and attention in text mining, we then propose a mechanism to capture the road users' social behavior amongst their neighbors. To achieve this, past trajectories of different road users are forward propagated through the LSTM network to obtain representative feature vectors for each road users' behaviour. These feature vectors are then passed through an attention-layer to obtain representations that incorporate information from other road users' feature vectors, which are in-turn used to predict future positions for every road user in the frame. It is seen that the attention based LSTM model slightly outperforms the plain LSTM models, while both substantially outperform the constant velocity model. A comparative qualitative analysis is performed to assess the behaviors that are captured/missed by the different models. The thesis concludes with a dissection of the behaviors captured by the attention module.

Acknowledgments

Firstly, I would like to thank Linköping University for giving me an opportunity to pursue this Master's program and thesis which have both taken me on a beautiful journey.

I would like to sincerely thank my university supervisor Per Sidén for guiding me ably and splendidly throughout this thesis. He gave me fresh perspectives and many ideas presented in this paper are a result of insightful discussions with him. After these discussions, I was encouraged to renew my work with fresh vigor, and try out the latest idea, which in the long term inevitably led to progress.

I am also extremely grateful to my external supervisor Yury Tarakanov from Viscando and want to thank him wholeheartedly for always being there and supporting me constantly throughout this project. His enthusiasm was unparalleled and his work ethic impeccable. He inspired me to work hard and infused a sense of passion in me towards this thesis project.

Next, I would like to thank Professor Jose M Peña for giving me good inputs and taking time out for examining my thesis, in an effort to make it better. In the same vein, I would also thank Sreenand Sasikumar for opposing my thesis and giving feedback. I would also like to thank Professor Oleg Sysoev for coordinating this thesis course and also multiple other past courses in the STIMA program.

My greatest source of support comes from my family and friends. I would like to thank my family for their unconditional love and support (Amma, Appa and Shreyas: thanks wouldn't be enough). Keshav, Samia and Nikil were (and continue to be) great support systems during my time in Sweden and I profusely thank them for bearing with me and also pushing me to be better in their own different ways. I also want to thank Shashi for his good guidance and peer-motivating me through his own example to strive for perfection in work. Lastly, I would be remiss without thanking Jubin, Deepak, Kavya, Aashay, Daya, Vaishakh, SKS, Mubasir (and all *Locos*) for being great pillars (and continue to be) through the years, always wishing me the best and relishing my success as if theirs.

Contents

List of Figures

List of Tables

1 Introduction

Traffic safety is a big concern because of increasing mobility needs and large numbers of casualties in traffic with approximately 1.3 million killed each year[38]. Accidents often happen as results of misunderstandings and improper reactions in interactions between road users. This is especially true in the case of urban traffic, where vulnerable road users (pedestrians and cyclists) are at high risk of harm in collisions with vehicles. Autonomous vehicles (AV) are envisioned to improve traffic safety, because of their faster reactions and full compliance to traffic rules. Despite ambitious promises, and intensive test activities, the first autonomous vehicle prototypes without human back-up driver are allowed to operate in very limited areas and traffic conditions in the latest months [40], and there is a lot of skepticism about the remaining work until autonomous vehicles will be able to truly safely operate in any traffic conditions. One of the main reasons for these challenges is the concern about the safety of interaction of AVs with other road users, especially vulnerable ones. Like an experienced driver, AV needs to understand their behavior, intentions and reactions to its maneuvers early and plan its motion accordingly. This is to both not provoke unsafe, dangerous reactions by unexpected maneuvers, and to proactively minimize accident risks for possible future evolution of traffic situations. Also in the case of slow traffic situations, an AV can plan maneuvering actions so that it doesn't need to unnecessarily stop and halt at many places. An AV can better plan its own trajectory if it is able to model the intent of road users and predict the behavior of different road users in an environment. Also, in general if we were able to predict intent of different road users in a system, we could develop warning systems and indicators at intersections to prevent risks of collisions and accidents.

1.1 Objective

Through this thesis, we look at developing such an intent prediction framework based on trajectory data of different road users. In this thesis, we will be working on data from infrastructure sensors that are able to record traffic movements in a given road segment during long times with high accuracy. The chief dataset is collected by Viscando AB at the Linköping University campus. This dataset specifically consists of trajectories of various road users including pedestrians, cyclists and an autonomous shuttle over an area of roughly 1300 square meters. The objective of this thesis is to explore models for the concerned data and construct a robust baseline/benchmark model along with a suitable final model. It will also be of great

interest to understand behaviors of different road users and where the models are able to/ not able to capture information. Specifically, we are interested in achieving these research objectives:

- Study relevant methods and design a suitable baseline model for intent prediction, which learns parameters from the data and is interpretable as well as robust.
- Design a more complex model that better captures dynamics of road user movement and also factors in social behaviors of road users in traffic.
- Compare and assess these models and understand what behaviors are captured/not captured by each of these models.

1.2 Contributions

In our work, we start by exploring the dataset and making suitable pre-processing to the raw dataset. Constant velocity[26] and acceleration models are very popular in the object/vehicle tracking literature and they are also considered as a good baseline for intent prediction. Therefore, a baseline constant velocity model is developed but with a significant change as compared to previous work in the domain of object tracking. In our work, we estimate the parameters of the constant velocity model through data for each road user, instead of heuristically assigning values to the parameters. This is achieved through the Expectation-Maximization algorithm[33]. Through uncertainty estimates, we measure the difference between the predicted distribution and the actual trajectory. After setting up the baselines and error estimates, we move on to construct a vanilla long short term memory (LSTM) model [10] to better model the dynamics of a road user's trajectory and capture location specific movements from the training data. This is then built up to a novel pooled LSTM model to account for social dependence between other road users. For this, we use a self-attention[37] based architecture to pool and weigh the hidden layer outputs of several LSTM networks, so that predictions are made for every road user in the frame. We then conclude our work by assessing where the models fail in predicting behavior and also try to gauge what behaviors are successfully captured by the models.

1.3 Uniqueness

There has been extensive research done on intent prediction in the past [24][27][1][29] . However, many of these studies lack large objective datasets on the interaction of road users with autonomous agents intended to operate in spaces shared with pedestrians and cyclists, such as autonomous shuttles or small delivery vehicles. There are three notable differences between a majority of the past research and our work:

- Many of these research studies deal with open source datasets with shorter trajectories, as on-board sensors of moving vehicles are used. On the other hand, our data is acquired through a stationary infrastructure sensor which collects sufficiently long trajectories for each road user over long time periods, days to years.
- Also, many of these experiments [24] [17] concern themselves with image data. In our case the sensor is already equipped with an AI system that tracks and identifies distinct road users. Therefore, we work with position data of each road user in time.
- Most studies[24][27][1][29] also deal with one kind of road user. For example, some of them model vehicles on a highway, whereas others model pedestrians in a crowd. In this paper, we have interactions between three kinds of road users: cyclists, pedestrians and shuttles and this has seldom been modeled.

- A novel method for intent prediction is proposed that uses a self-attention mechanism on past trajectories of road users in a scene, to predict the future positions for all road users in the scene.

2 Related Work

In this section, relevant literature is briefly presented and ideas from these papers are discussed. Also, the context of our research is highlighted , where the similarities as well as differences between past papers are discussed.

Methods related to human pose detection and intent prediction have been well explored in the past. Recently, there have been advancements in the field of human pose detection and the literature is substantially rich ([9],[7]). Human pose detection is also an important feature to further predict action/intent of a pedestrian. It has been shown that human pose is important to measure distance between humans, and also if two pedestrians are making eye-contact or not ([3]). Pose estimation techniques are implemented by [24] to construct intermediate features which are later used to determine the crossing behavior of pedestrians. For our problem, we are more focused on intent prediction methods without considering pose, as the dataset consists of trajectories and not of images. It is also of further interest to see if models that only rely on position data, perform well without the use of richer feature information such as pose/gestures.

2.1 Classification problems

There are two broad divisions of study in intent prediction, amongst which the first looks at classifying intent into separate classes. For example, a binary prediction is enough when predicting whether a pedestrian is going to cross the road or not. A person is detected to be bending, starting or stopping based on silhouettes gathered from stereo-vision data in [17]. On the other hand, [16] tries to predict if a person is going to the cross the road based on augmented features derived from a dense optical flow. A procedure based on graph convolution is used in [19] to extract spatiotemporal relationships between other objects in the frame and previous time steps, so that they are useful in assessing whether a pedestrian will cross the street or not. In the trajectory data present for our case, there aren't too many turns or chances for collisions. For this reason, our area of focus is chiefly on trajectory prediction, where instead of binary/multiple classes, a path is predicted for pedestrians.

2.2 Trajectory prediction problems

Explicit modeling methods

The first class of solutions in trajectory prediction problems pertain to explicit modeling methods, where the pedestrians/vehicles are modeled based on the laws of physics . Constant velocity models, constant acceleration, constant turn models incorporate basic laws of kinematics to predict future positions. There are also more complex non-linear models such as the extended Kalman filter[14], unscented Kalman filter[36] which can instead be used to model the dynamics of pedestrian movement. An interacting multiple model[21] is used in [41] that combines two such physics-based models and adjusts model parameters to improve its predictive accuracy. A comparative study across various such filter-based models is performed by [30] , where path prediction accuracy is assessed. In our work, the Kalman filter constant velocity model is used as the baseline owing to its simple yet robust design[31]. However, unlike normal implementation where model parameters are chosen heuristically, in this thesis, parameters are estimated from the data. The filter-based approaches are successful in modeling short range predictions ($<$ 1 second), but fail when predicting over longer time horizons [20]. There are certain methods such as ([12],[20]) which attempt to model long-term predictions using concepts of particle filters, dynamic bayesian networks and gaussian processes. However, these methods were implemented on a highway dataset for predicting lane changes, where the vehicles move in a more deterministic fashion than pedestrians at intersections.

Planning based approaches

There are also related planning-based approaches through which the end-goal/general intent of a pedestrian is inferred and the path of a pedestrian is "planned" based on this end goal. For example, this end-goal is modelled as a hidden state in [15] and the intent is modeled as a stochastic policy conditioned on the end-goal. An innovative way is tried in [39] for assigning trajectories to certain clusters of a Gaussian mixture model through which intent is modelled. This modeled intent then determines the path traversed by a pedestrian. A novel way of combining this so-called planning based approach and path prediction approach is used in [29]. They do so by using an inverse-reinforcement learning network to learn the reward function and later use this reward function as an input to a sequence modeling network (such as a recurrent neural network [22]) that predicts future trajectories. These solutions require modeling of a goal which then influence the path being planned. However, it would be convenient to avoid this two-step process and directly model both the path and the goal/intent.

Sequence learning based architectures

The last class of solutions of particular interest to us involve sequence-learning based architectures to model the temporal nature of trajectories and some of them also try to account for the social interactions between pedestrians. A long short term memory (LSTM) network[11] is used in [28] to preserve patterns in the long past trajectories of pedestrians that is otherwise not preserved in non-gated architectures like the recurrent neural networks (RNN)[22]. Through an architecture of dual LSTM networks, trajectories were predicted for all surrounding vehicles in [42]. A spatiotemporal LSTM was constructed by [5] that measured not only temporal patterns but also the spatial interactions between multiple vehicles. They also modified the LSTM network so as to limit the vanishing gradient problem. In recent past, [1] has been regarded as a seminal paper that combined the success of an LSTM network and integrated social modeling through a pooling layer. An LSTM network was used to model the plain vanilla behavior of the ego user and his/her neighbors. The hidden states of these

neighbors were combined based on their proximity to the ego user and this combined hidden layer was then used for prediction.

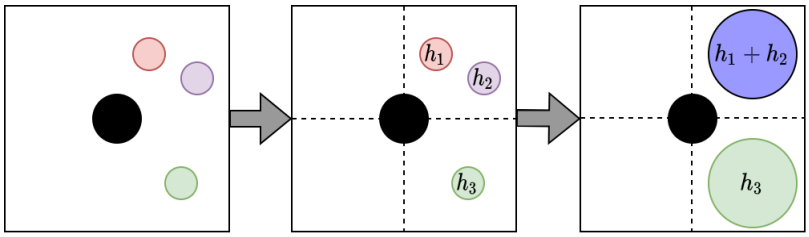


Figure 2.1: Pooling in the Social LSTM

In the above figure, the black dot concerns the ego-road user and the other dots signify the neighbors. We see that the red and purple hidden states are added as they belong to a specific region. This kind of pooling creates a tensor that partially preserves social information of the different neighbors. An excellent review of other deep learning approaches for trajectory prediction is present in [35].

Attention

In the above mentioned deep-learning based frameworks, trajectory prediction is generally considered as a sequence to sequence task, where the past trajectory sequence is used to predict the future trajectory. A novel idea of using attention coupled with a RNN was used by [2] to improve performance in a sequence to sequence sentence translation task. Attention selectively focused on certain parts of the input sentence to help generate the translated sequence. The idea of self-attention was made popular by [37], where the attention mechanism relates to different positions of the *same* sequence in order to obtain a representation of that sequence. It is to be noted that both [37],[2] use the concept of soft attention. According to [32], soft attention models take a weighted average of the entire input space and thus are deterministic and can be trained through backpropagation. Hard attention on the other hand selectively focuses on an approximate area of the input with the help of attention scores (for example by sampling based on the attention score, only certain discrete areas of the input can be obtained). A more detailed technical review of soft-attention has been presented in Section 3.6. After the large success of attention mechanisms in the natural language processing domain, attention is beginning to be used in other domains as well. Attention was used to selectively focus on certain areas of an image to caption them in [43]. Temporal attention has already been used to improve trajectory prediction, to focus selectively on some parts of the past trajectory. Beyond just focusing on some parts of a sequence, the concept of attention can be used instead to focus on certain neighbors of a ego road user, to in-turn incorporate their features in determining the trajectory of the ego-user. Soft-attention was used in [8] to model the temporal nature of the trajectory and hard-attention was utilized for attending to neighbors, where attention weights are calculated based on the distance between road users. However, the importance/attention that road users give each other depend on factors other than distance. For example, a road user would mostly be concerned with someone coming in the opposite direction, rather than someone in the same lane. It would be more natural for a model to learn these dependencies rather than manually encode these behaviours.

Methods used in this thesis

In this thesis, we would like to propose a novel soft self-attention approach to model the relationships between various road users and predict future positions of all these road users. The individual past trajectories of road users are encoded through an LSTM network, and then attention is applied to the encoded feature vectors of all road users (social self attention)

in the scene. As a result, the output feature vectors of each road user contain information from other road users in the same scene. We hypothesize that this would lead to better trajectory predictions for all road users in a scene. In conclusion, there are 3 models built as follows:

- A dynamic constant velocity model, where the novelty is that parameters are estimated from the data instead of being set heuristically.
- A plain robust LSTM network that uses the full length past trajectory, to model the future positions of road users.
- A novel LSTM network combined with an attention encoder to predict future positions of all road users by baking in social dependencies between different road users.

All of these methods are elaborated upon in further sections.

3 Theory

This section will elaborate and build upon all the theoretical foundations required to implement most of the methods presented in this paper. Firstly, we will look at defining Kalman filter models and some properties related to these models. Then, we focus on the constant velocity model and derive the expectation maximization algorithm in order to estimate parameters from the data. We then shift our attention to recurrent neural network based architectures that can be implemented on time series data. Specifically, the scope of long short term memory models, transformer models on our problem and dataset is studied in more detail and depth.

3.1 Kalman Filter

State space models represent a group of probabilistic graphical models that describe the dependence between a hidden state variable and an observed state variable. The hidden state consists of information that is hidden/latent as opposed to an observed state variable. As an example, one can think of noisy measurements of object position from a sensor as the observed state, whereas the actual object position can be considered as a hidden state. These models that incorporate the hidden state and observed states are very useful as they capture long range temporal dependencies between observations, while maintaining sparsity.

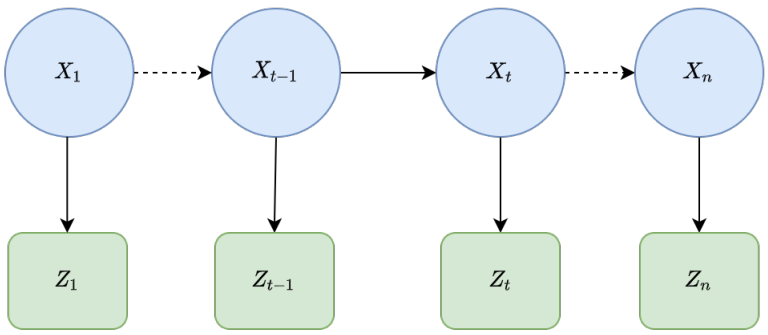


Figure 3.1: An illustration of the state space model

In the Figure 3.1, X_t represents the latent/hidden state space variable. X_t is independent of $X_{0:t-2}$ given X_{t-1} showing us that the hidden state exhibits Markovian properties. This essentially means that the future hidden states only depend on the present hidden state and does not depend on past history. On the other hand, the measured variable Z_t is independent of $Z_{0:t-1}$ given X_t . The model can entirely described using the following two equations:

$$\begin{aligned} X_t &= AX_{t-1} + \eta_t \\ Z_t &= CX_t + \epsilon_t \end{aligned} \tag{3.1}$$

where η_t and ϵ_t are distributed as $N(0, Q)$ and $N(0, R)$ respectively. Here A and C govern the dynamics of the state space transition and the emission between state space and observed state respectively. The covariance matrices Q and R represent the state space transition covariance matrix and the emission state covariance matrix.

Estimators of the hidden state

In the Kalman filter, the primary focus is to obtain estimates of the hidden state X_t given the observations $Y_{1:n}$ upto time n. The estimation of the hidden state X_t at the current time step given data till the current time step Y_t is called as the filtering problem. This can be considered as when $n = t$. When $n > t$, the problem is called smoothing, where estimates of X_t are estimated given observations in the future. Lastly, when $n < t$ the problem now pertains to prediction as estimate of X_t is obtained only using the observations before t. The expectation of X_t given observations $Y_{1:n}$ can be written as:

$$X_t^n = E(X_t|Y_n) \tag{3.2}$$

Apart from measuring just these estimates, the Kalman filter by construction gives us good ways to measure the precision/uncertainty of estimates. For example the covariance P_t^n can be defined as :

$$P_t^n = E((X_t - X_t^n)(X_t - X_t^n)') \tag{3.3}$$

The covariance between the estimates at different time steps t_1 and t_2 can be written as:

$$P_{t_1,t_2}^n = E((X_t - X_{t_1}^n)(X_t - X_{t_2}^n)') \tag{3.4}$$

Consequently, the one-lag covariance smoother, which is nothing but covariance between consecutive estimates of the hidden state, can be defined as:

$$P_{t,t-1}^n = E((X_t - X_t^n)(X_{t-1} - X_{t-1}^n)') \tag{3.5}$$

The means and covariances defined above give us a good idea about the moments (or the characteristics) of the distribution of X_t which will prove to be useful when information regarding X_t is required.

Filtering and Prediction

In the previous section, the filtered, predicted and smoothed estimates were defined, but not calculated. In order to calculate these estimates of means and covariances, there are well defined filtering and smoothing algorithms. In this section and the next, relevant algorithms are presented to compute the estimates defined in the previous section. The proofs for these algorithms and properties are presented in detail in [34].

For a state space model defined in Eq. 3.1, assuming the initial conditions as $X_0^0 = \mu_0$ and $P_0^0 = \Sigma_0$, for $t = 1, 2...n$

$$\begin{aligned} X_t^{t-1} &= AX_{t-1}^{t-1} \\ P_t^{t-1} &= AP_{t-1}^{t-1}A' + Q \end{aligned} \tag{3.6}$$

with.

$$\begin{aligned} X_t^t &= X_t^{t-1} + K_t(Z_t - CX_t^{t-1}) \\ P_t^t &= [I - K_t C]P_t^{t-1} \end{aligned} \tag{3.7}$$

where

$$K_t = P_t^{t-1}C'[CP_t^{t-1}C' + R]^{-1} \tag{3.8}$$

Here K_t is called as the Kalman gain. This decides the weight that needs to be placed on the measurements relative to the previous state in deciding the next state. If the measurements are accurate, then more weight is placed on the current measurement. When the state is known accurately, the previous state is given importance to predict the next. Starting from X_0^0 and P_0^0, we can compute the one-step predictive estimates X_1^0 and P_1^0 using Eq. 3.6. Using Eq. 3.7 and Eq. 3.8, the filter estimates X_1^1 and P_1^1 can then be computed. This entire process can be iteratively repeated until X_n^n and P_n^n are computed. For the case of prediction beyond $t > n$, Eq. 3.6 can be used to compute X_{n+1}^n and P_{n+1}^n. The filter distribution X_{n+1}^{n+1} and P_{n+1}^{n+1} can then be assumed to be the same as the predictive distribution. In this way, multi-step ahead prediction would be similar to iteratively stacking multiple one-step predictions. For more detailed proofs regarding Eq. 3.6, Eq. 3.7 and Eq. 3.8 please refer to [34].

Smoothed Estimates

For the model specified in Eq. 3.1 with initial conditions X_n^n and P_n^n obtained through Eq. 3.6, Eq. 3.7 and Eq. 3.8, for $t = n, n-1, ...1$,

$$\begin{aligned} X_{t-1}^n &= X_{t-1}^{t-1} + J_{t-1}(X_t^n - X_t^{t-1}), \\ P_{t-1}^n &= P_{t-1}^{t-1} + J_{t-1}(P_t^n - P_t^{t-1})J_{t-1}', \end{aligned} \tag{3.9}$$

where

$$J_{t-1} = P_{t-1}^{t-1}A'[P_t^{t-1}]^{-1} \tag{3.10}$$

Starting from $t = n$, the smoothed estimates can be computed backwards using the filter and predicted estimates obtained in the filtering step. Also the lagged one-step covariance estimates defined in Eq. 3.5 can be obtained through a similar algorithm. Assuming K_t,P_n^n, J_t are defined from the filtering, smoothing steps and with initial condition

$$P_{n,n-1}^n = (I - K_n C)AP_{n-1}^{n-1} \tag{3.11}$$

,the lagged one-step covariance estimates can be obtained for $t = n, n-1, ...2$

$$P_{t-1,t-2}^n = P_{t-1}^{t-1}J_{t-2}' + J_{t-1}(P_{t,t-1}^n - AP_{t-1}^{t-1})J_{t-2}' \tag{3.12}$$

Again, the proof for all of these properties can be found in [34].

3.2 Constant Velocity Model

In the 1 dimensional constant velocity (CV) model, the latent positions and velocities can be described as follows:

$$\begin{aligned} x_t &= x_{t-1} + \dot{x}_{t-1}\Delta t + \eta_t' \\ \dot{x}_t &= \dot{x}_{t-1} + \eta_t'' \end{aligned} \tag{3.13}$$

As can be seen, the current position at time t (represented by x_t) when added to the distance that the object has traversed in Δt timesteps, can give a good indication of the object's final

position. Here, the assumption made is that the object is moving at constant velocity between time step $t-1$ and t, giving rise to the name of this model. In real life, we can assume some noise that offsets the position. Also it can be seen that velocity at time step t in addition to some random noise, gives the velocity at time $t+1$. The mean velocity is constant from time step $t-1$ to t. Therefore if $\dot{x}_t$ is known at time t, then $E[\dot{x}_{t+1}], E[\dot{x}_{t+2}], ..., E[\dot{x}_n] = \dot{x}_t$.

$$Z_{x_t} = x_t + \epsilon_t \tag{3.14}$$

The final position measurement at time t (represented by Z_{x_t}) is the same as the actual position offset by some measurement noise. Since, the data consists of two coordinates, the same model can be rewritten in a slightly different way as below:

$$\begin{aligned} X_t &= AX_{t-1} + \eta_t \\ Z_t &= CX_t + \epsilon_t \end{aligned} \tag{3.15}$$

where

$$X_t = \begin{pmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{pmatrix}, A = \begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The state space consists of the x-coordinate, velocity along the x-axis, y-coordinate and velocity along the y-axis. The matrix A is obtained from the two dimensional form of Eq. 3.13 and it governs the state space transition.

$$Z = \begin{pmatrix} Z_{x_t} \\ Z_{y_t} \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The measured state consists of the observed x and y-coordinate with emission between state and observation being governed by C. Also η and ϵ are normally distributed with 0 mean and covariance matrices Q, R respectively.

A heuristic approach to choose the transition and emission covariance

If we assume that the random noise is a random acceleration, then through laws of kinematics [13], we can approximate the next state $x_t, \dot{x}_t, y_t, \dot{y}_t$ as follows:

$$\begin{aligned} x_t &= x_{t-1} + \dot{x}_{t-1}\Delta t + \frac{\Delta t^2}{2}\eta_{x,t-1} \\ y_t &= y_{t-1} + \dot{y}_{t-1}\Delta t + \frac{\Delta t^2}{2}\eta_{y,t-1} \\ \dot{x}_t &= \dot{x}_{t-1} + \Delta t\eta_{x,t-1} \\ \dot{y}_t &= \dot{y}_{t-1} + \Delta t\eta_{y,t-1} \end{aligned} \tag{3.16}$$

Here $\eta_{x,t-1}$ and $\eta_{y,t-1}$ are considered to be random noisy accelerations. This can be written in the form of Eq. 3.15 where

$$\eta_t = \begin{pmatrix} \frac{\Delta t^2}{2}\eta_{x,t} \\ \eta_{x,t} \\ \frac{\Delta t^2}{2}\eta_{y,t} \\ \eta_{y,t} \end{pmatrix}$$

Since $\eta_t \sim N(0, Q)$, the covariance of the noise matrix Q can be written as

$$\begin{aligned} Q &= E(\eta_t \eta_t^T) \\ &= E(\begin{pmatrix} \frac{\Delta t^2}{2}\eta_{x,t} \\ \eta_{x,t} \\ \frac{\Delta t^2}{2}\eta_{y,t} \\ \eta_{y,t} \end{pmatrix} \begin{pmatrix} \frac{\Delta t^2}{2}\eta_{x,t} & \eta_{x,t} & \frac{\Delta t^2}{2}\eta_{y,t} & \eta_{y,t} \end{pmatrix}) \\ &= \begin{pmatrix} \frac{\Delta t^4}{4}\sigma_{\eta_x}^2 & \frac{\Delta t^2}{2}\sigma_{\eta_x}^2 & 0 & 0 \\ \frac{\Delta t^2}{2}\sigma_{\eta_x}^2 & \sigma_{\eta_x}^2 & 0 & 0 \\ 0 & 0 & \frac{\Delta t^4}{4}\sigma_{\eta_y}^2 & \frac{\Delta t^2}{2}\sigma_{\eta_y}^2 \\ 0 & 0 & \frac{\Delta t^2}{2}\sigma_{\eta_y}^2 & \sigma_{\eta_y}^2 \end{pmatrix} \end{aligned} \tag{3.17}$$

In the above equation, $E(\eta_{x,t}^2), E(\eta_{y,t}^2)$ is substituted with $\sigma_{\eta_x}^2$ and $\sigma_{\eta_y}^2$. Further, assuming that $\sigma_{\eta_x}^2 = \sigma_{\eta_y}^2 = \sigma_\eta^2$, we get the following expression for Q

$$Q = \sigma_\eta^2 \begin{pmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^2}{2} & 0 & 0 \\ \frac{\Delta t^2}{2} & 1 & 0 & 0 \\ 0 & 0 & \frac{\Delta t^4}{4} & \frac{\Delta t^2}{2} \\ 0 & 0 & \frac{\Delta t^2}{2} & 1 \end{pmatrix} \tag{3.18}$$

If it is assumed that measurement noise in one direction is independent of the measurement noise in another direction, the covariance matrix R can be written as:

$$R = \begin{pmatrix} \sigma_{\epsilon x}^2 & 0 \\ 0 & \sigma_{\epsilon y}^2 \end{pmatrix} \tag{3.19}$$

It is a design choice to choose appropriate $\sigma_\eta^2, \sigma_{\epsilon x}^2, \sigma_{\epsilon y}^2$. Once these parameters are chosen, all the parameters in the model represented by Eq. 3.16 are known.

However, in this paper, we aim to estimate parameters such as Q, R with the help of data and not by assigning heuristic values as in Eq. 3.18 and Eq. 3.19. We thus disregard the assumption that noise affects the dynamics of motion in a specific way as in Eq. 3.16. This gives us more flexibility in our design to choose the best parameters through data-centric methods. The best parameters can be obtained by maximizing the likelihood of observed data with respect to the parameters. This is elaborated in further detail in the next section.

3.3 Single trajectory data likelihood and the need for the EM algorithm

As mentioned in the previous section, we aim to maximize the likelihood of data and in-turn obtain the maximum likelihood parameters. Let us say that the trajectory length was n, the hidden states were $X_0, X_1, X_2 ... X_n$ and the observations were $Z_1, Z_2, Z_3 ..., Z_n$ with the model being described as:

$$\begin{aligned} X_t &= AX_{t-1} + \eta_t \\ Z_t &= CX_t + \epsilon_t \end{aligned} \tag{3.20}$$

It can be assumed that $X_0 \sim N(\mu_0, \Sigma_0)$ where

$$\mu_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$
$$\Sigma_0 = \begin{pmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{pmatrix} \tag{3.21}$$

The mean is a zero matrix with the covariance matrix having high uncertainty around the mean. Since all these distributions are normal, the complete data log-likelihood can be written as below:

$$\begin{aligned} -2\log p(X_{0:n}, Z_{1:n}) = {} & n\log|Q| + \sum_{i=1}^{n}(X_t - AX_{t-1})'Q^{-1}(X_t - AX_{t-1}) \\ & + n\log|R| + \sum_{i=1}^{n}(Z_t - CX_t)'R^{-1}(Z_t - CX_t) \\ & + \log|\Sigma_0| + (X_0 - \mu_0)'\Sigma_0^{-1}(X_0 - \mu_0) + constant \end{aligned} \tag{3.22}$$

As can be seen, the log-likelihood is dependent on not just the parameters A, C, Q and R, but also on the hidden states. These hidden states are latent and only the estimators of the hidden states can be calculated through properties mentioned in Sections 3.1 3.1. At this stage, ideas from the Expectation Maximization (EM) [6] algorithm can be used. EM algorithm is generally used in missing data scenarios, where the likelihood cannot be maximized because the data itself is missing. In our case, the hidden states can be considered to be missing data and the observed states can be considered as non-missing data. Assuming some model parameters, one can obtain estimators and moments for these hidden states/missing data which can in-turn be used to find the expected data log-likelihood given the current parameters and the observed data. Now, this expected data log-likelihood can be maximized with respect to new parameters and analytical update equations can be obtained for each parameter. This updated parameter can in-turn be used to calculate fresh estimators and moments for the missing data and the process can be repeated until convergence of parameters. It has been shown in [6] that the EM algorithm converges to the maximum likelihood estimate. Therefore, instead of calculating the data log-likelihood and maximizing it with respect to the parameters, we can instead use the two-step iterative Expectation Maximization algorithm that briefly entails the following:

- Given current values of parameters and the measured data, the expected complete data log-likelihood is calculated through a closed-form expression
- Maximize the expected data log-likelihood with respect to the new parameters and update each parameter. These new parameters are then used to again compute the expected data log-likelihood and this iteration continues until parameters converge

3.4 Expectation and Maximization step

The derivation for the EM step presented for the single trajectory case has been independently derived and is presented below. However, this derivation can also be found in [34]. The extension to multiple trajectory time-series in the next section has also been independently derived. This extension of the EM algorithm to multiple time-series sequences was not found in relevant literature to the best of our knowledge.

Single trajectory

The expected data log-likelihood for a single trajectory of length n can be written as follows, where ϕ_{j-1} represents the older parameters:

$$\begin{aligned}-2E[\log p(X_{0:n}, Z_{1:n})|\phi_{j-1}, Z_{1:n}] &= n\log|Q| + n\log|R| + \log|\Sigma_0| \\ &+ E[\sum_{i=1}^{n}(X_t - AX_{t-1})'Q^{-1}(X_t - AX_{t-1})|\phi_{j-1}, Z_{1:n}] \\ &+ E[\sum_{i=1}^{n}(Z_t - CX_t)'R^{-1}(Z_t - CX_t)|\phi_{j-1}, Z_{1:n}] \\ &+ E[(X_0 - \mu_0)'\Sigma_0^{-1}(X_0 - \mu_0)|\phi_{j-1}, Z_{1:n}] + constant\end{aligned} \tag{3.23}$$

It can be noticed that the matrix products of each individual term in the summation is a scalar and using the property that $A = trace(A)$ if A is scalar, one can write Eq. 3.23 as follows:

$$\begin{aligned}-2E[\log p(X_{0:n}, Z_{1:n})] &= n\log|Q| + n\log|R| + \log|\Sigma_0| \\ &+ E[tr(\sum_{i=1}^{n}(X_t - AX_{t-1})'Q^{-1}(X_t - AX_{t-1})] \\ &+ E[tr(\sum_{i=1}^{n}(Z_t - CX_t)'R^{-1}(Z_t - CX_t)] \\ &+ E[tr((X_0 - \mu_0)'\Sigma_0^{-1}(X_0 - \mu_0)] + constant\end{aligned} \tag{3.24}$$

All the expectations in 3.24 are conditioned given $\phi_{j-1}, Z_{1:n}$ and is not represented explicitly. Also using the properties that $trace(AB) = trace(BA)$ and $trace(A) + trace(B) = trace(A + B)$, the inverse of the covariance matrices can be brought outside the summation as below:

$$\begin{aligned}-2E[\log p(X_{0:n}, Z_{1:n})] &= n\log|Q| + n\log|R| + \log|\Sigma_0| \\ &+ tr(Q^{-1}E[\sum_{i=1}^{n}(X_t - AX_{t-1})(X_t - AX_{t-1})']) \\ &+ tr(R^{-1}E[\sum_{i=1}^{n}(Z_t - CX_t)(Z_t - CX_t)']) \\ &+ tr(\Sigma_0^{-1}E[(X_0 - \mu_0)(X_0 - \mu_0)']) + constant\end{aligned} \tag{3.25}$$

After simplifying, this can be written as:

$$\begin{aligned}-2E[\log p(X_{0:n}, Z_{1:n})] &= n\log|Q| + n\log|R| + \log|\Sigma_0| \\ &+ tr(Q^{-1}\sum_{i=1}^{n}(E(X_tX_t' - AX_{t-1}X_t' - X_tX_{t-1}'A' + AX_{t-1}X_{t-1}'A'))) \\ &+ tr(R^{-1}\sum_{i=1}^{n}(Z_tZ_t' - E[CX_tZ_t' + Z_tX_t'C' - CX_tX_t'C'])) \\ &+ tr(\Sigma_0^{-1}(\mu_0\mu_0' + E[X_0X_0' - \mu_0X_0' - X_0\mu_0'])) + constant\end{aligned} \tag{3.26}$$

The moments of hidden states at various $t = 1, 2, ...n$ needs to be computed given the entire observed data trajectory $(Z_1, Z_2...Z_n)$. This would correspond to estimating the smoothed moments of hidden states. From Eqs. 3.4 and 3.5, we can express $E(X_tX_t'), E(X_tX_{t-1})$ as:

$$\begin{aligned}E(X_tX_t') &= P_t^m + X_t^mX_t^{m\prime} \\ E(X_tX_{t-1}) &= P_{t,t-1}^m + X_t^m(X_{t-1}^m)'\end{aligned} \tag{3.27}$$

Substituting this in the equation 3.26 we get

$$\begin{aligned} -2E[\log p(X_{0:n}, Z_{1:n})] &= n\log|Q| + n\log|R| + \log|\Sigma_0| \\ &+ tr(Q^{-1}[S_{11} - AS'_{10} - S_{10}A' + AS_{00}A']) \\ &+ tr(R^{-1}(M_{ZZ} + M_{XZ} - M_{XX})) \\ &+ tr(\Sigma_0^{-1}(\mu_0\mu'_0 + E[X_0X'_0 - \mu_0X'_0 - X_0\mu'_0])) + constant \end{aligned} \tag{3.28}$$

where

$$\begin{aligned} S_{11} &= \sum_{t=1}^{n}(X_t^n(X_t^n)' + P_t^n) \\ S_{10} &= \sum_{t=1}^{n}(X_t^n(X_{t-1}^n)' + P_{t,t-1}^n) \\ S_{00} &= \sum_{t=1}^{n}(X_{t-1}^n(X_{t-1}^n)' + P_{t-1}^n) \end{aligned} \tag{3.29}$$

and

$$\begin{aligned} M_{ZZ} &= \sum_{i=1}^{n}(Z_tZ'_t) \\ M_{XZ} &= \sum_{i=1}^{n}Z_t(X_t^n)'C' - CX_t^nZ'_t \\ M_{XX} &= \sum_{i=1}^{n}CP_t^nC' + CX_t^n(X_t^n)'C' \end{aligned} \tag{3.30}$$

All the smoothed estimates $X_t^n, P_t^n, P_{t,t-1}^n$ can be obtained by following the filtering, smoothing steps in Section 3.1, 3.1. Once these smoothing steps are substituted, one can obtain update equations for Q and R as follows:

Eq. 3.28can be differentiated with respect to Q^{-1} and R^{-1} to obtain:

$$\begin{aligned} \frac{\partial - 2E[\log p(X_{0:n}, Z_{1:n})]}{\partial Q^{-1}} &= nQ - [S_{11} - AS'_{10} - S_{10}A' + AS_{00}A'] \\ \frac{\partial - 2E[\log p(X_{0:n}, Z_{1:n})]}{\partial R^{-1}} &= nR - [M_{ZZ} + M_{XZ} - M_{XX}] \end{aligned} \tag{3.31}$$

$$\begin{aligned} Q &= \frac{[S_{11} - AS'_{10} - S_{10}A' + AS_{00}A']}{n} \\ R &= \frac{[M_{ZZ} + M_{XZ} - M_{XX}]}{n} \end{aligned} \tag{3.32}$$

The values of Q and R can be initialized with the heuristic covariance matrices obtained in section 3.2, by setting σ_η^2, $\sigma_{\epsilon x}^2$ and $\sigma_{\epsilon y}^2$ as 1, 0.01 and 0.01 respectively.

$$\begin{aligned} Q_0 &= \sigma_\eta^2 \begin{pmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^2}{2} & 0 & 0 \\ \frac{\Delta t^2}{2} & 1 & 0 & 0 \\ 0 & 0 & \frac{\Delta t^4}{4} & \frac{\Delta t^2}{2} \\ 0 & 0 & \frac{\Delta t^2}{2} & 1 \end{pmatrix} \\ R_0 &= \begin{pmatrix} \sigma_{\epsilon x}^2 & 0 \\ 0 & \sigma_{\epsilon y}^2 \end{pmatrix} \end{aligned} \tag{3.33}$$

Extending the EM algorithm for multiple trajectory time series

The previous section dealt with estimating parameters for a single time series. However, in the case of our data, there are multiple time-series trajectories and one needs to estimate parameters for an entire collection of such trajectories. Therefore the expected complete data log-likelihood is maximized for multiple trajectories instead of one trajectory.

$$\text{maximize } E[\log p(X_{1,0:n_1}, Z_{1,1:n_1}, X_{2,0:n_2}, Z_{2,1:n_2})..., X_{j,0:n_j}, Z_{j,1:n_J})]$$

Here J represents the number of trajectories, n_k represents the length of k^{th} trajectory and the expectation is conditioned given the current value of parameters and observed data.

In order to maximize this, one can assume that these trajectories are independent and identically distributed. This assumption may not be entirely valid in our case as discussed more elaborately in Section 2.3. However, this assumption makes simplification easier and also largely holds true.

$$\text{maximize } E[\log p(X_{1,0:n_1}, Z_{1,1:n_1})p(X_{2,0:n_2}, Z_{2,1:n_2})..., p(X_{j,0:n_J}, Z_{j,1:n_J})]$$

The expected complete data log-likelihood can be written as $E[\log p(X_{1:J}, Z_{1:J})]$ for simplicity. Thus, the expected log-likelihood of all trajectories is the sum of expected log-likelihood of individual trajectories and can be written as:

$$\begin{aligned}
-2E[\log p(X_{1:J}, Z_{1:J})] = \sum_{i=1}^{J} (n_i \log|Q| + n_i \log|R| + \log|\Sigma_0| \\
+ E[\sum_{t=1}^{n_i} (X_{i,t} - AX_{i,t-1})' Q^{-1} (X_{i,t} - AX_{i,t-1})] \\
+ E[\sum_{t=1}^{n_i} (Z_{i,t} - CX_{i,t})' R^{-1} (Z_{i,t} - CX_{i,t})] \\
+ E[(X_0 - \mu_0)' \Sigma_0^{-1} (X_0 - \mu_0)] + constant)
\end{aligned} \tag{3.34}$$

By solving similar steps as in the above equations we obtain update equations as follows:

$$\begin{aligned}
Q &= \frac{\sum_{i=1}^{J} [S_{i,11} - AS'_{i,10} - S_{i,10}A' + AS_{i,00}A']}{\sum_{i=1}^{J} n_i} \\
R &= \frac{\sum_{i=1}^{J} [M_{i,ZZ} + M_{i,XZ} - M_{i,XX}]}{\sum_{i=1}^{J} n_i}
\end{aligned} \tag{3.35}$$

A similar update step can also be used to obtain maximum likelihood estimates for the other parameters A, C. The EM algorithm can thus be iteratively used until convergence of all parameters and more details about the implementation of this algorithm is present in Algorithm 1.

3.5 Constant Acceleration model

The constant acceleration model assumes that the objects have constant acceleration, instead of constant velocity. The same constant velocity model with slight adjustments can be rewritten as below:

$$\begin{aligned}
X_t &= AX_{t-1} + \eta_t \\
Z_t &= CX_t + \epsilon_t
\end{aligned} \tag{3.36}$$

where

$$X_t = \begin{pmatrix} x_t \\ \dot{x}_t \\ \ddot{x}_t \\ y_t \\ \dot{y}_t \\ \ddot{y}_t \end{pmatrix}, A = \begin{pmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Now, the state space consists of the positions, velocities and accelerations along the x and y-direction. Accordingly A has also changed to incorporate acceleration in the calculation of position and velocities.

$$Z = \begin{pmatrix} Z_{x_t} \\ Z_{y_t} \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

The observed state contains the observed positions as before. The matrix C is responsible for obtaining positions from the hidden state. As before η and ϵ are normally distributed with 0 mean and covariance matrices Q and R respectively. The parameter estimation approach discussed in the previous section can also be extended to the constant acceleration model. The only difference is that certain parameters have different values and dimensions for parameters A, C, Q and R.

3.6 Neural Network based Architectures

The construction of a Kalman filter makes it very intuitive to use on time series data. However, a Kalman filter suffers from some limitations. Mainly, a Kalman filter cannot model non-linear distributions as all dependencies are linear. There are other non-linear state space models which can be used as an alternative to the Kalman filter. However, due to reasons discussed later in Section 5.2, we resort to explore neural network based architectures. The subsequent sections aim to introduce the foundations and intuitions behind the mechanics of various neural network architectures.

Artificial Neural Networks

In many machine learning tasks, one would want to approximate the dependent variable Y as a function of independent variables X.

$$Y \sim f(X)$$

Equipped with the data for both Y and X, the function f needs to be approximated. Artificial Neural Networks are widely used for their capability to be excellent function approximators [18]. The working of an Artificial Neural Network (ANN) is loosely inspired from the neural connections in the brain. ANN consists of multiple neurons that are conceptually similar to a biological neuron. All these neurons are stacked together in a hidden layer. A typical neural network consists of multiple such hidden layers coupled with an input and an output layer.

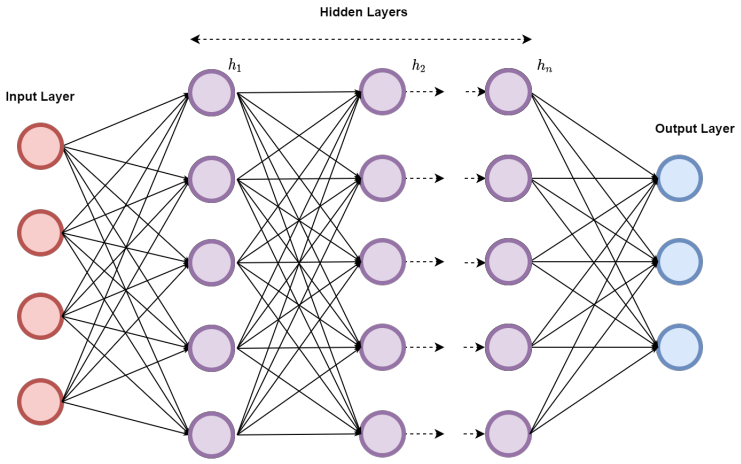


Figure 3.2: An illustration of neural connections in an ANN

The input layer consists of the input features/data vector. There are connections between every neuron in the hidden layer and the input layer. Every connection carries an associated weight that transforms the input to the output of every neuron as below:

$$\begin{aligned} a_j^1 &= \sigma(\sum_{i=1}^{n_f}(X_i W_{i,j}^1) + b_j^1) \\ \sigma(x) &= \frac{1}{1+\exp(-x)} \end{aligned} \tag{3.37}$$

Here a_j^1 is the output at the j^{th} neuron of the first hidden layer and n_f is the number of input features. $W_{i,j}^1$ refers to the weights between the input and the first hidden layer, whereas b_j^1 refers to the bias of the j^{th} neuron in the hidden layer. A non-linear activation function is used to transform a linear transformation of the the input X_i to the output a_j^1. The non-linear activation used above is a sigmoid activation. Many other activation functions such as tanh, ReLU can also be used. The purpose of these activations is to obtain a non-linear transformation of the input X_i. An ANN further consists of multiple hidden layers that perform similar transformations to a_j^1. The final layer of an ANN is called as the output layer as it takes the output obtained from the last hidden layer and transforms it to a vector having the same dimensions as the target feature. Therefore, there are connections between each neuron in the last hidden layer to every neuron in the output layer as below:

$$a_j^l = \sigma(\sum_{i=1}^{n_l}(a_i^{l-1} W_{i,j}^l) + b_j^l) \tag{3.38}$$

Here n_{l-1} refers to the size of the last hidden layer $l-1$ and the output at the j^{th} neuron is calculated as the dot product of the hidden layer output with the output neuron weights added to the bias of the output neuron. An activation function may be applied here, depending on the target feature. If the target feature is categorical, it may be useful to use a sigmoid function activation that squishes the values between 0 and 1. This entire transformation of input through multiple hidden layers and activations to the final output is termed as a forward pass.

Most of the ingenuity in an Artificial Neural network lies in tuning the weights and biases of these networks, given the target data. Having computed the "predicted" outputs from the forward pass, it becomes paramount to construct some error metric to gauge the difference between predicted and actual target vectors. For the problem of regression, a simple mean squared error metric can be utilized. The problem can now be termed as minimizing this

loss function/error metric, by fine-tuning the weights of the network. This loss function represented by $F(x)$ is multi-variate, as it is dependent on many weights and biases. The loss function reduces the fastest in the direction of its negative gradient. Using this observation, the weights can be updated as below

$$W_{n+1} = W_n - \alpha \nabla F(W_n) \tag{3.39}$$

α signifies the learning rate which represents the size of the steps taken towards reaching the minima. In the above equation, it will thus be necessary to compute the gradients of the objective function with respect to each of the weights and biases. For this, gradients of loss are computed with respect to the activations of the final layer. Using the chain rule of differentiation, the gradients of loss with respect to the previous hidden layer's weights and biases can be computed.

$$\frac{\partial F}{\partial W^l_{j,k}} = \frac{\partial F}{\partial a^l_j} \frac{\partial a^l_j}{\partial W^l_{j,k}} \tag{3.40}$$

The chain rule can be applied multiple times to compute gradients backwards for all weights and biases. This part of tuning the weights is referred to as backpropagation and is responsible for training the network.

The idea for training the network is to compute the loss function for the entire data (such as MSE) and then use backpropagation to tune the weights and repeat the process until convergence. However, this becomes computationally slow and also requires loading the entire dataset into the memory. On the other hand loss and gradients can only be calculated for every data point in our dataset and gradient updates can be made for every forward pass. This is called as stochastic gradient descent[25], but this gives noisy updates to the parameters. In practise, the solution is to combine these two ideas to feed-forward mini-batches of data through the network. Average loss and gradients are computed and weights are updated over the complete mini-batch. A complete pass through the entire dataset is called an epoch. The network is trained through multiple epochs until the error estimate/loss function starts converging.

Recurrent Neural Networks

The working of a recurrent neural network (RNN) shares similarities to the working of an Artificial Neural Network. Both ANNs and RNNs contain layers and activation functions with many parameters that are tuned to approximate the target variable Y as a function of input features X. The standard ANN discussed above has connections between the input features and the target feature through neurons and hidden layers. However, when the input/target features are temporal (or correlated across time), there exists no such connections between features separated through time. Thus, the standard ANN cannot be used to model temporal dependencies in the data. RNNs on the other hand have proven to be a modern standard in dealing with time series data. The intuition behind how RNN deals with temporal data can be understood by borrowing some concepts studied in the Kalman Filter section5.1. In the Kalman Filter, there is a hidden state (h_t) that affects the measured output (y_t). At the same time, the hidden state itself is temporal in nature as the next hidden state is dependent on the previous hidden states. As a result of this cross-dependence, the output states themselves are temporally related to each other. The Elman network [22] introduced in 1990 which is one of the most widely used RNN variant uses this concept of a hidden state to good use.

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned} \tag{3.41}$$

where W_h, U_h, W_y, b_h,b_y represent parameter weight,bias vectors and σ_h,σ_y represent activation functions.

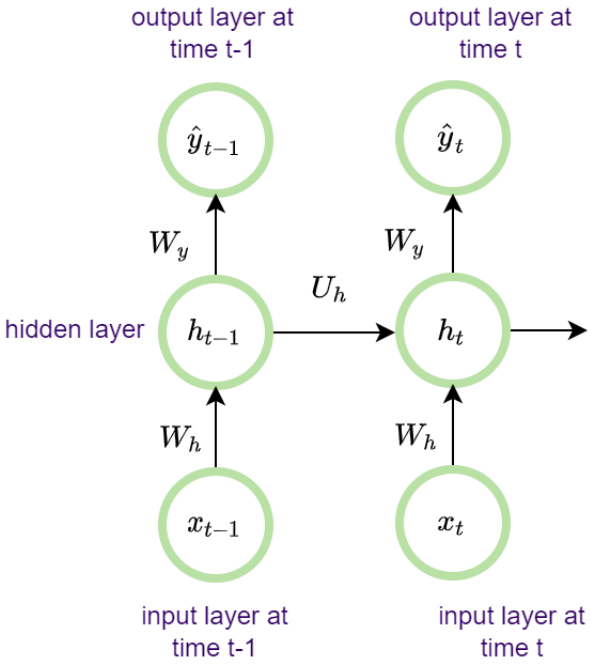


Figure 3.3: A time unfolded representation of the Elman network

As can be seen there is a hidden layer at time t which is dependent on the input at time t and the hidden layer from the previous time $t-1$ through activation functions and respective weights. The previous hidden layer acts as some sort of memory/ past context which when coupled in a specific way with the present input, gives the present context. This present context h_t influences the output y_t through more such weights and activation functions. W_h controls the amount of information that should be passed from the input to the hidden state. Similarly, U_h controls the information flow between past and present hidden state. Lastly, W_y dictates the information flow between the hidden state and the output vector. The same parameter vectors are used for each time step t. Thus, through these hidden states, the network develops some sort of a memory through which time is represented via its effect in intermediate computations. The network's parameters are trained using the same backpropagation principles present in an ANN.

RNNs have been used with high degrees of success on a variety of tasks. However, the novel idea used in an RNN to unroll the network across multiple time steps also proves to be its downfall. As the number of time-steps increases, the network is unrolled over many layers. Let us say that the output at the last layer was dependent on the first one. When the network tries to learn this dependency and the gradients at the last layer are small, the gradients vanish as we move backwards through the many layers present in the network. This is referred to as the vanishing gradient problem [23]. As a result, earlier layers don't learn and the signal is lost. On the other hand if the gradients begin large at the last layer, gradients will get larger as they are backpropagated through to the input. Therefore, the weights near the input layer will be large and as a result will forward propagate larger signals, making the error estimates very large. This is known as the exploding gradient problem. The mathematics of gradient vanishing and explosion are present in [23]. In conclusion, Elman Networks were successful in solving relatively simple problems and as sequences scaled up in size and complexity, these networks frequently struggled.

Long Short Term Memory

The Long Short Term Memory (LSTM) network [11], introduced in 1997 was designed to combat many of the problems that RNNs were facing. LSTMs are special kinds of RNN

architecture that operates in the same loop-like way of an RNN. However the architecture of an LSTM cell is markedly different as it has a gated circuit that controls information flow from previous time step to current time step.

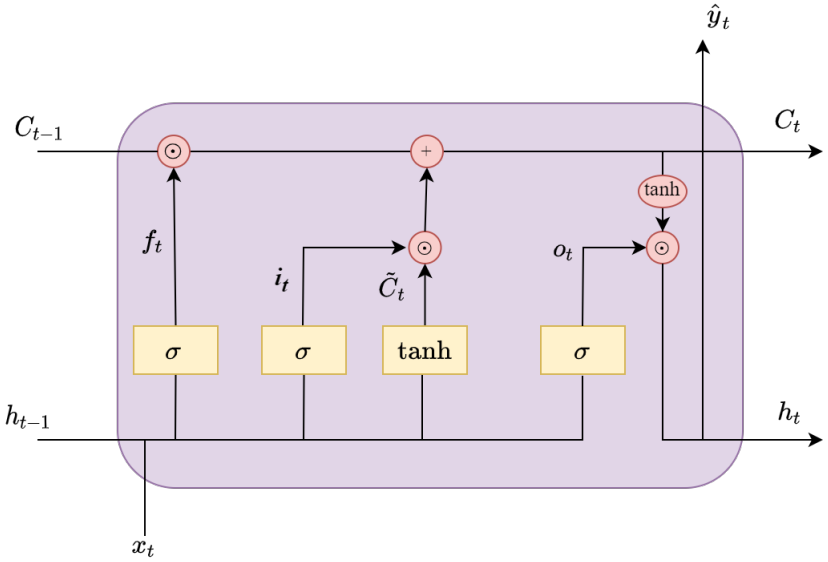


Figure 3.4: LSTM gated architecture

In the figure above, the rectangular boxes are fully connected layers with trainable parameters. The circular boxes refer to element-wise operations between different vectors. h, x and C refer to the hidden state vector, input feature vector and the cell state vector respectively. The LSTM can be thought of as making three separate decisions at each time step: forget/keep information, update information and output information using the memory. To understand these three decisions, a forward pass in this LSTM is elaborated upon.

- Firstly, the feature vector from the current time step (x_t) and the previous hidden state vector (h_{t-1}) are concatenated. For convenience, this concatenated state is referred to as the concatenated input vector from here on. This concatenated input vector is passed through a feed forward layer with a sigmoid activation. The output from this activation is later element wise multiplied with the previous cell state. The purpose of this stage (also called the input gate) is to forget irrelevant information from the previous state.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.42}$$

- Secondly, the concatenated input vector is passed through two feedforward layers: one with sigmoid activation and another with tanh activation. The output from the tanh function decides the update that should be passed into memory for each value and is termed as candidate memory. The output from the sigmoid activation, also referred to as the input gate decides which values needs to be updated. The element wise multiplication of the candidate memory and the input gate combines update information and updates only specific values in this vector. The present cell state is the sum of what we decided to keep from the previous cell state and the updates from the new concatenated input vector.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \end{aligned} \tag{3.43}$$

- Lastly, the concatenated input vector is passed through another feedforward layer with a sigmoid activation, also referred to as the output gate. The tanh function is applied on the cell state and the vector from the output gate is element wise multiplied to obtain the new hidden state. This essentially regulates what information from the cell state needs to be passed on to the hidden state.

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t * tanh(C_t) \end{aligned} \tag{3.44}$$

It can be shown that in these kinds of gated units, the gradients don't vanish exponentially fast ([10]) because of the cell state and that's why it becomes pertinent to use LSTMs in long-trajectory time-series methods.

Self Attention and Transformer Encoders

Transformers have revolutionized the field of natural language processing and computer vision in the last couple of years. They have enabled greater transfer learning potential in machine learning problems and their mechanisms chiefly rely on the concept of attention and self-attention. Attention was first introduced in [2] to improve performance of sequence to sequence tasks in natural language processing. A popular example of sequence to sequence modeling is translation of an input sequence from one language (say English) to another language (say French). The traditional approach to this translation task consisted of encoding entire English sentences into a fixed length vector through an encoder. The decoder then used this fixed length vector to then produce a sequence in French. Here, the encoder needs to compress the whole sentence into a fixed length vector, which may lead to some loss in information. In [2] the model was allowed to automatically search for parts of the source sequence which are most relevant for predicting the next target word, instead of just having a fixed length encoded vector. Thus, in this way attention was used to model the dependencies between the source and a target sequence. Self attention was first introduced in [4] to capture the dependencies between different tokens in the same sentence. The word "self" is used to refer to the fact that attention is now being computed between different tokens of the *same* sentence/sequence. For example in the sentence "John went to the store because he wanted milk", the word "he" needs to be associated with "John". Thus, Self attention is the mechanism through which the model understands these kind of dependencies between different tokens in the same sequence. In our case, we have multiple road users traversing and each of their behaviors is influenced by some other road user's behavior. Therefore it becomes important to try and bake these inter road user dependencies in the model. If we consider the collection of trajectories at a particular time to be a sequence, then self-attention refers to this between-sequence/inter-trajectory dependencies. We propose to use the self attention mechanism in our problem through a transformer encoder architecture. The transformer encoder architecture was introduced in [37], where the transformer was used to process an entire sentence as a whole in a non-recurrent fashion. The recurrent nature of LSTMs or RNNs made their performance dependent on whether the hidden state was preserving information from previous time steps. By processing entire sequences as one, the Transformer architecture had no risk of losing past information. Transformers have revolutionized the field of natural language processing and computer vision, because of their capability to adapt to different tasks. Through this section, this architecture and its constituent building block of self attention will be discussed in detail. The intuition behind how self-attention captures dependencies in a sequence is also elaborated in detail.

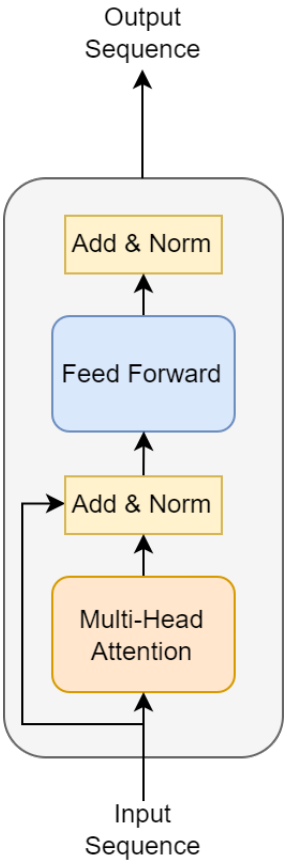


Figure 3.5: Transformer encoder architecture

The transformer encoder architecture consists of a self attention layer followed by a feed forward layer. Let us say that the input to the transformer architecture is a feature vector, where a feature vector somehow represents the individual behavioural state of one road user. At a particular point of time, we may have multiple road users and thus a collection of feature vectors collected into a matrix. Let us assume that there are p feature vectors and d is the dimension of each feature vector, also called as the embedding dimension.

$$\begin{aligned} Q &= XW_Q \\ K &= XW_K \\ V &= XW_V \end{aligned} \tag{3.45}$$

The feature matrix (X) we have is firstly multiplied with three trainable parameter matrices W_Q, W_K and W_V. Each of these matrices have a dimension of $d \times w$. Therefore for each feature vector we obtain a query vector, a key vector and a value vector. As a matrix, we obtain the query (Q), key (K) and value matrices(V) of dimension $p \times w$.

Next, the query matrix is multiplied with the transpose of the key matrix. To understand the intuition behind this, let us say that $p = 2$ or that there are two feature vectors. Therefore the resulting QK^T will be a 2×2 matrix.

$q_1 . k_1$	$q_1 . k_2$
$q_2 . k_1$	$q_2 . k_2$

Figure 3.6: An illustration of QK^T for a 2×2 matrix

The first cell (1,1) of this matrix is the dot product between the query vector of the first feature and the key vector of the first feature. This can be thought of as the score that first feature vector gives itself. It becomes more interesting when we look at the second (1,2) and third cell (2,1). The second cell is the dot product between the query vector of the first feature and the key vector of the second feature. This can be thought of as the score/importance that the first feature gives to the second feature. Similarly the third cell is the score/importance that the second feature gives the first feature and the fourth cell is score/importance the second feature gives itself. Therefore, the resulting matrix represents the importance/attention that every feature vector gives every other feature vector. This entire matrix is divided by the square root of the dimension of the key vectors, that serves a more practical purpose of having stable gradients. Then the softmax function is applied over the resulting matrix to normalize the scores into some kind of a probabilistic distribution. The row sum of the resulting matrix is 1 and each cell of a row represents how important that particular cell is for the row.

Now the resulting normalized score matrix obtained is multiplied with the value matrix. Essentially, every row of the resulting matrix is the weighted sums of different feature's value vectors, weighted by the normalized score of a feature. The n^{th} row will consist of the sum of the weighted value vectors of each feature, weighted by their importance relative to the n^{th} feature.

$$Z = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.46}$$

There can also be multiple-attention heads where there are multiple W_Q, W_K and W_V matrices for each attention head. As a result we would obtain $Z_1, Z_2 ... Z_{natt}$ as the output of these attention heads. These outputs are concatenated and multiplied with another trainable parameter W_O to obtain Z_{net}

$$Z_{net} = W_o(concat(Z_1, Z_2 ... Z_{natt})) \tag{3.47}$$

Each feature vector in the Z_{net} matrix is then passed through two feed-forward layers, with the first feed-forward layer having a ReLu activation to obtain Z_{out}. The input and the output of both the multiple-attention head layer and the feed-forward layer are added. These connections between the input and the output is termed as a residual connection and is implemented for the practical purpose of limiting the vanishing gradient problem. Also, layer normalization is applied where inputs are normalized across all the features as opposed to batch-normalization which normalizes inputs across the mini-batch. The number of elements in a batch varies (depending on the maximum sequence length) and this leads to different number of elements in each batch. As a result, the normalization constant used in batch-normalization varies across batches and this leads to large-fluctuations during training. This is why layer normalization is used instead of batch-normalization.

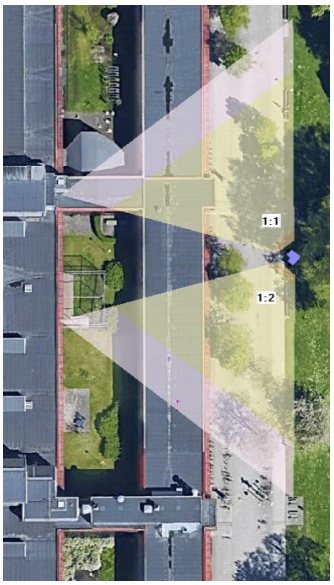
At the end of the transformer encoder, an output feature vector is produced for each feature vector input. The n^{th} output feature vector incorporates information from all the input

features with respect to the n^{th} input feature vector. The details regarding how these neural network architectures are implemented for the specific problem in our thesis is elaborated in the Section 5.3, 5.2

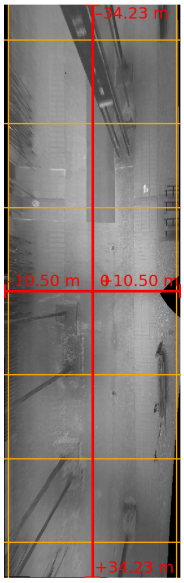
4 Data

Through this section, we present details regarding the features, quantity and quality of data. Further, through data exploration and visualization, key facets and characteristics of the data are presented. Lastly, important pre-processing steps that suitably mould the data for later algorithms are enumerated.

The data has been supplied by Viscando, whose technology tracks all road users to understand movement patterns, traffic flow and human behaviour. Installed on static road infrastructure like light posts or building walls, Viscando's sensor OTUS3D detects, classifies, and tracks all road users using 3D vision and AI. Images simultaneously captured by two spatially separated cameras are converted into a dense 3D point cloud. Point clustering is then applied to the point cloud to detect single road users and to estimate their sizes and 3D shapes. Then, the objects are classified using shape- and appearance-based AI methods. Finally, object tracking using Kalman filter is applied to compute trajectories of different objects, with time-resolved position, speed, acceleration and heading. In case the measurement system consists of several sensors, the detections from individual sensors are merged before tracking. Two such vision sensors were placed on the lightpost of the Linköping University campus. The data collected over a week contains trajectories of pedestrians, cyclists and the autonomous shuttles, collected in collaboration with the EU project SHOW.

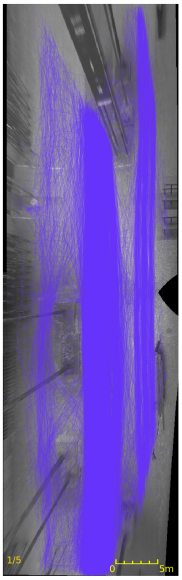


(a) Sensor positions and approximate fields of view

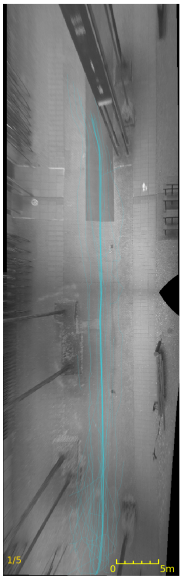


(b) Top view of the arena

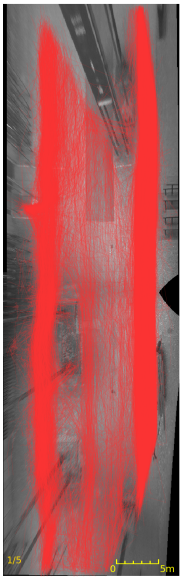
Figure 4.1: A map of the ground and the sensors



(a) Cyclists



(b) Shuttles



(c) Pedestrians

Figure 4.2: Line plots for trajectories of road users

As can be seen from Fig. 4.1a, the field of vision of the sensor covers an area of 70×21 metre squared. Also the actual measurement area represented in Fig. 4.1b is reconstructed by projecting the captured images of the area to the ground, and merging them together.

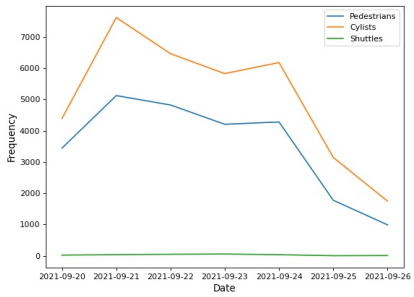


Figure 4.3: Number of road user trajectories across the week

The feature names of the data exported from the measurements along with their definition and type are tabulated below:

Feature Name	Definition	Variable type
ID	Unique ID assigned to each road user	Continuous
Timestamp	Time at which road user was tracked	Time
x	Tracked X-Coordinate of a road user	Continuous
y	Tracked Y-Coordinate of a road user	Continuous
Speed	Tracked speed of a road user	Continuous
Type	Indicator for cyclist, pedestrian and shuttle	Categorical

Table 4.1: Definition and type of variables in the dataset

Type	No. of trajectories	Avg speed (km/h)	Avg Trajectory Length (s)
Pedestrians	24639	4.6	30
Cyclists	35373	14.0	13
Shuttles	208	6.6	21

Table 4.2: Characteristics of each road user in the dataset

4.1 Data Manipulation

Through the analysis of sampling times, there was an interesting observation made. Sampling time is defined as the time difference between any two consecutive observations for the same road user. It can also be thought of as the inverse of sensor frequency, where the sensor frequency is defined as number of frames captured per second. Table 4.3 shows the distribution of sampling time in the data.

Sampling Times (in seconds)	Frequency
Between 0.05 and 0.2	3540
0.24	822904
Between 0.3 and 1.5	8524
Greater than 1.5	4472

Table 4.3: Distribution of sampling times

Times	User 1	User 2	User 3
09:10:40:00	$[x_1, y_1]$	$[x_2, y_2]$	$[x_3, y_3]$
09:10:40:20	$[x_1', y_1']$	$[x_2', y_2']$	$[x_3', y_3']$
09:10:40:40	$[x_1'', y_1'']$	$[x_2'', y_2'']$	$[x_3'', y_3'']$
09:10:41:00	$[x_1''', y_1''']$	$[x_2''', y_2''']$	$[x_3''', y_3''']$

Table 4.4: Aligned road users' data

It is seen that the sampling time is not constant throughout the data. Most of the data has a sampling time of 0.24 seconds. However, there are cases where the sampling time is

higher/lower because of real-time data processing jitter observed in sensors. It is important to maintain a constant sampling time, because then we can have a common time axis for all road users. Then, one can align different road user's data as Table 4.4, and at any particular time, the positions of all road users in the frame are known. In order to make the sampling time uniform, positions and speeds of road users need to be calculated for a common time scale. It was decided to choose a time scale of 0.2 seconds because of its proximity to the most frequent sampling time (0.24 seconds) and for convenience. Let us say a road user's position was known at 0.24, 0.48,0.72, 0.96 seconds the road user's position at 0.2, 0.4, 0.6, 0.8 and 1 second needed to estimated. Simple cubic interpolation was used to interpolate positions and speeds to the nearest 0.2^{th} second. Now the data had a uniform sampling time of 0.2 seconds. By taking the 5^{th} observation of every road user, dataset with a sampling time of 1 second was also created. Thus, there were two final datasets each with sampling times of 0.2 second and 1 second respectively.

4.2 Data processing for recurrent neural networks

Every road user maps out a trajectory which keeps evolving as the road user moves through the frame. A frame corresponds to the positions of different road users in the observed area at a given instant of time. In order to predict where the road user will be in the future, his/her past trajectory is of utmost importance. Therefore, it is desired that the network predicts the coordinates at time step $t+s$ given the input trajectory until time t. For the network to be able to do its best job predicting, data needs to be packaged in an $[input_t^1, target_t^1]$ fashion where $input$ represents what the network needs to know about road user 1 in order to predict the future positions or $target$. The $input$ trajectory of a road user 1 at time t consist of feature representations from each time step $1, 2, ...t$ of his/her trajectory . In turn each of these feature representations at time k are the road user's position (x_k^1, y_k^1) and two dummy variables (ped, cyc) signifying whether the road user is a pedestrian or cyclist. Notice that these two dummy variables will be the same for each time step $t = 1, 2...n$. The $target$ feature is the position of road user s steps later, or in other words, x_{t+s}^1, y_{t+s}^1 needs to be predicted. The data structure also called as the *Individual Trajectories* data structure will consist of two columns $[input, target]$ and the equations below (Eq. 4.1) help in understanding how each row of $[input, target]$ will look like. The Kalman Filter EM Algorithm and the Vanilla LSTM network introduced later in Section 5.1, 5.2 will be trained on this data structure after applying some exclusions that is mentioned later in Section 4.2.

$$\begin{aligned} input_t^1 &= [[x_1^1, y_1^1, ped, cyc], [x_2^1, y_2^1, ped, cyc], ...[x_t^1, y_t^1, ped, cyc]] \\ target_t^1 &= [x_{t+s}^1, y_{t+s}^1] \end{aligned} \tag{4.1}$$

If the data is structured in a different way, there is more information that can be leveraged. In the above representation, the input features at time t consist of a road user's past trajectory and the target feature consists of the road user's future trajectory. However, at time t there may be m multiple road users in the scene, influencing each other's behaviour. At time t it would certainly be more interesting to tell the network that there are other road users in different coordinates at the same time. Thus, the input feature needs to be packaged such that all the current road users' trajectories are present. Consequently the target feature should consist of all the road users' future coordinates. Therefore the representation of the entire scene (and not just one road user) at time t $[input_t, target_t]$ will look as in Eq. 4.2. The data structure called as the *Frames* data structure will consists of multiple rows of such frames and the Attention based LSTM network introduced later in Section 5.3 will be trained on this kind of a data structure.

$$\begin{aligned} input_t &= [input_t^1, input_t^2 ... input_t^m] \\ target_t &= [target_t^1, target_t^2, ... target_t^m] \end{aligned} \tag{4.2}$$

where $input_t^k$ and $target_t^k$ are obtained from Eq. 4.1

Data Exclusions

There are some exclusions applied to the data and the reason for applying these exclusions are elaborated in this section.

When analyzing the distributions of speed, there were some observations which had a very high speed (>50 kmph). On further analysis, it was seen that this happened mostly at the start frame when the road user was captured. It is important to know that in the data given to us, the first three frames of the tracked road user is discarded as they will contain noisy tracked estimates of positions and speeds. Speeds are derived from previous positions and sometimes it may be the case that speed estimate in the fourth/fifth frame is also noisy, because uncertainty in measuring speed is higher than uncertainty in measuring position (as the speeds are derived from the positions). Therefore, for safe measure these road users ($< 0.5\%$) are discarded in our analysis.

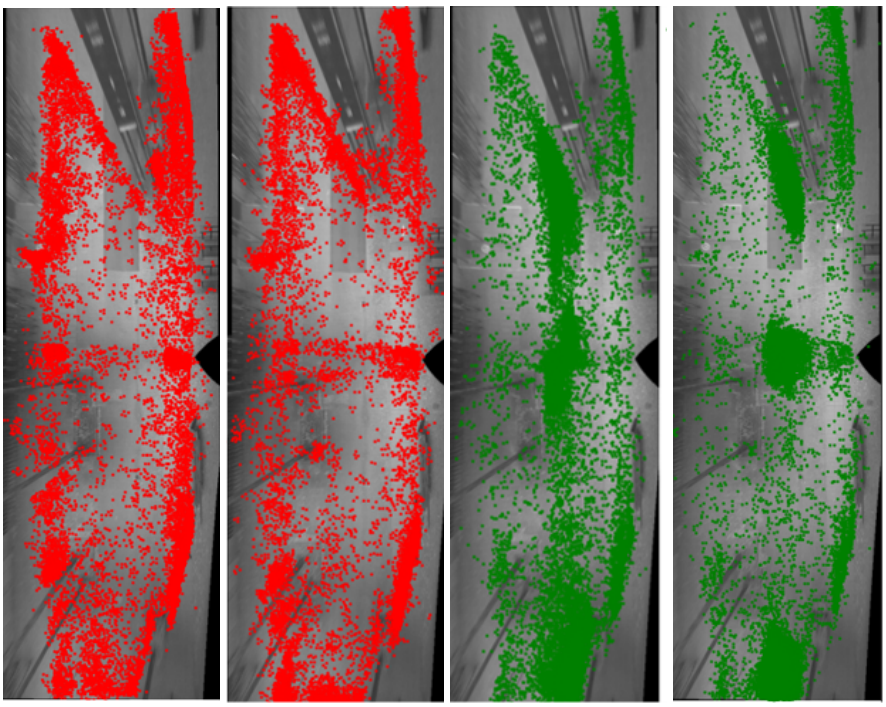
It is also mandated that the trajectory of a road user consists of at least two seconds (which is two timesteps steps in a 1 second sampled data and 10 timesteps in a 0.2 second sampled data) for future predictions. This is done so that there is enough input to the network, before a network starts predicting. This serves the purpose of eliminating noise from too few observations to make a prediction. This exclusion is applied to all these datasets.

There are some interesting cases when constructing the *Frames* data structure. There may be three road users at the time of observation, but s time steps later, one of them (or all) may have left the frame. In this case for the missing road user who left the frame, the target features are populated as NaNs (not a number/missing value). As mentioned later in Section 5.3, the network does not calculate loss over these cases. There may be also be cases where a new road user entering the scene has just one second of data, whereas all other road users have history that extends multiple seconds into the past. Thus, one row of the *Frames* data structure has some road users with a good past and some road users without the minimum two seconds. One option here would be to remove this new road user from the observation (input) and prediction (target). However, that road user, even with one observation, may be influencing the scene as his/her entrance to the scene may cause a change in behavior for all other road users. Therefore his/her data removal may violate the natural social structure present in the scene. In order to avoid this problem , that entire timestamp's scene $[input_t, target_t]$ is removed from the data. In the next timestamp, this new road user will have two prior timesteps of features and if no other road users freshly enter the scene, this scene is included for training. This step doesn't exclude entire road users' trajectories, but predictions are not performed on some time steps in these road users' trajectories.

Another interesting case that is observed in the *Frames* data structure is that two pedestrians/ two cyclists are very close to each other throughout their trajectory. Sometimes the distance between the two road users is 0 throughout the trajectory. This means that one of the road users was detected as two road users. These kinds of observations were removed from the dataset. Also, two road users can be quite close to each other when they are walking. For example, if acquaintances are walking together, they do not maintain a lot of distance between them. A heuristic cutoff of 20 cm is used as the minimum distance between any pair of road users throughout their trajectory. This means that if two road users maintain a distance of less than 20 cm at all points in their trajectory, only one road user is considered as we assume that one road user was detected as two. Through this cutoff, $< 1\%$ of road users were removed.

It is worthy to mention that there is still some noise in the data. The Figure 4.4 represents clusters of start and end points for pedestrians and cyclists. It is seen that for pedestrians, the start and end points are detected at the borders of the frame which is to be expected. However, sometimes cycles start getting detected late or stop getting detected too soon. This may be because of errors in calibration that results because of the shift in object position when

the objects move between two fields of views. It maybe happening more for cyclists because cycles are relatively faster and their trajectories are more volatile leading to these trajectory breaks.



(a) Pedestrian start positions (b) Pedestrian exit positions (c) Cyclist start positions (d) Cyclist exit position

Figure 4.4: Start and end points of detected positions for cyclists, pedestrians

The trajectory breaks don't affect the CV or LSTM model too much, as these models can still train on the observed part of the trajectory. But these trajectory breaks will affect the training of the Attention based LSTM model as the undetected road user may be influencing the trajectories of the detected road users. However, since this noise cannot be eliminated without removing many frames/trajectories we proceed with the assumption that this noise doesn't affect training too much.

After all the cleaning, roughly 25% of road users were removed, of which most of them had a very short trajectory length ($<$ 2 seconds). The *Individual Trajectories* data structure can be used for the Kalman Filter CV model training and for the Vanilla LSTM network. The *Frame* data structure can be used for the attention based LSTM network's training as this network tries to model the social behavior. To make matters more convenient, the trajectories present in the *Frames* data structure are used to train the Kalman Filter network, LSTM network and the Attention based LSTM network. This is because it would be interesting to see how these methods compare when trained and tested on the same data.

5 Methodology

In this section, the rationale behind choosing the methods mentioned in the theory is elaborated. Further, implementing these theoretical methods on the specific data is presented in more detail.

5.1 Constant Velocity Model and the Kalman Filter

The data as we have studied in the previous section consists of pedestrians, cyclists and shuttles. The trajectories of all these road users are present from the moment they were detected by the sensors to the moment their detection was stopped. These trajectories themselves are temporal in nature as the position in the future is correlated with the position in the past. Multiple trajectories thus can be treated as multiple time series/autoregressive sequences. A solution often used when dealing with time-series specific problems is the Kalman filter. The Kalman filter as mentioned in Section 3.1 constructs dependencies between a latent variable and an observed variable.

$$\begin{aligned} X_t &= AX_{t-1} + \eta_t \\ Z_t &= CX_t + \epsilon_t \end{aligned} \tag{5.1}$$

where η_t and ϵ_t are distributed as $N(0, Q)$ and $N(0, R)$ respectively.

From the above equation, we see that the latent variable has temporal dependencies and the observed variable is a function of the latent variable. If the model parameters and the hidden states in Eq. 5.1 are known, one can calculate the future hidden states and future observed states. The advantage of a Kalman filter is that one can make n-step predictions not only for the means of the observed states, but also for the uncertainty around the observed states. This makes the Kalman filter useful in calculating predictions ahead of time. In the case of our problem, the latent variable is assumed to be the actual positions and speeds of a road user. The observed variable is the measured position that is obtained from the data. This setting along with specific parameters for A and C is used in the constant velocity model.

$$X_t = \begin{pmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{pmatrix}, A = \begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix} Z = \begin{pmatrix} Z_{x_t} \\ Z_{y_t} \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The constant velocity model has been extensively used in the object tracking literature. The tracking problem deals with obtaining noise free estimates of positions given noisy sensor measurements of an object's position. The constant velocity model is considered as a robust method for tracking and can also be extended to prediction ([31]). This has already been done before and the constant velocity model is generally considered as a solid, robust baseline for intent prediction. For all these reasons, it was decided to go ahead with a CV model albeit with a slight change in implementation.

The values of Q, R are usually empirically chosen in CV models. However, in this paper we aimed to estimate Q, R through data driven approaches. This was achieved through the EM algorithm whose theoretical details are present in Section 3.4. In order to estimate parameters for these multiple trajectories, one solution is to model all these road users separately/uniquely. However, there are a large number of unique road users and the built model cannot be extended to a new road user entering the frame. The other solution is to divide the road users by some behaviours and model these road users collectively. For example, pedestrians, cyclists and shuttles can be uniquely modelled. One can also further model these categories into subcategories based on how fast or how slow they are moving at the start of the frame. In this paper, parameters will be estimated separately for pedestrians, cyclists and shuttles.

Algorithm 1 EM Algorithm for multiple trajectories parameter estimation

Ensure: Q, R are symmetric matrices
Initialize $\mu_0, \Sigma_0, Q_0, R_0$ from 3.21 and 3.33
A, C is known according to 3.15
for road users r **do**
  while Q_k and R_k have not converged **do**
    for each trajectory i of trajectory length n_i **do**
      Run the forward filtering step to obtain X_t^t, P_t^t 3.1
      Run the backward smoothing step to obtain $X_t^{n_i}, P_t^{n_i}, X_{t,t-1}^{n_i}, P_{t,t-1}^{n_i}$ 3.1
      Compute S_{00},S_{11}, S_{10} and S_{00} 3.29
      Compute M_{ZZ}, M_{XZ}, M_{XX} 3.30
      $Q_i \leftarrow [S_{11} - AS_{10}' - S_{10}A' + AS_{00}A']$
      $R_i \leftarrow [M_{ZZ} + M_{XZ} - M_{XX}]$
    end for
    $Q_k \leftarrow \frac{\sum Q_i}{\sum n_i}$
    $R_k \leftarrow \frac{\sum R_i}{\sum n_i}$
    $Q_k = 0.5[Q_k + Q_k^T]$
    $R_k = 0.5[R_k + R_k^T]$
  end while
end for

The algorithm above calculates Q, R for pedestrians, cyclists and shuttles. This algorithm can be extended to calculate A, C as well or calculate all of these parameters for some other linear Gaussian state space model such as the Constant Acceleration model. Once parameters are obtained, the task of prediction is pretty straightforward. For the first b timesteps, no prediction is done and filtered estimates of the hidden state are obtained. This is called as the burn-in period to give a good starting point for the hidden states. After b timesteps are complete, the Kalman filter is propagated forward (using again the filtering algorithm in

Section 3.1) s steps ahead to obtain the latent and measured state. The mean, covariance of the latent and measured state can be estimated through simple forward filtering as illustrated in Fig. 5.1.

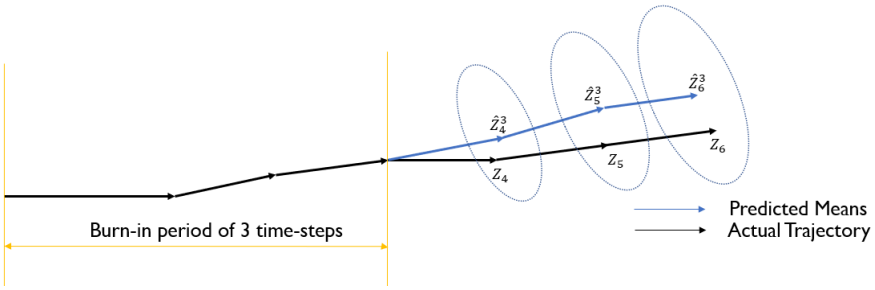


Figure 5.1: Illustration of predicted means and covariances

Fig. 5.2 gives an intuition behind the predictions in the CV model. At $t = 1,2,3$, the positions of the road user is known (represented by Z_1, Z_2, Z_3) and this phase is known as the observation window. At $t = 1,2,3$, hidden state estimators can be calculated using the known positions Z_1, Z_2, Z_3 through the filtering algorithm. The estimated velocity in the observation window is a function of the distance the road user has travelled in every time step. The velocity at $t = 3$ will be higher than the velocity at $t = 2$ since the road user travels more distance between Z_2 and Z_3 than between Z_1 and Z_2. During prediction, the last known velocity $\dot{x}_3$ is assumed to be the mean velocity at $t = 4,5,6$. Therefore during prediction, the Kalman Filter CV model extrapolates state dynamics from the last known observation point into the future.

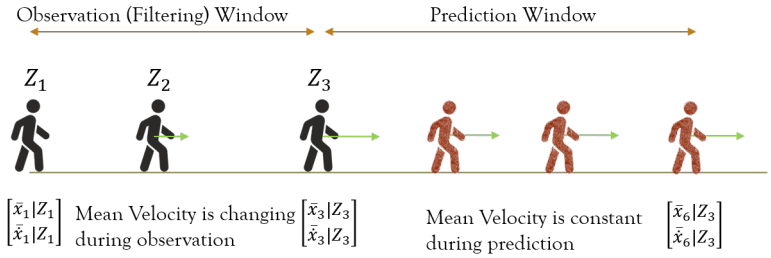


Figure 5.2: An illustration for the intuition behind prediction in the CV model

5.2 Vanilla Long Short Term Memory networks

There are three major limitations to using the CV model for predictions:

- The transition from the hidden state at time t to the hidden state at time $t+1$ is a linear transition. The emission between the hidden state and the observed state is also linear. However, the behaviour of road users is highly non-linear and modeling through this kind of a linear Gaussian state space model is not ideal.
- The number of parameters in the hidden state is very few. Usually pedestrian/cyclist behavior cannot be explained only through their past positions and speeds. For example a cyclist when executing a turn, traverses a curve and this curve is dependent on the speed he/she has when making this turn. There are many more such behaviors and in-turn more parameters that decide trajectory of a road user.
- It is also difficult for filters to capture the spatial and social dependence of a road users' trajectory.

One way of solving these limitations is to resort to non-linear state space models. However, it has been seen ([30]) that they don't offer considerable lift from a CV model. This is probably because even these models assume some kind of road user behavior which is not always displayed by a road user in isolation. A road user can alternate between behaviors such as constant velocity or constant acceleration or constant turn depending on the situation. Also, as the non-linear model complexity increases, it becomes increasingly difficult to use parameter estimation algorithms such as the EM algorithm as calculating closed form update equations for the parameters become increasingly complex.

Therefore, it may be necessary to move away from filter-based models and rather look at an unlikely philosophical off-shoot of the Kalman Filter. The below equation governs the mechanics of the Elman network. This network also has a hidden state propagating forward in time and an observed state which is a function of the hidden state.

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned} \tag{5.2}$$

where W_h, U_h, W_y, b_h,b_y represent parameter weight,bias vectors and σ_h,σ_y represent activation functions. This is a widely used RNN-variant and improves on the Kalman Filter as all the relationships between the hidden states and the observed state is non-linear. With the help of backpropagation, all the parameter weights and biases can be obtained through training on the data. If the parameters are known, the network can be propagated forward in time to obtain estimates of h_t and y_t. Thus these networks are more complex and at the same time easier to train as analytical closed form expressions need not be derived as in 3.35. This make it a top choice for understanding patterns in temporal data. However, when the trajectory lengths are long these networks suffer from the vanishing gradient or the exploding gradient problem. To overcome these shortcomings of an RNN, Long Short Term Memory (LSTM) was introduced. Moreover LSTM networks have also been widely used with high degrees of success in intent prediction tasks ([1])[27]). The idea behind using an LSTM network for our problem is to better capture the dynamics of a road user's trajectory, compared to the CV model.

The implementation of this network is done on PyTorch. The data is fed into the network in the format mentioned in Eq. 4.1.Every neural network takes as input batches of data of size b. The forward step is done on the entire batch of data and this basically means that multiple tuples $input_1, input_2, ..., input_b$ are clustered together and fed forward into the neural network. Each of these inputs consist of feature representations of the entire trajectory. At a particular time step t a feature representation consists of the x,y coordinate as well as dummy variables indicating the type of road user. Therefore each of these inputs are of size $t \times 4$ where t signifies trajectory length. Inside a batch of data, the first input may have a trajectory length of 200, whereas the second input may have a trajectory length of just 10. In order to train a neural network, this trajectory length must be constant throughout the batch. This is because internal computations (such as calculation of intermediate outputs in a network) are done on the entire batch together. Therefore every trajectory in the batch is appended with multiples of $[0, 0, 0, 0]$ until they reach a length of the longest trajectory in the batch. This concept is known as padding and is quite prevalent in dealing with varying length sequences. Thus, $b \times t_{max} \times 4$ is the dimension of the input data where t_{max} represents the maximum trajectory length in the batch. However, more computations are performed over these padded indices and this costs quite a lot of computation time. In order to save compute time and not make the network calculate over these padded indices, there is a very clever idea called as packing implemented in PyTorch. Let us say that there were varying length trajectories in one batch. If the lengths of these trajectories were passed as input, internally the trajectories would be sorted and stacked like the image below.

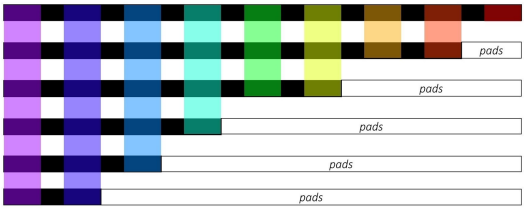


Figure 5.3: Padded sequences sorted by decreasing lengths

At time step 1 and 2 in Fig 5.3, the effective batch size that is considered by the network for computation is 6. This means that all the 6 observations are used for calculation of internal states. At the third time step, the effective batch size is now 5, indicating to the network that the compute now needs to be performed only for the first five observations in the entire batch. Thus, by using this concept of an effective batch size at each time step, the network knows how to ignore padded values.

For a trajectory length of t, the network produces the hidden states at each time step $h_1, h_2 ... h_t$ of size h for an LSTM cell. The final hidden state h_t can be considered as a good representation of the trajectory as it contains information from previous time steps. Multiple such LSTM cells can be stacked one after the other so that the output of one LSTM cell is the input to another. If there are n layers, then n final hidden states are obtained in total $(h_t^1, h_t^2, ..., h_t^n)$. All of these final hidden states are concatenated to obtain a total dimension of nh. This concatenated hidden state is then finally passed through a feedforward layer which brings down the dimensions required such that for each trajectory, the output from the LSTM and feedforward layer is k-dimensional.

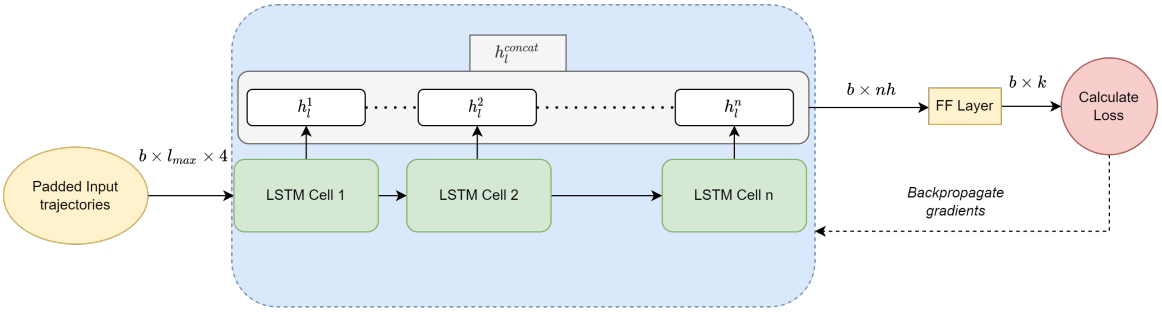


Figure 5.4: The architecture of the Vanilla LSTM approach

h, n are the hidden layer size and the number of hidden layers and these are design choices that are made to tune the network's performance. In addition k can be chosen to represent two different kinds of predictions:

- The final predictions can be made to represent the x and y coordinate of the road user at time step $t + s$. If (x_i, y_i) and $(\hat{x}_i, \hat{y}_i)$ represent the actual and predicted positions at observation i and n_{obs} represents total number of observations,

$$MSE = \frac{\sum_{i=1}^{n} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2}{n_{obs}} \tag{5.3}$$

  Mean squared error can be used as a loss function which needs to be minimized. An alternative metric that can be measured is the average final displacement error (AFDE), which is the displacement between the final predicted and actual position. This is computed as follows:

$$AFDE = \frac{\sum_{i=1}^{n} \sqrt{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2}}{n_{obs}} \tag{5.4}$$

In our case, MSE is used as the loss function. For every batch, MSE is computed and the gradients for this loss are back-propagated through the network to obtain an update for each parameter in the network. In this case, the feedforward layer brings down the dimension coming out of the LSTM output to 2.

- Instead of directly predicting the positions, one can also predict the means and covariances of a normal distribution such that

$$(\hat{x}_t, \hat{y}_t) \sim N(\mu_t, \sigma_t, \rho_t) \tag{5.5}$$

where μ_t, σ_t, ρ_t represent the predicted means, covariances and correlation coefficient. The log-likelihood of observing (x_t, y_t) given this normal distribution can be written as

$$\begin{aligned} log(p(x_t, y_t|\mu_t, \sigma_t, \rho_t)) &= \log(N(\mu_t, \sigma_t, \rho_t)) \\ &= \log N(\begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \end{pmatrix}, \begin{pmatrix} \sigma_{t,x}^2 & \rho_t \sigma_{t,x} \sigma_{t,y} \\ \rho_t \sigma_{t,x} \sigma_{t,y} & \sigma_{t,y}^2 \end{pmatrix}) \end{aligned} \tag{5.6}$$

where $\mu_{t,x}, \mu_{t,y}, \sigma_{t,x}, \sigma_{t,y}, \rho_t$ represent the predicted x and y-coordinate means, their respective variances and also the correlation coefficient The objective is to now maximize the log-likelihood in Eq. 5.6 and it was individually derived that for a bivariate normal distribution,maximizing the log-likelihood is equivalent to minimizing $f(x_t, y_t, \mu_{t,x}, \mu_{t,y}, \sigma_{t,x}, \sigma_{t,y}, \rho_t)$ where f is given in the below equation.

$$\begin{aligned} f(x_t, y_t, \mu_{t,x}, \mu_{t,y}, \sigma_{t,x}, \sigma_{t,y}, \rho_t) &= \log(\sigma_{t,x}\sigma_{t,y}) + 0.5[\log(1-\rho_t^2)] + 0.5(x_t - \mu_{t,x})^2 \sigma_{t,y}^2 \\ &+ 0.5(y_t - \mu_{t,y})^2 \sigma_{t,x}^2 - \rho_t (y_t - \mu_{t,y})(x_t - \mu_{t,x}) \sigma_{t,x} \sigma_{t,y} \end{aligned} \tag{5.7}$$

Instead of the network predicting $\sigma_{t,x}, \sigma_{t,y}, \rho_t$, the network can instead be made to predict $\log(\sigma_{t,x}), \log(\sigma_{t,y})$ and $\text{arctanh}(\rho_t))$ for numerical stability. In addition to $\mu_{t,x}$ and $\mu_{t,y}$ the network predicts a 5-dimensional output. Hence the feedforward layer in this case brings down the dimension coming out of the LSTM output to 5. It can also be the case that the position of a road user more closely resembles some other known distribution, other than the normal distribution. However, for this paper, the assumption is that a road user's final position is normally distributed with some mean and covariance matrix.

5.3 Attention based LSTM network

There is one limitation to the approach used in the above section. The Vanilla LSTM model tries to model the dynamics of individual trajectories in the best way possible. However, sometimes road user dynamics are also governed by some other road user in the vicinity. For example, if there was a cyclist coming in front of a pedestrian, either of them will move away from the point of collision. Also, there are cases when two pedestrians moving in the same direction move similarly as one follows the path that the other has taken subconsciously. Road users tend to make these complex maneuvers that takes into consideration other road users without seemingly thinking a lot about it. Therefore, when it comes to predictions, it becomes important to incorporate the social aspect of behaviors in a road user's trajectory. The vanilla LSTM network by itself cannot be used to model dependencies between different trajectories. This is because all of these different trajectories constitute different time-series sequences and a vanilla LSTM can be used to model these trajectories one at a time. To model all these trajectories *together*, one would need to combine/pool these networks' outputs as done in [1]. The

Social-LSTM [1] is a benchmark paper that has shown to capture social interactions between different road users to a reasonable extent. The principle of a social LSTM is to combine the workings of a Vanilla LSTM with a so-called social pooling layer. At a particular frame, all road users' features are passed into a plain LSTM network. This LSTM network then produces hidden states for each of these road users. These hidden states represent what a road user would do without the presence of other social factors. If a pedestrian's trajectory needs to be predicted, then their neighbors' (who are within a certain spatial distance) hidden states are stacked in a specific way to construct a pooled tensor (Fig 2.1). Neighbor hidden states outside a certain radial distance are discarded. The pooling thus manually combines hidden states based on their proximity to the ego-road users. However, it would be more interesting to let the network itself selectively choose which neighbors' hidden state is important in determining the ego road users' trajectory. In our paper, Attention discussed in 3.6 is used in soft-assigning weights to the different hidden states. Soft-assignment of weights ensures all neighbours' hidden states are considered, with some hidden states being more important than others. In conclusion, the output representation coming from the LSTM network for each road user is passed through an attention layer which learns how to assign weights and combine the different hidden states . The hypothesis for building this kind of a network is that:

- The LSTM networks should output a representation of a road user's behavior without any social influence
- If a road user's trajectory was changed due to some other road user(s) in the network, this behavior should be captured by the attention layer

Practically, this is implemented as a transformer encoder architecture which basically consists of multi-head attention layer and a feedforward layer as mentioned in Section 3.6 . The format of data that is passed into the network is mentioned in Eq. 4.2. Essentially, the number of road users in a frame is another dimension that is added to the previous data used for the Vanilla LSTM network. The input to the vanilla LSTM network was of dimensions $b \times l_{max} \times 4$, where b represents batch size and l_{max} represents the longest trajectory length in the batch. The output coming from the LSTM network for each road user is of dimension $b \times nh$ where h is the dimension of the hidden state. In total there will be r hidden states for all road users present in the frame. In a batch of frames, there may be instance with different number of road users in each frame. A network cannot take variable sized input and therefore one option is to pad the hidden states outputted from the LSTM network and then pass it into the transformer. However, the idea of packing cannot be used in a transformer as a transformer computes on all instances of time at once. There is no recurrent nature in a transformer and because of this, the concept of effective batch size cannot be used. One solution is to use the concept of attention masking where attention scores are not calculated for padded positions in the input. The concept used in this thesis is to batch together all frames/instances where the number of road users are same. For example there may be multiple instances/occurrences when there are 4 road users in a frame. All these instances are batched together and sent into the Attention based LSTM network. The number of road users in a frame can range between 0 to 20.

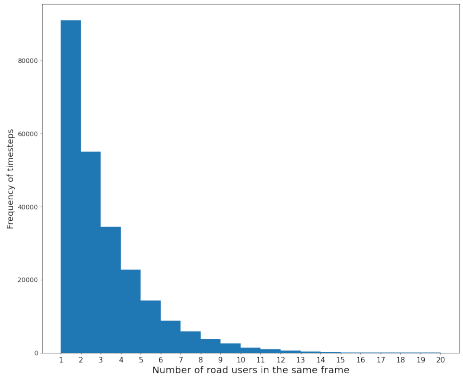


Figure 5.5: Distribution of number of road users in the same frame

Again, a feedforward network is used to bring down the dimension of the output coming from a transformer to $b \times k$ where k represents the output dimension (can be equal to 2 or 5 depending on the loss function used as in Section 5.2). The final predictions are made for all the road users in the frame and the loss for every training instance can be the MSE loss or the log-likelihood loss presented in the previous section. In the case that the road user has exited the frame at prediction time, the actual positions of the road user is unknown. In these circumstances, loss is not calculated for these road users.

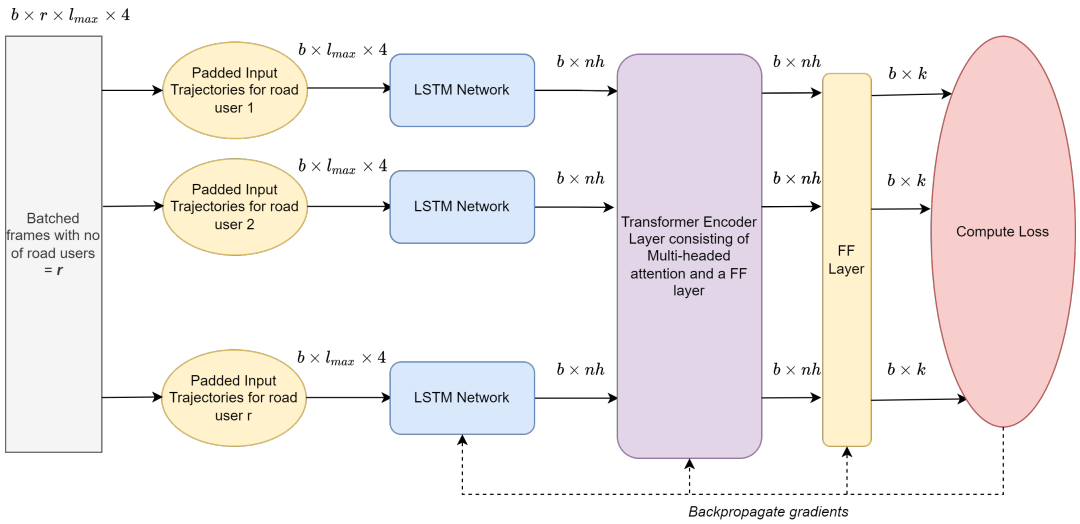


Figure 5.6: Architecture of the self-attention based LSTM

6 Results and Discussion

In this section, we will focus on the quantitative and qualitative results obtained through applying the various methods on our data. Firstly there are two datasets: the dataset with a sampling time of 0.2 seconds and a dataset with sampling time 1 second. For convenience we will call them 0.2s dataset and 1s dataset respectively. The Kalman Filter constant velocity (CV) model has been trained and tested on both these datasets, whereas the neural network models have been trained and tested on 1s dataset. This is because it is computationally more expensive to train the network on the 0.2s dataset, as there are 5 times the number of observations. The chosen dataset is then split into train/test/validation. Since this data was collected over a week, 4 days of this week including a weekend day have been chosen for training. 2 days (including another weekend day) of data have been used for validation with the remaining 1 day used for test. Also, for the purposes of training, a minimum of 2 seconds of past trajectory is required as mentioned in the data exclusions section. This means that all trajectories are at least 2 seconds old. Given trajectories up till time t, the prediction task is to approximate the position of the road user s steps ahead. If s is chosen to < 1 second, the prediction although accurate won't be helpful since all reactions to this prediction should be made in less than 1 second and this isn't practically possible. If s is chosen to be very high (say > 6 seconds), the predictions themselves will deviate a lot from the actual positions, since there is high scope for change in trajectory with such a large window in a slow-traffic setting. Therefore it was decided to predict 3 seconds into the future and this setting applies to all the results mentioned here.

6.1 Constant Velocity Model

For this model, the train/test/validation split was done on 0.2s dataset and 1s dataset. Algorithm 1 was used on the training data to obtain parameter estimates for Q, R. It was seen that these matrices converged in 10-15 iterations/passes through the data. The obtained parameters were then used to propagate the model forward and obtain the mean and covariance of prediction. The below table tabulates the mean squared error of 3s ahead prediction on the test,validation dataset for the three different road users.

Road User	0.2s Dataset	1s Dataset
Pedestrian	1.4	1.2
Cyclist	2.8	3.7
Shuttle	2.4	2.4
Total	1.9	2.2

Table 6.1: 3s ahead mean squared error on validation

Road User	0.2s Dataset	1s Dataset
Pedestrian	1.2	1.2
Cyclist	2.7	3.6
Shuttle	1.7	1.2
Total	1.8	2.2

Table 6.2: 3s ahead mean squared error on test

Apart from MSE, the likelihood of observing the actual position given the predicted mean and covariance can be estimated. The average log-likelihood estimate which is a proper-scoring rule can be constructed as below:

$$\frac{\sum_{i=1}^{n}\sum_{t=1}^{t_i}\log(p(x_t, y_t)|\mu_{x,t}, \mu_{y,t}, \Sigma_t)}{\sum_{i=1}^{n}\sum_{t=1}^{t_i}[\mathbb{1}]} \tag{6.1}$$

Road User	0.2s Dataset	1s Dataset
Pedestrian	-3.3	-2.3
Cyclist	-3.8	-3.4
Shuttle	-3.4	-2.7
Total	-3.5	-2.7

Table 6.3: Avg log-likelihood on validation

Road User	0.2s Dataset	1s Dataset
Pedestrian	-3.3	-2.3
Cyclist	-3.8	-3.4
Shuttle	-3.4	-2.4
Total	-3.5	-2.7

Table 6.4: Avg log-likelihood on test

As can be seen from Tables 6.1 6.2, the MSE is almost similar for all road users except for cyclists where MSE is lower for the 0.2s dataset. This is because cyclists tend to move faster and a higher sampling time will give relatively more information. Also Tables 6.3 6.4 present the average log-likelihood score. The reason that the log-likelihood for cyclists is lower for the 0.2s dataset compared to the 1s dataset, even when the MSE is lower for the former is because both distributions have very different covariance matrices and hence differently shaped distributions. The model built on the 0.2s dataset has a larger covariance matrix at the time of prediction. It is to be noted that 3s ahead prediction for the 0.2s dataset implies a 15 step ahead prediction. The covariance matrix blows up as shown in 5.1 when taken 15 steps ahead and this leads to a flatter distribution. The large covariance matrix implies a non-informative /flat distribution leading to a lower log-likelihood estimate. From this, we can see that the covariance matrix Q, R estimated from the 0.2s dataset is not very informative.

6.2 Hyperparameter-tuning for the networks

Vanilla LSTM

For all the neural network based experiments, the batch size was set to 64 and the learning rate to 0.0005. For the vanilla-LSTM, the number of layers and the size of hidden layers are two important hyperparameters. To tune them, a coarse grid hyperparameter tuning was employed where the different networks were trained on combinations of different parameters. The networks' validation MSE is shown below:

Hidden Layer Size	Number of LSTM layers	Validation MSE
10	2	1.18
	3	1.16
	4	1.16
16	2	1.13
	3	1.13
	4	1.10
20	2	1.09
	3	1.12
	4	1.09
25	2	1.08
	3	1.09
	4	1.07
28	2	1.05
	3	1.05
	4	1.05

Table 6.5: Hyperparameter tuning for the LSTM network

We would want to select the hyperparameters that lead to the lowest validation MSE. As can seen from Table 6.5 above, the validation MSE is very similar for the last three entries and the least complex network amongst these three networks was chosen (highlighted in green)

Attention based LSTM

To train this network, parameters were shared from the trained LSTM network and dropout of 0.1 was implemented in the transformer encoder layer. Dropout arbitrarily drops some percentage (10 in this case) of neurons from training to prevent the model from overfitting. A similar cross-validation as in the above section was performed, with an additional parameter of number of attention heads.

Hidden layer size	No. of Layers	No. of heads	Validation MSE
20	2	2	1.05
		4	1.1
	3	2	1.04
		4	1.03
	4	2	1.05
		4	1.04
28	2	2	0.99
		4	0.99
	3	2	0.98
		4	0.99
	4	2	1.04
		4	1.00

Table 6.6: Hyperparameter tuning for the Attention based LSTM network

6.3 Quantitative comparison between the different methods

The best models were chosen through hyperparameter tuning and the MSE obtained on the test data for different road users is tabulated below:

Road user \ Models	CV	LSTM	TLSTM
Pedestrian	1.20	0.68	**0.66**
Cyclists	3.57	1.63	**1.47**
Shuttles	**1.23**	1.37	1.61
Total	2.19	1.08	**1.00**

Table 6.7: Comparison of Test MSE for different models across road users

For convenience the Attention based LSTM network is also referred to as the TLSTM network (standing for Transformer based LSTM) and the plain LSTM network is referred to as just the LSTM network. It is seen that the LSTM based models substantially outperform the CV model. The TLSTM model is only able to outperform the vanilla LSTM model slightly. The MSE is in general highest for cyclists, followed by shuttles and then pedestrians. Cyclists tend to move faster and are thus more unpredictable in terms of estimating positions. There are very few data points for the networks to learn behavior of shuttles and therefore a conclusion cannot be drawn for this population. Pedestrians move more predictably and that's why their errors are lower across all models. An in-depth comparison of these models, and why they perform the way they do is presented in the subsequent section.

Also, another hypothesis is that the TLSTM model fares better than the Vanilla LSTM model when the number of road users in the same frame increases. To test this, we have taken the ratio of MSE error between the TLSTM model and LSTM model for different number of road users in the same frame.

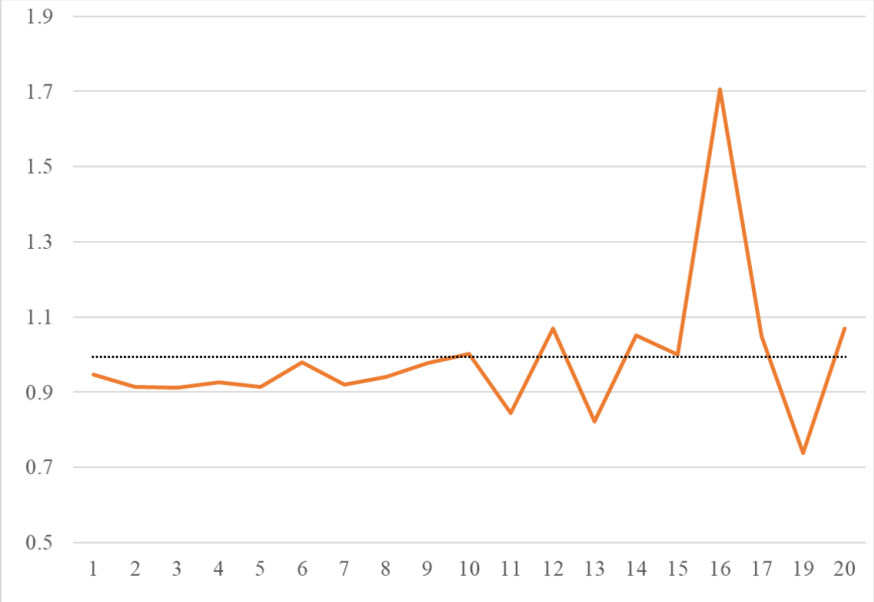


Figure 6.1: Ratio of MSE between TLSTM vs LSTM across number of road users in a frame

This ratio is supposed to be 1, if they perform equally well (represented by the dotted black line in Fig. 6.1. It is expected that the ratio between these errors decrease (or that the TLSTM model performs better and better) as the number of users in the same frame increase.

However when the number of road users increase, detecting and tracking these road users get harder and noisy detections/estimates of positions are obtained. Also the frequency of occurrences of a high number of road users in the same frame is low (Fig 5.5), leading to some noise,because of insufficient training data.

6.4 Qualitative Analysis

The above section dealt with quantitatively comparing different models. It would also be interesting to assess the behaviors that these models try to capture. Therefore this section is dedicated to visualizing predictions made by these models, in an attempt to delineate and differentiate these models better. This was achieved by building a simulation module and manually visualizing certain IDs. Mostly, the analysis in this section deals with cyclists and pedestrians as we believe the models haven't understood shuttle patterns because of the limited data.

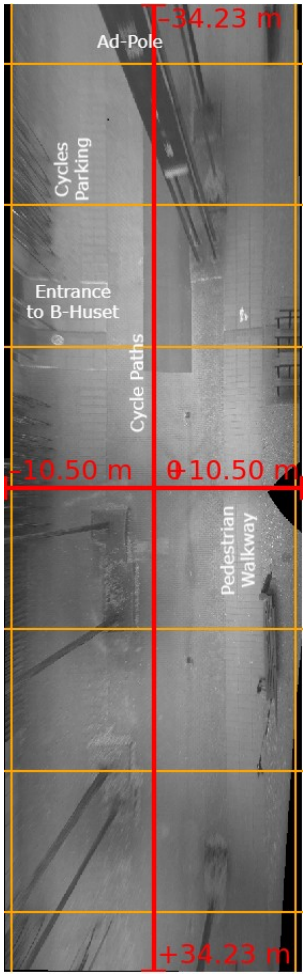
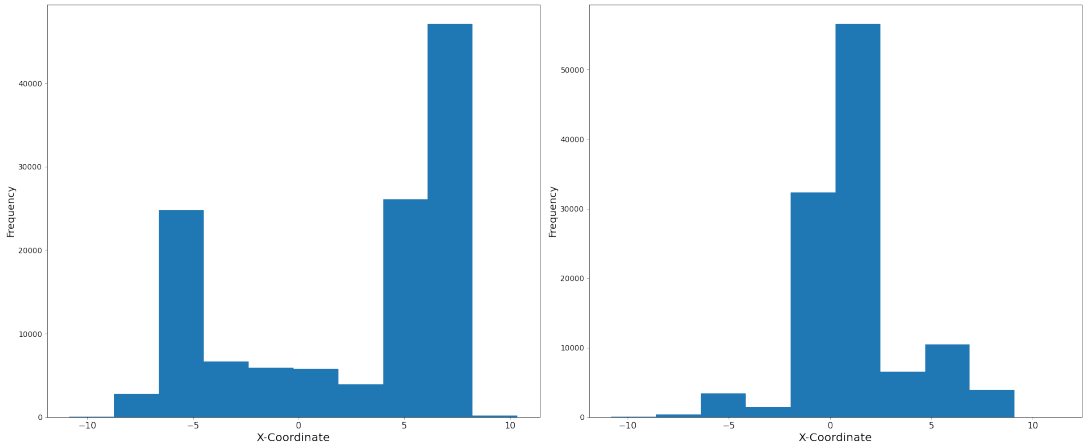


Figure 6.2: Map of the ground frame

For convenience, in all the figures above the vertical axis is called as the y-axis and the horizontal axis is called as the x-axis. Pedestrians and cyclists usually move longitudinally along the y-axis and in some instance take turns or slight maneuvers so that their x-coordinate also changes. Along the left end of the image there is an entrance to the building and cycles are parked along the left border of the image.



(a) Pedestrians (b) Cyclists

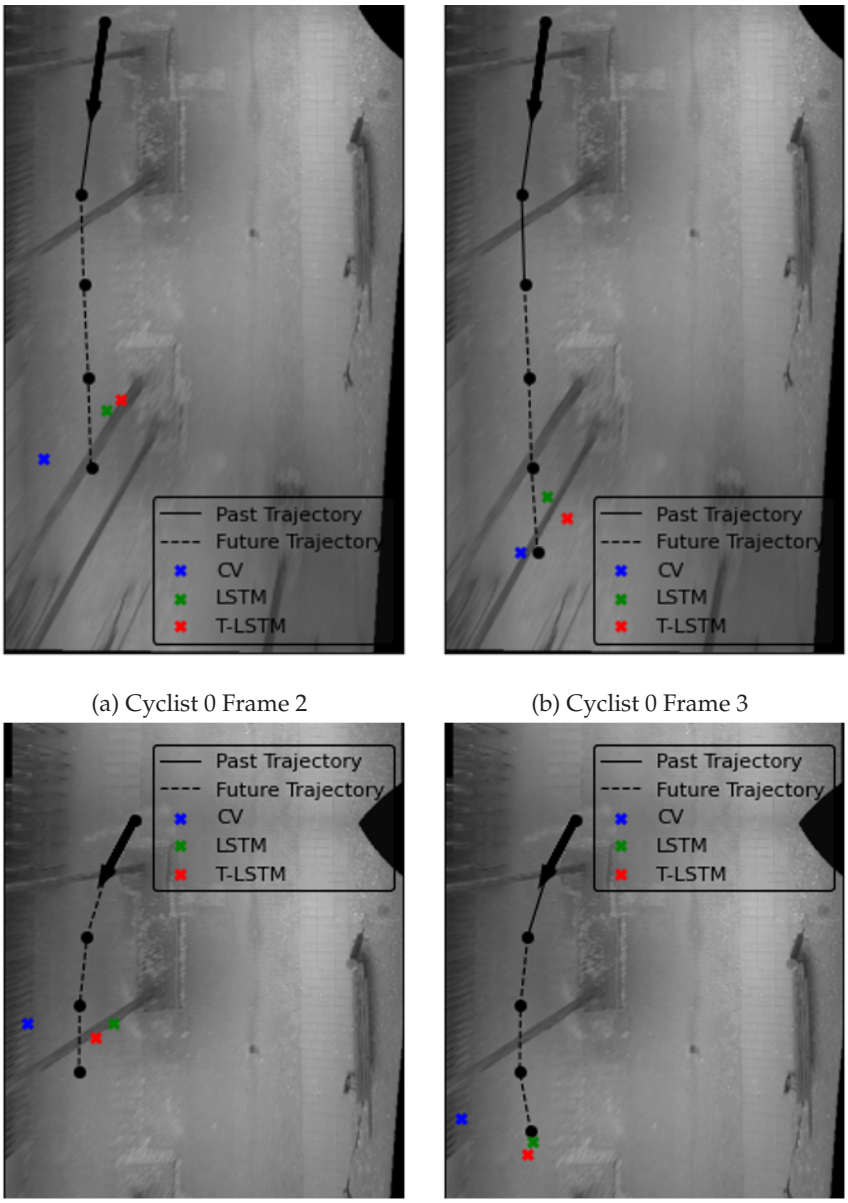
Figure 6.3: Distribution of x-coordinates for road users

As seen in the above image, cyclists generally move closer to the center of the axis. Pedestrians usually traverse on the ends of the frame, specifically on the right as there is a pedestrian walkway present there.

In all the figures from here on, the solid black line represents the past trajectory, the dotted-line represents the future trajectory. In some images, an arrow will be showcased to illustrate the start position and direction of the road user. The blue, green and red crosses represent the 3-step ahead prediction for the constant velocity model, vanilla LSTM network (referred to as the LSTM network) and the Transformer/Attention based LSTM network (referred to as the TLSTM network). For conveniences,each of these trajectories are also labelled with the type of road user (Cyclist, Pedestrian, Shuttle) and a unique ID (1,2,3..) for each road user to isolate and differentiate trajectories. Also the frame numbers added to the images indicate the moment from which predictions are being made. Frame 1 signifies the prediction made from the first 2 seconds of trajectory input, frame 2 signifies the prediction made from the first 3 seconds of input and so on. After individually studying many trajectories and running simulations of the videos, some behaviors were seen for the various models on the given data. It is to be pointed out that the trends seen here aren't a representative trend for the entire dataset and is rather a representative from a sample. These samples were obtained by looking at entries where one model's error is lower compared to the other models, so that the behaviors captured/missed by the models are assessed.

CV model vs LSTM based models

There are quite a few instances where the constant velocity models are not able to capture behaviors that are captured well by the LSTM based models. The first of these instances is when the constant velocity model is not able to capture self-corrections made by pedestrians and cyclists alike. This is when a pedestrian/cyclist is correcting his/her course, so that their trajectory is not obtuse and is parallel to one of the axes.



(a) Cyclist 0 Frame 2

(b) Cyclist 0 Frame 3

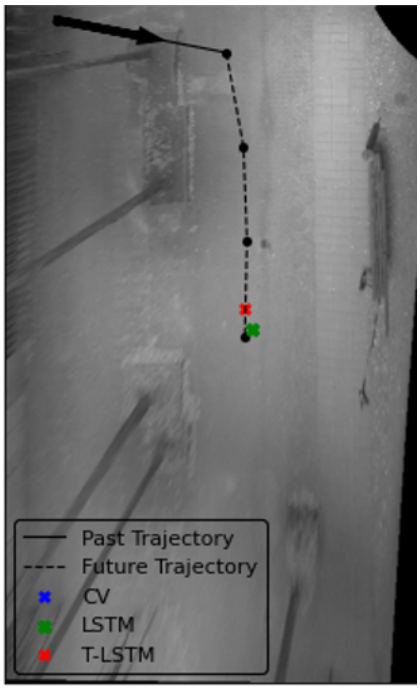
(c) Cyclist 1 Frame 1

(d) Cyclist 1 Frame 2

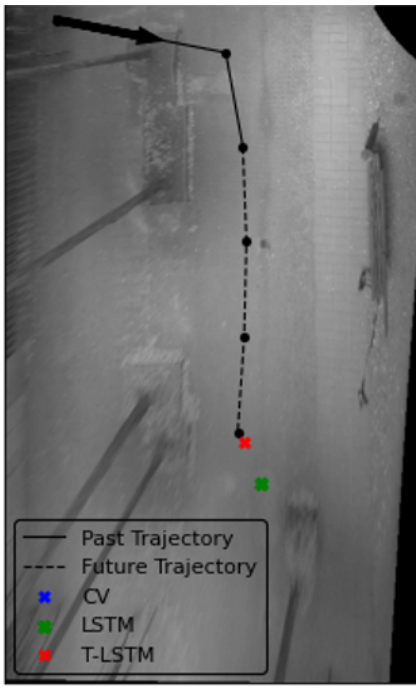
Figure 6.4: Course correction trajectories for road users

As can be seen from Fig. 6.4 usually pedestrians and cyclists tend to correct their course, so that they move parallel to the y-axis. This kind of behavior is not captured by a CV model as they tend to just extrapolate current state dynamics forward in time and don't recognize such spatial dependencies. This maneuver was frequently exhibited by pedestrians and cyclists alike and was well captured by the LSTM based networks.

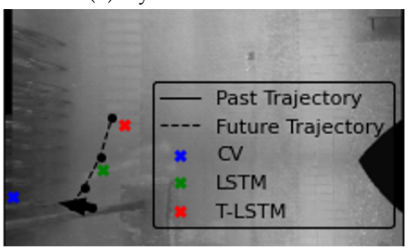
Another kind of maneuver observed in the dataset is a turn. A turn can be defined as a sharp change in direction of a road user. The self-corrections that were discussed above were acute changes in direction (this can be heuristically defined as $< 60^{\circ}$). A turn is an obtuse/significant change in direction ($> 60^{\circ}$). These turns are hard for even human observers to detect and now we will be looking at cases where the LSTM based models are able to detect turns that were otherwise undetected by the CV model.



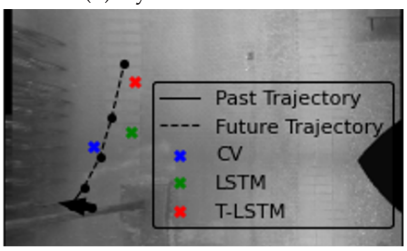
(a) Cyclist 2 -Frame 2



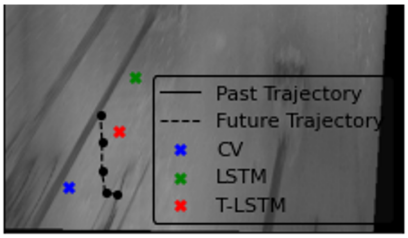
(b) Cyclist 2 - Frame 3



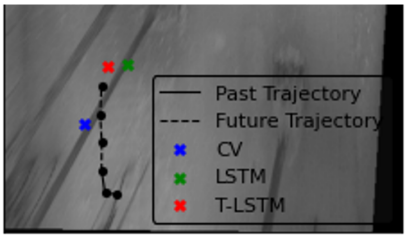
(c) Cyclist 3 Frame 2



(d) Cyclist 3 Frame 3



(e) Cyclist 4 -Frame 1

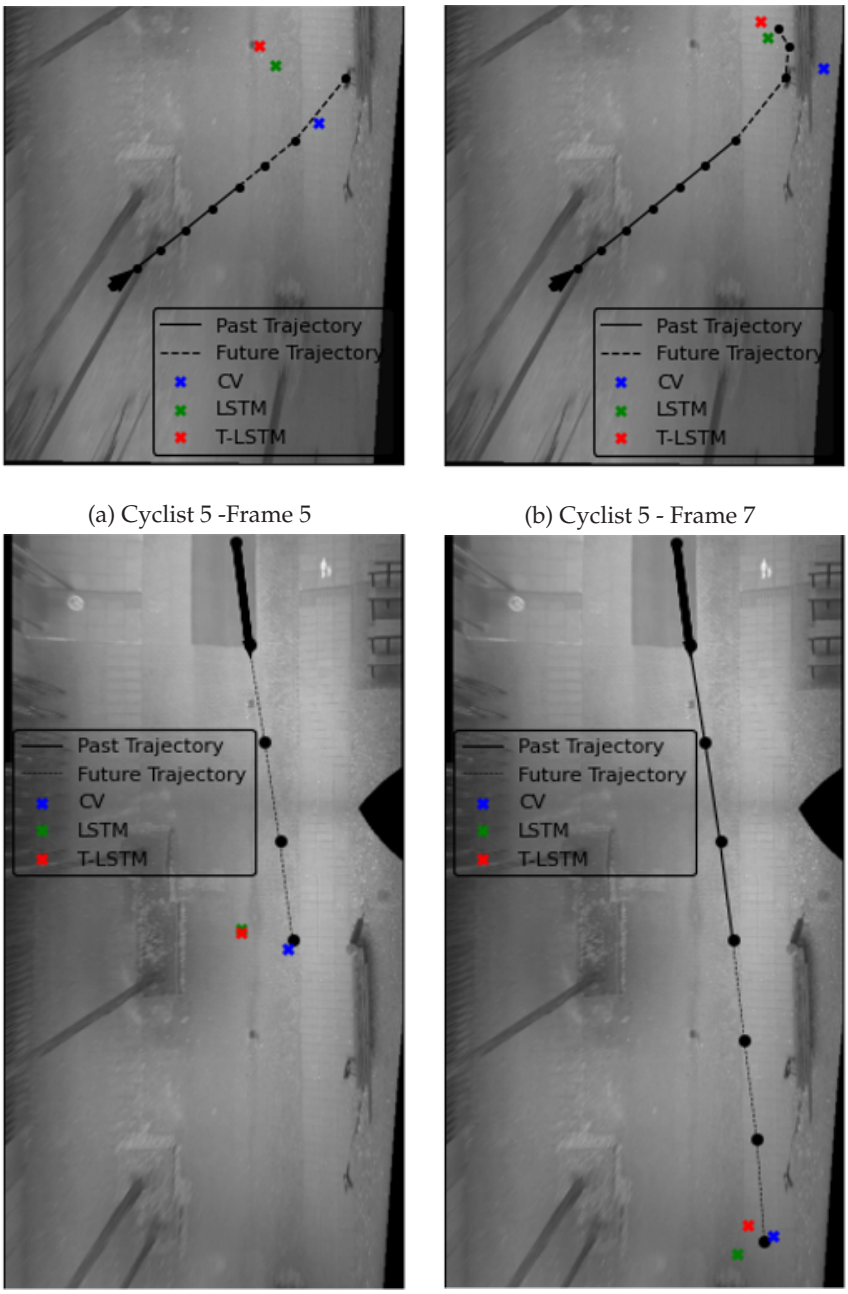


(f) Cyclist 4 - Frame 2

Figure 6.5: Turning maneuvers exhibited by road users

In Fig 6.5a and Fig. 6.5b , the cyclist takes a turn toward the right, which has been correctly predicted by the LSTM based networks. The network would have assessed that the road user needs to correct his/her trajectory. The movement of the cyclist towards the negative y-axis in the first two seconds may have indicated to the network that the cyclist will turn right. Note that the CV prediction is not present in both these figures, as it predicts the user to be far outside the frame (from just extrapolating state dynamics). Figures 6.5c 6.5d 6.5e and 6.5f on the other hand are more complicated turns. These turns are perpendicular or even obtuse. There is a hint albeit slight in both these cases that the user might go towards the positive y-axis, which may be helping the network predict these turns. There are again many such cases found in the data where a turn has been correctly predicted by both the LSTM networks.

Now, we will be looking at the few cases where the CV model has performed better than the two LSTM based networks.



(a) Cyclist 5 -Frame 5

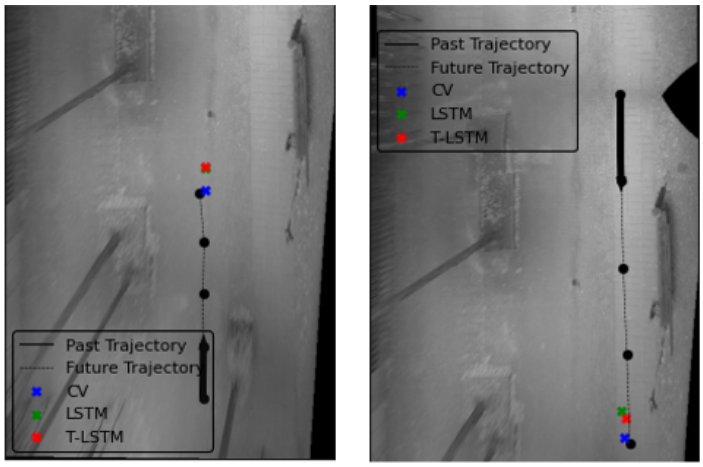
(b) Cyclist 5 - Frame 7

(c) Cyclist 6 Frame 1

(d) Cyclist 6 Frame 4

Figure 6.6: Trajectories where road users don't course correct

In these cases we notice that the road users don't course correct and these cases are unusual in the dataset. In such situations, it is noticed that the LSTM networks always try for course-correction at a suitable place and eventually as the past trajectory length gets longer in the same straight line, they go for a more conservative (and not immediate) course-correction close to the end of the frame.



(a) Cyclist 7 -Frame 1 (b) Cyclist 8 - Frame 1

Figure 6.7: Trajectories where velocities are estimated incorrectly by the LSTM based networks

The above figures represent some infrequent situations when the LSTM based network either overestimates or underestimates the velocity of cyclists. Cyclists usually show higher variance in their speed and as a result, their velocities are hard to estimate. In the above cases, the cyclist is travelling at a constant velocity and is thus captured well by the CV model. On the other hand the LSTM based networks add some noise as they have seen just two seconds of past trajectory data. However, it has to be pointed out that these are rare occurrences and don't represent the majority of cases where the LSTM based network either capture velocity better or to the same extent as the CV model.

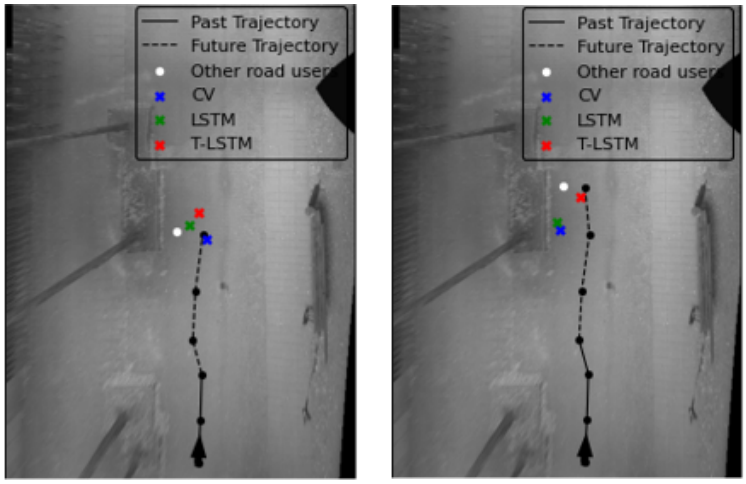
LSTM vs TLSTM

In the previous section, we compared qualitatively the behavior of a CV model vs neural network based models. In this section, we will look at comparing the two LSTM models to assess the behaviors that are captured by one and not another. In addition to the legends from the previous plots, the present plots also consists of white points. These white points represent the position of other road users in the scene *at the time of prediction*. This means that if the network was predicting the ego road user's position at time t using trajectory information till time $t-3$, then the white points represent all other road users present at time t. This adds another dimension to the image and conveys how close the ego-user's actual trajectory was to some other road users.

(a) Pedestrian 1 -Frame 1 (b) Pedestrian 2 - Frame 1 (c) Pedestrian 3 - Frame 1

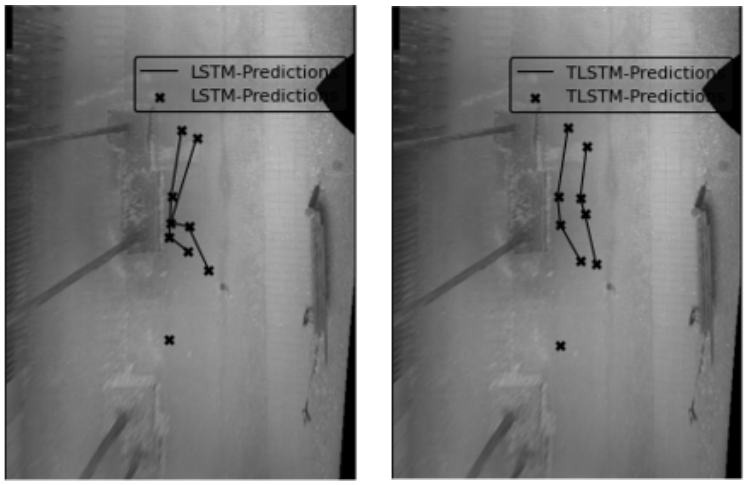
Figure 6.8: Turn trajectories correctly predicted by TLSTM

Figures 6.8b and 6.8c are turns at the entrance. After the first two seconds, the attention based network is correct in assessing that the pedestrian will turn left for both these cases. Another interesting observation is that this happens with other co-pedestrians as is represented from the white points. The same is the case with 6.8a where the pedestrian turns right and then moves towards the negative y-axis. It is to be pointed out again that many pedestrians move in groups in this data. These pedestrians usually have similar trajectories and maintain a very small distance from each other. These group movement patterns may be better assessed by the Attention based LSTM networks and hence they may be able to better predict turning maneuvers exhibited by groups of pedestrians.



(a) Cyclist 9 -Frame 2 (b) Cyclist 9 - Frame 3

Figure 6.9: Trajectories of one of the cyclists at different times

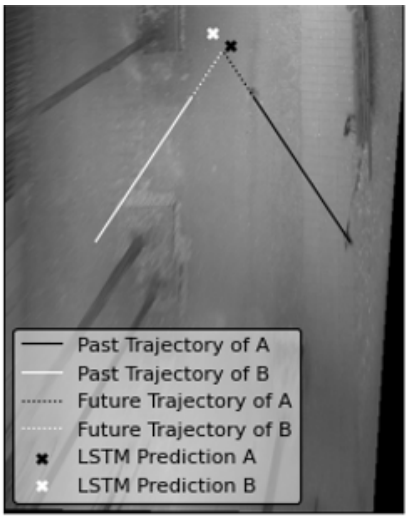


(a) LSTM predictions (b) TLSTM predictions

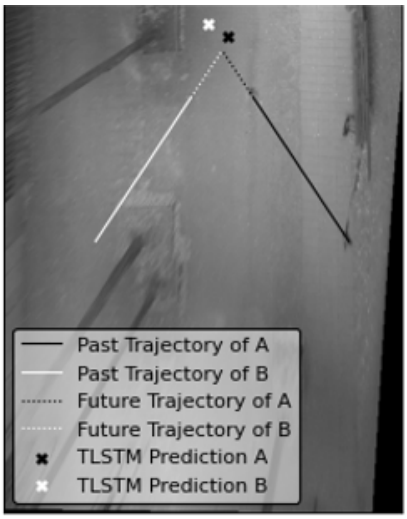
Figure 6.10: 3s ahead predictions for both cyclists (9&10) in the frame

In Figures 6.9a and 6.9b we clearly see that both the LSTM network and CV model predict the user to go somewhere left where there is another cyclist. On the other hand, the TLSTM network maintains some distance from the other road user on the left. Figures 6.10a and 6.10b refer to 3s ahead predictions made by both the LSTM and TLSTM network respectively for both cyclists in the frame. We notice that the LSTM predictions lead to some intersection in trajectories, whereas the TLSTM predictions keep the road users roughly par-

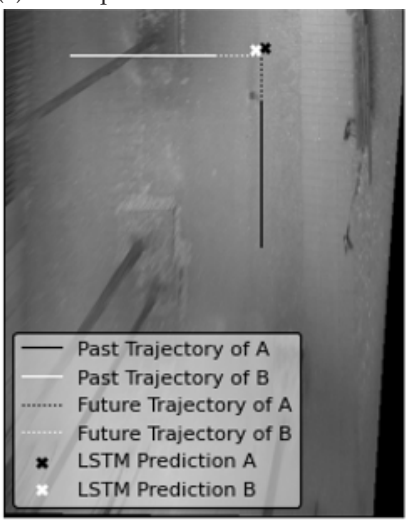
allel to each other. This maybe because the TLSTM network pays attention to the road user on the left and takes care to keep distance from that road user. In this dataset, head on collisions/intersections are rare as people usually move parallel to each other and in some cases maintain very less distance from each other. Therefore, we did a simulation of two pedestrians such that they are on a collision course. We compared the predictions of the LSTM and the TLSTM network to see where they are different.



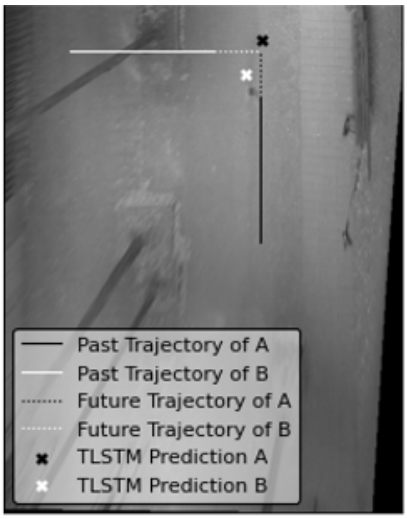
(a) LSTM prediction for acute collision



(b) TLSTM Prediction for acute collision



(c) LSTM Prediction for perpendicular collision



(d) TLSTM Prediction for perpendicular collision

Figure 6.11: Simulation of trajectory collision

It is noticed from Figures 6.11a and 6.11b that for an acute collision, both of them seem to course correct trajectories of A and B. The distance between the final predictions for A and B is slightly greater for the TLSTM network compared to a LSTM network. However, the more interesting observation is seen in 6.11c and 6.11d where the TLSTM network avoids collision between the two road users whereas the plain vanilla LSTM predicts both road users to be approximately in the same spot. Also, given below is the attention matrix extracted from the network when it was making the prediction. This is similar to the matrix explained in Fig 3.6. It can be seen that in both these cases, road users are giving each other some attention. In

the acute collision case, one of the road users gives the other road user lesser attention, even though it is expected to be symmetric for both these road users. In the perpendicular collision case, road user A gives much lesser attention to B than B gives to A. This can be reasoned on the basis that it is usually the responsibility of a person who is crossing (or moving on the lesser travelled direction) to pay attention to other road users on the main way.

Road User	A	B
A	0.92	0.08
B	0.63	0.37

Table 6.8: Attention for perpendicular collision

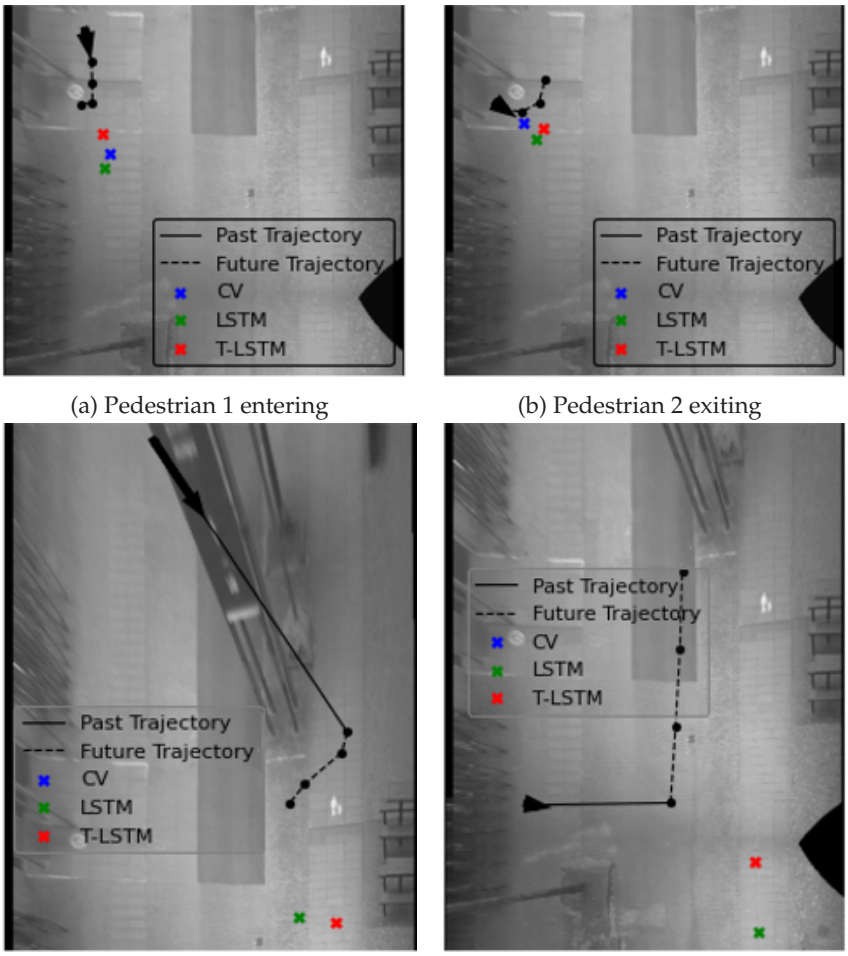
Road User	A	B
A	0.78	0.21
B	0.58	0.41

Table 6.9: Attention for acute collision

There were very few cases where the LSTM model was outperforming the TLSTM model. Even in such cases, both the models were far away from the actual predictions. Thus, the LSTM predictions could not be used as a gold standard. In conclusion the TLSTM model either performed better or similar to the LSTM model. However, on the general quantitative level there isn't a substantial lift in performance for the TLSTM model in comparison with the LSTM model for the following possible reasons:

- People mostly move across the y-axis parallel to each other, since the lane that we are studying, connects important destinations in the university. Barring one entrance and a small cycle parking, there is not much reason for the road user to move to the left/right. Therefore most trajectories are longitudinal and rarely involve road users intersecting. This is reflected in the quantitative error metrics which don't show substantially high lift.
- At the same time there are usually few road users in a large area (of more than $1400m^2$), owing to which, the density of road users is very low (Fig. 5.5) for the network to learn complex patterns.
- Also, as mentioned in Fig. 4.4 the data is noisy to understand crowd behaviors and this may affect the training of the TLSTM network to some degree. When there are more road users in the frame, the network has a higher ability to learn complex behavioral patterns. However, when the number of road users in the same frame increases, it is harder to isolate and detect these road users independently. This becomes a Catch-22 and therefore the best way to improve the model, is to improve data acquisition processes.

Cases where all models fail to capture behavior



(a) Pedestrian 1 entering

(b) Pedestrian 2 exiting

(c) Cyclist 11 - Frame 3

(d) Cyclist 12 - Frame 2

Figure 6.12: Undetected/wrongly detected road user maneuvers

Lastly, it is interesting to look at cases where all 3 of our models fail. Figure 6.12a illustrates a pedestrian entering the building and all three models don't suggest that the pedestrian will make a sharp turn. Figure 6.12b illustrates a pedestrian leaving the building and all three models predict again that the pedestrian is going to go right and instead the pedestrian goes left. Figures 6.12c and 6.12d represent situations when there are abrupt turns and none of the model expect these road user maneuvers. In Figs. 6.12d the LSTM networks expects the cyclist will take a right as the cyclist is moving with a high velocity, but he instead ends up taking a left.

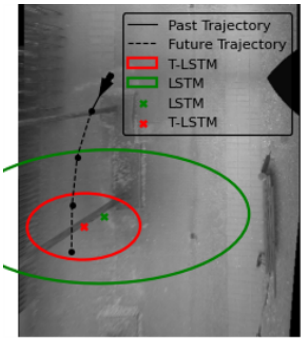
6.5 Networks with uncertainty

In this section, networks are modified to predict the means and covariances of a normal distribution. The predicted position is considered to be normally distributed with the predicted means and covariances. This would require modifying the loss function to minimize 5.7 rather than minimizing MSE. Since now the network has to predict 5 parameters for each road user instead of 2, the network takes longer to train and learn. Strictly speaking, a separate hyperparameter tuning needs to be performed for this uncertainty based network. However, in an effort to save computational time, these networks are trained with the best hyperpa-

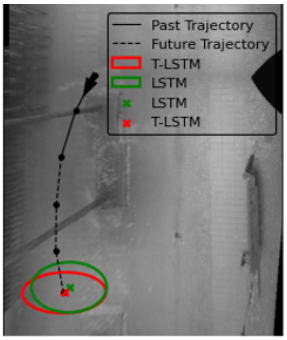
rameters obtained from 6.2. Average log-likelihood estimates are constructed using 6.1 and Table 6.10 compares the average log-likelihood across the three models. It can be seen that the TLSTM network has the highest average log-likelihood for all road users. However, it has to be mentioned that the TLSTM network contains the highest number of parameters, followed by the LSTM network. To gauge the effect of higher number of parameters, the LSTM model was built with higher number of parameters (more layers, larger hidden size). It was seen that the performance of the LSTM model was saturating, implying that higher parameters had little to no impact on model performance.

Road user \ Models	CV	LSTM	TLSTM
Pedestrian	-2.25	-0.95	**-0.71**
Cyclists	-3.36	-1.93	**-1.77**
Shuttles	-2.37	-1.63	**-1.51**
Total	-2.72	-1.36	**-1.16**

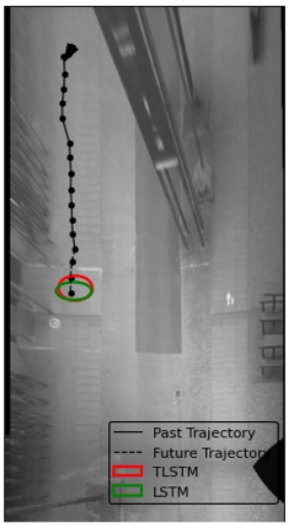
Table 6.10: Comparison of Average Log-Likelihood for 3s ahead predictions across different models and road users on test data



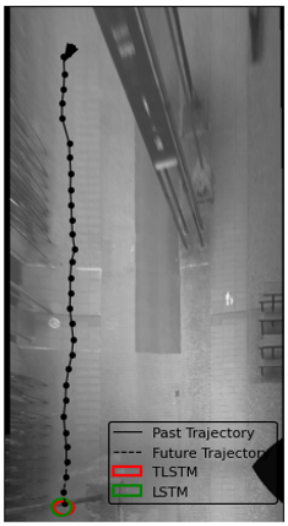
(a) Cyclist taking a turn- Frame 1



(b) Cyclist taking a turn- Frame 2



(c) Pedestrian - Frame 1



(d) Pedestrian - Frame 2

Figure 6.13: Uncertainty prediction bands for road users

Now, with the help of the mean and covariance matrix, one can visualize confidence intervals for positions. In the case of a bivariate distribution, these confidence intervals take

the form of ellipses as seen in the below diagram. In the Fig 6.13, 95% confidence ellipses for the LSTM and TLSTM predictions are displayed for various road users at specific times. Larger confidence ellipses signify higher uncertainty whereas smaller ellipses signify lower uncertainty. It is seen in Fig. 6.13a that during the beginning of a turn there is larger uncertainty and as the turn evolves in Fig. 6.13b, the uncertainty reduces. Also, in Fig. 6.13c the uncertainty is higher near the entrance of B-huset and reduces in Fig. 6.13d at a place far away from the entrance and the cycle parking.

6.6 Analysis of Attention

Attention matrices were briefly discussed through an example in Table 6.8 and Table 6.9. It would be interesting to look at attention throughout the entire dataset, so that attention behaviors can be generalized. For this reason, there are some additional metrics created in this dataset as follows:

- **Road user self attention**: At time t the trajectories of all road users are passed as input to the TLSTM network which predicts their positions at time $t + 3$. In the process of making this prediction, the attention matrices computed are reflective of the dependencies between road users at time step t. Road user self attention is the attention that a road user pays himself/herself. The lower the value of road user self attention implies higher attention paid to other road users in the scene. It is to be highlighted that the self attention referred to earlier (Section 3.6 and Section 5.3) is inter-trajectory/within sequence self attention. A sequence at time t is defined to consist of multiple trajectories and *self* here refers to the same sequence. The self attention matrix at time t for a collection of trajectories (or sequence) is of the form of Tables 6.8 and 6.9. However, in this section, road user self-attention refers to the attention a road user pays himself/herself. For all road users in a scene, this corresponds to the diagonal elements in Tables 6.8 and 6.9. In this section, all mentions of self attention refer to this road user self attention and not the entire self attention matrix.

- **Minimum Distance at the time of observation** : At time t, for a particular road user r, the closest road user's distance is calculated. This is the minimum of all distances between the concerned road user r and all other road users in the scene. If this distance is low, it implies that there are one (or more) road users in close proximity to the road user r. The analysis below excludes road users who do not have any other neighbor road users in the frame at a given timestamp.

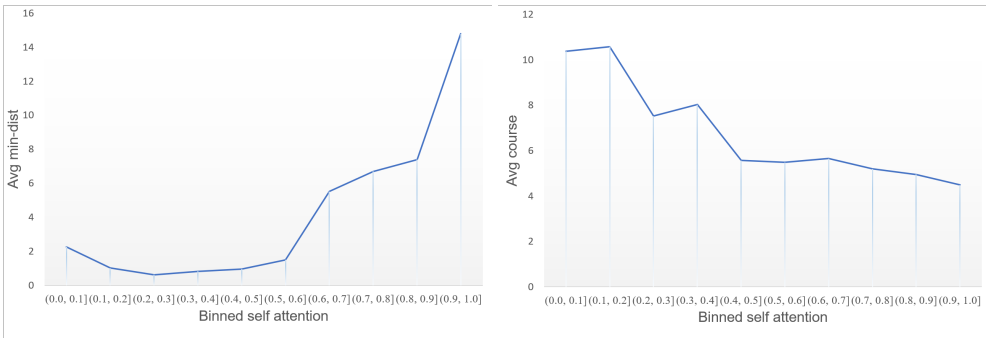
- **Course of a road user**: Let the position of a road user at time step t and $t-1$ be represented as (x_t, y_t) and (x_{t-1}, y_{t-1}) where x_i and y_i represents the x and y-coordinate of the road user at time i. Course is defined as the angle of this vector relative to the y-axis. Mathematically, it can be written as

$$Course = \arctan \frac{|x_t - x_{t-1}|}{|y_t - y_{t-1}|} \tag{6.2}$$

  The absolute differences are taken, as we disregard the sign of this angle. The higher the angle (closer to 90° implies that the road user is crossing the road and lower the angle (closer to 0°) implies that the road user is moving parallel to the y-axis on the main pathway.

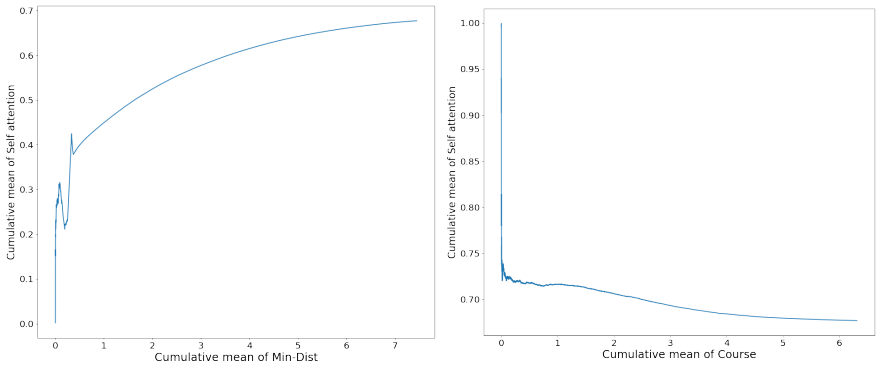
For convenience sake, we will refer to minimum distance at the time of observation as *Min_Dist* and the angle or course of a road user wrt to the y-axis as *Course*. Self attention is binned into 10 discrete bins such as [0-0.1),[0.1-0.2) and so on for all road users and mean values are calculated for *Min_Dist* and *Course* across these different bins. The resulting plot is displayed in Fig. 6.14a and Fig. 6.14b.

Also the entire dataset is sorted on the basis of *Min_Dist* and cumulative means of self attention is plotted against the cumulative means of *Min_Dist* and in Fig. 6.15b. Since, *Min_Dist* is sorted, the cumulative means of *Min_Dist* will be increasing. Against this increasing trend, we would like to gauge the trend of self-attention. A similar exercise is done in Fig. 6.15a by sorting the entire dataset on the basis of *Course* and plotting the cumulative mean of self-attention against the cumulative means of *Course*.



(a) Metric: Min-Dist (b) Metric: Course

Figure 6.14: Plots of average values of metric vs binned-self attention



(a) Metric: Min-Dist (b) Metric: Course

Figure 6.15: Cumulative mean plots of self-attention vs the sorted cumulative means of metrics

From Fig 6.14a, we observe that barring the first two bins, as self-attention increases, the average minimum distance at the time of observation increases. Also, a similar trend is seen in Fig. 6.15b where the cumulative self-attention mean seems to increase (except a small hiccup) as cumulative mean of $Min - Dist$ increases. It is generally seen that if there is no road user in the immediate vicinity of the concerned road user (*Min_Dist* is high) the self-attention is high. From these graphs, we can infer that the network is largely able to correlate proximity of a road user to attention. These graphs also indicate that having low values of self-attention (<0.2) does not necessarily mean that there is another road user in close proximity. The self-attention may have been low because of factors other than proximity (such as if the road user is crossing the road). It may also be the case that the network has some noise and is not able to adequately capture these interactions in some cases.

We observe from Fig. 6.14b that average angle in degrees seems to decrease as we go to higher self-attention bins. From Fig. 6.15a we again observe that self attention seems to sharply decrease when the cumulative mean of *Course* increases. Thus, the networks also seem to be correlating the course of a road user to attention. This means that when the road

user is crossing the street or moving at an incline towards the y-axis, the attention this road user pays other road users is high.

From an observer's standpoint, we would also expect that road users' attention to other road users depend on factors such as distance and the course of a road user. Generally, a road user would pay more attention to closer road users. We would also expect the road user to pay more attention when they are moving at an angle with respect to the main pathway. Therefore,the network is constructing attention matrices that seem to take into account explainable factors such as proximity and the course of a road user. The network may also be capturing more complex dependencies , but it is indeed encouraging that it captures dependencies that one would naturally expect.

7 Conclusion and Future work

This thesis focused on studying methods that analyze and predict the intent of road users on a university lane. Firstly, a Kalman filter constant velocity model was built that extrapolated state dynamics to predict future positions of road users. This constant velocity model was robust and worked pretty well in normal scenarios when road users continued in the same direction without much deviation. It also worked well when road users travelled at a constant pace and didn't change their pace often. A simpler auto-regressive model, which just considers p past positions and does not consider velocities, could also be another benchmark model that could have been implemented for this problem. We saw that the CV model failed when the road user made course corrections, or turns in their trajectories. The CV model also largely failed to accurately estimate velocity when road users changed their velocity abruptly. In summary, the CV model is still a useful model and benchmark which can be extended as follows:

- It can be used to detect anomalies in pedestrian behavior. For example, there may be a location where a particular type of road user turns away and later course corrects. This may be happening because of some problems in the spatial area such as an obstruction or bad design of the roadway. A heatmap of errors for this type of road user can be drawn across the spatial area, to identify such locations where everyone behaves unexpectedly and the problem can be corrected
- As mentioned in the Theory section, this model can also be extended to a constant acceleration model with relative ease. The parameters A, C in Eq. 5.1 can also be estimated through the EM algorithm and these changes will make the Kalman filter model more general and hopefully also more flexible
- The tracking employed by the sensors already use Kalman filters to obtain non-noisy estimates of position. The prediction module presented in this paper can easily be integrated into the same tracking software, so that the Kalman filter model both tracks and predicts positions online.

The plain LSTM based model was seen to perform substantially better than the CV model. It was successful in identifying course corrections. It learnt quickly that users move parallel to the y-axis and if given a chance try to correct their trajectories. The model was also able to identify some major turns because it utilized information from the entire past trajectory,

where hints from the past such as slowing down, or a slight change in direction was picked up by the LSTM network. It also understood the dynamics and velocity of user motion better than a CV model. However, the LSTM network did not account for other road users in the vicinity and this sometimes led to the network predicting collision of two trajectories. The plain LSTM model can be extended/improved in the following ways:

- The model can be trained on the 0.2s sampled data. It wasn't done in this thesis because of the computational resources required for training. As can be seen from the CV model, the predictions were better in the case of 0.2s sampled data for cyclists. The 0.2s sampled data will give a finer level of granularity in the road users' (especially cyclists') movements.
- The velocity obtained as a by-product of running the Kalman filter can be used as input for the LSTM network. If velocities along the x and y-axes for the entire past trajectory is fed as input, along with the positions, the network will have more features to learn from to make predictions. The network already must be learning the velocities, but by feeding it the velocities directly, the network can focus on learning more complex dynamics.
- Presently, only the future positions are predicted. We can also predict future trajectories which are positions at different times in the future. This is a typical sequence-to-sequence modeling framework and can be implemented by including a decoder which converts the interim hidden states of the LSTM cell into a sequence of future positions.
- The LSTM network can quite easily also be extended into the social-LSTM network proposed in [1]. The hidden state feature vectors obtained from the network can be combined for different users and predictions can be made for each road user independently. However, the proposed TLSTM model presents a different solution to the same problem.
- For the uncertainty based neural network, the final position of a road user was assumed to be normally distributed. Different distributions can be examined to gauge which distribution gives the best likelihood given the data.

The attention based LSTM borrowed the same ideas from an LSTM network, but tried to improve upon the plain LSTM network by modeling inter-user patterns. The attention based LSTM network marginally improved on the results from a plain LSTM with performance being the same/better compared to a plain LSTM. Some examples and simulations showed that this network adjusts the positions of each road user in-order to maintain some distance between them. The attention matrices showed us that the network is learning to give high attention to road users who are closer to each other. It is also able to correlate attention to the angle road users make with respect to the main walkway when they are travelling. The improvements mentioned for the plain LSTM network can also be implemented for this network. In addition to that, this attention based network can be improved as follows:

- The number of transformer encoders can be increased to increase the complexity of interactions captured. Also, a finer hyperparameter tuning must be performed over a larger hyperparameter space.
- As mentioned in the results section, some trajectories were not completely detected by the sensors. Also when the number of road users increase, detecting these road users becomes difficult. We believe that this network can learn interactions better with cleaner data. Therefore, importance should be given to data pre-processing to ensure that it represents the actual behaviors and trajectories.

- Lastly, with some more study of the attention module, it can be transferred to other geographies, to learn more behaviors and make use of the existing knowledge of behaviors.

Through this thesis, the attempt was to build up a solution for intent prediction in such a way that each new method covers for some deficiencies in the other older methods. In this way, there were increments in each new method. Also, by gauging the outputs of each methods, an attempt was made to understand what the models are "seeing". We believe that this work can be used as good starting point for studying the various aspects of intent prediction and with more detailed study, improving on the same.

8 Ethical Considerations

The toolchain that is responsible for acquiring trajectories is executed in an embedded computational unit. Therefore, images from cameras are removed directly after 3D reconstruction and classification (typically within 20 ms). Hence personal data is neither recorded nor transmitted. Therefore, the sensor is fully GDPR compliant and hence all the data used in this thesis is compliant with anonymity norms and regulations. Also, the process of acquiring data does not incur any harm for the environmental/living beings. Every part of this thesis is drafted with good and responsible ethics and none of it violates any ethical consideration to the best of my knowledge.

Bibliography

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. "Social LSTM: Human Trajectory Prediction in Crowded Spaces". en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 961–971. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.110. URL: http://ieeexplore.ieee.org/document/7780479/ (visited on 01/23/2022).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv:1409.0473 [cs, stat]* (May 2016). arXiv: 1409.0473. URL: http://arxiv.org/abs/1409.0473 (visited on 05/05/2022).

[3] Lorenzo Bertoni, Sven Kreiss, and Alexandre Alahi. "Perceiving Humans: from Monocular 3D Localization to Social Distancing". In: *arXiv:2009.00984 [cs]* (Mar. 2021). arXiv: 2009.00984. URL: http://arxiv.org/abs/2009.00984 (visited on 05/04/2022).

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. *Long Short-Term Memory-Networks for Machine Reading*. Tech. rep. arXiv:1601.06733. arXiv:1601.06733 [cs] type: article. arXiv, Sept. 2016. URL: http://arxiv.org/abs/1601.06733 (visited on 05/24/2022).

[5] Shengzhe Dai, Li Li, and Zhiheng Li. "Modeling Vehicle Interactions via Modified LSTM Models for Trajectory Prediction". In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 38287–38296. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2907000.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data Via the *EM* Algorithm". en. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (Sept. 1977), pp. 1–22. ISSN: 00359246. DOI: 10.1111/j.2517-6161.1977.tb01600.x. URL: https://onlinelibrary.wiley.com/doi/10.1111/j.2517-6161.1977.tb01600.x (visited on 04/27/2022).

[7] Hao-Shu Fang, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. "RMPE: Regional Multi-person Pose Estimation". In: *arXiv:1612.00137 [cs]* (Feb. 2018). arXiv: 1612.00137. URL: http://arxiv.org/abs/1612.00137 (visited on 05/04/2022).

[8] Tharindu Fernando, Simon Denman, Sridha Sridharan, and Clinton Fookes. "Soft + Hardwired Attention: An LSTM Framework for Human Trajectory Prediction and Abnormal Event Detection". In: *arXiv:1702.05552 [cs]* (Feb. 2017). arXiv: 1702.05552. URL: http://arxiv.org/abs/1702.05552 (visited on 05/05/2022).

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN". In: *arXiv:1703.06870 [cs]* (Jan. 2018). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870 (visited on 05/04/2022).

[10] Sepp Hochreiter. "Long Short Term Memory". In: (1997). URL: http://www.bioinf.jku.at/publications/older/2604.pdf.

[11] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[12] Stefan Hoermann, Daniel Stumper, and Klaus Dietmayer. "Probabilistic long-term prediction for autonomous vehicles". en. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. Los Angeles, CA, USA: IEEE, June 2017, pp. 237–243. ISBN: 978-1-5090-4804-5. DOI: 10.1109/IVS.2017.7995726. URL: http://ieeexplore.ieee.org/document/7995726/ (visited on 01/22/2022).

[13] "Kinematics, Newton's Laws of Motion". In: *Mechanics, Waves and Thermodynamics: An Example-based Approach*. Ed. by Sudhir Ranjan Jain. Cambridge IISc Series. Cambridge: Cambridge University Press, 2016, pp. 10–22. ISBN: 978-1-316-53523-3. DOI: 10.1017/CBO9781316535233.003. URL: https://www.cambridge.org/core/books/mechanics-waves-and-thermodynamics/kinematics-newtons-laws-of-motion/1F27E8FB139301A8D47D9EF12944A715 (visited on 05/26/2022).

[14] S. Julier and J. Uhlmann. "New extension of the Kalman filter to nonlinear systems". en. In: *undefined* (1997). URL: https://www.semanticscholar.org/paper/New-extension-of-the-Kalman-filter-to-nonlinear-Julier-Uhlmann/a8b141556f2dd7694e5f6343f8ce3650f8ca5b60 (visited on 05/05/2022).

[15] Vasiliy Karasev, Alper Ayvaci, Bernd Heisele, and Stefano Soatto. "Intent-aware long-term prediction of pedestrian motion". en. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. Stockholm, Sweden: IEEE, May 2016, pp. 2543–2549. ISBN: 978-1-4673-8026-3. DOI: 10.1109/ICRA.2016.7487409. URL: http://ieeexplore.ieee.org/document/7487409/ (visited on 05/04/2022).

[16] Christoph G. Keller and Dariu M. Gavrila. "Will the Pedestrian Cross? A Study on Pedestrian Path Prediction". en. In: *IEEE Transactions on Intelligent Transportation Systems* 15.2 (Apr. 2014), pp. 494–506. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2013.2280766. URL: http://ieeexplore.ieee.org/document/6632960/ (visited on 05/04/2022).

[17] S. Kohler, M. Goldhammer, K. Zindler, K. Doll, and K. Dietmeyer. "Stereo-Vision-Based Pedestrian's Intention Detection in a Moving Vehicle". English. In: vol. 2015-October. 2015, pp. 2317–2322. ISBN: 978-1-4673-6595-6. DOI: 10.1109/ITSC.2015.374.

[18] Shiyu Liang and R. Srikant. *Why Deep Neural Networks for Function Approximation?* Tech. rep. arXiv:1610.04161. arXiv:1610.04161 [cs] type: article. arXiv, Mar. 2017. DOI: 10.48550/arXiv.1610.04161. URL: http://arxiv.org/abs/1610.04161 (visited on 05/24/2022).

[19] Bingbin Liu, Ehsan Adeli, Zhangjie Cao, Kuan-Hui Lee, Abhijeet Shenoi, Adrien Gaidon, and Juan Carlos Niebles. "Spatiotemporal Relationship Reasoning for Pedestrian Intent Prediction". In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020). Conference Name: IEEE Robotics and Automation Letters, pp. 3485–3492. ISSN: 2377-3766. DOI: 10.1109/LRA.2020.2976305.

[20] Jinxin Liu, Yugong Luo, Zhihua Zhong, Keqiang Li, Heye Huang, and Hui Xiong. "A probabilistic architecture of long-term vehicle trajectory prediction for autonomous driving". en. In: *Engineering* (Mar. 2022). ISSN: 2095-8099. DOI: 10.1016/j.eng.2021.12.020. URL: https://www.sciencedirect.com/science/article/pii/S2095809922001412 (visited on 05/04/2022).

[21] E. Mazor, A. Averbuch, Y. Bar-Shalom, and J. Dayan. "Interacting multiple model methods in target tracking: a survey". In: *IEEE Transactions on Aerospace and Electronic Systems* 34.1 (Jan. 1998). Conference Name: IEEE Transactions on Aerospace and Electronic Systems, pp. 103–123. ISSN: 1557-9603. DOI: 10.1109/7.640267.

[22] Risto Miikkulainen. "Simple Recurrent Network". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 906–906. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_762. URL: https://doi.org/10.1007/978-0-387-30164-8_762.

[23] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training Recurrent Neural Networks". In: *arXiv:1211.5063 [cs]* (Feb. 2013). arXiv: 1211.5063. URL: http://arxiv.org/abs/1211.5063 (visited on 04/28/2022).

[24] Haziq Razali, Taylor Mordan, and Alexandre Alahi. "Pedestrian intention prediction: A convolutional bottom-up multi-task approach". en. In: *Transportation Research Part C: Emerging Technologies* 130 (Sept. 2021), p. 103259. ISSN: 0968-090X. DOI: 10.1016/j.trc.2021.103259. URL: https://www.sciencedirect.com/science/article/pii/S0968090X21002710 (visited on 01/20/2022).

[25] H. Robbins and S. Monro. "A stochastic approximation method". In: *Annals of Mathematical Statistics* 22 (1951), pp. 400–407.

[26] Kenshi Saho. "Kalman Filter for Moving Object Tracking: Performance Analysis and Filter Design". In: (2017). URL: https://www.intechopen.com/chapters/57673.

[27] Khaled Saleh, Mohammed Hossny, and Saeid Nahavandi. "Early intent prediction of vulnerable road users from visual attributes using multi-task learning network". In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Oct. 2017, pp. 3367–3372. DOI: 10.1109/SMC.2017.8123150.

[28] Khaled Saleh, Mohammed Hossny, and Saeid Nahavandi. "Intent prediction of vulnerable road users from motion trajectories using stacked LSTM network". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. ISSN: 2153-0017. Oct. 2017, pp. 327–332. DOI: 10.1109/ITSC.2017.8317941.

[29] Khaled Saleh, Mohammed Hossny, and Saeid Nahavandi. "Long-Term Recurrent Predictive Model for Intent Prediction of Pedestrians via Inverse Reinforcement Learning". In: *2018 Digital Image Computing: Techniques and Applications (DICTA)*. Dec. 2018, pp. 1–8. DOI: 10.1109/DICTA.2018.8615854.

[30] Nicolas Schneider and Dariu M. Gavrila. "Pedestrian Path Prediction with Recursive Bayesian Filters: A Comparative Study". en. In: *Pattern Recognition*. Ed. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Joachim Weickert, Matthias Hein, and Bernt Schiele. Vol. 8142. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 174–183. DOI: 10.1007/978-3-642-40602-7_18. URL: http://link.springer.com/10.1007/978-3-642-40602-7_18 (visited on 04/28/2022).

[31] Christoph Schöller, Vincent Aravantinos, Florian Lay, and Alois Knoll. *What the Constant Velocity Model Can Teach Us About Pedestrian Motion Prediction*. Tech. rep. arXiv:1903.07933. arXiv:1903.07933 [cs] type: article. arXiv, Jan. 2020. URL: http://arxiv.org/abs/1903.07933 (visited on 05/19/2022).

[32] Shikhar Sharma, Ryan Kiros, and Ruslan Salakhutdinov. "Action Recognition using Visual Attention". In: *arXiv:1511.04119 [cs]* (Feb. 2016). arXiv: 1511.04119. URL: http://arxiv.org/abs/1511.04119 (visited on 05/05/2022).

[33] R.H. Shumway. “An approach to time series smoothing and forecasting using the EM algorithm”. In: (1982). URL: https://www.stat.pitt.edu/stoffer/dss_files/em.pdf.

[34] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications*. Springer, 2000.

[35] Bogdan Ilie Sighencea, Rareș Ion Stanciu, and Cătălin Daniel Căleanu. “A Review of Deep Learning-Based Methods for Pedestrian Trajectory Prediction”. en. In: *Sensors* 21.22 (Jan. 2021). Number: 22 Publisher: Multidisciplinary Digital Publishing Institute, p. 7543. ISSN: 1424-8220. DOI: 10.3390/s21227543. URL: https://www.mdpi.com/1424-8220/21/22/7543 (visited on 05/05/2022).

[36] *The unscented Kalman filter for nonlinear estimation | IEEE Conference Publication | IEEE Xplore*. URL: https://ieeexplore.ieee.org/document/882463 (visited on 05/05/2022).

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. “Attention Is All You Need”. In: *arXiv:1706.03762 [cs]* (Dec. 2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (visited on 04/28/2022).

[38] WHO. *Road Traffic Injuries*. URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

[39] Jürgen Wiest, Matthias Höffken, Ulrich Kreßel, and Klaus Dietmayer. “Probabilistic trajectory prediction with Gaussian mixture models”. In: *2012 IEEE Intelligent Vehicles Symposium*. ISSN: 1931-0587. June 2012, pp. 141–146. DOI: 10.1109/IVS.2012.6232277.

[40] Wikipedia. *Waymo is now allowed to transport passengers in its self driving cars*. URL: https://en.wikipedia.org/wiki/Waymo#:~:text=On%20October%2030%2C%202018%2C%20the,roads%20and%20highways%20in%20California..

[41] Hao Wu, Wenjun Xu, Bitao Yao, Yang Hu, and Hao Feng. “Interacting Multiple Model-Based Adaptive Trajectory Prediction for Anticipative Human Following of Mobile Industrial Robot”. en. In: *Procedia Computer Science* 176 (2020), pp. 3692–3701. ISSN: 18770509. DOI: 10.1016/j.procs.2020.09.330. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877050920322456 (visited on 05/04/2022).

[42] Long Xin, Pin Wang, Ching-Yao Chan, Jianyu Chen, Shengbo Eben Li, and Bo Cheng. “Intention-aware Long Horizon Trajectory Prediction of Surrounding Vehicles using Dual LSTM Networks”. In: *arXiv:1906.02815 [cs, stat]* (June 2019). arXiv: 1906.02815. URL: http://arxiv.org/abs/1906.02815 (visited on 05/05/2022).

[43] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: *arXiv:1502.03044 [cs]* (Apr. 2016). arXiv: 1502.03044. URL: http://arxiv.org/abs/1502.03044 (visited on 05/05/2022).