AI-based prediction of road users' intents and reactions

Akshay Gurudath

Supervisor: Per Sidén

April 29, 2022

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[16]. 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 [17], 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, 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 against each other 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 manipulations to the raw dataset. Constant velocity[9] and acceleration models are very popular in the 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 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[13]. Through proper scoring rules [3] and other 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 to account for spatial dependency on behavior of road users. This is then built up to a slightly more complex pooled LSTM model to account for social dependence of other road users. For this we use Attention to pool and weigh the outputs of several LSTM networks. We then conclude our work by assessing where the models fail in predicting behavior and also try to gauge why the models are not able to capture certain road behavior.

1.3 Uniqueness

There has been extensive research done on intent prediction in the past [7][10][1][11] . However, many of these studies lack large objective datasets on the interaction of road users with autonomous shuttles and small delivery vehicles. There are two 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 moving in-vehicle sensors are used. In-turn our data is collected through a stationary infrastructure sensor which collects sufficiently long trajectories for each road user.
- Also, many of these experiments 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.

Both of these differences have certain associated advantages and disadvantages in terms of intent prediction as will be discussed later.

2 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.

2.1 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.

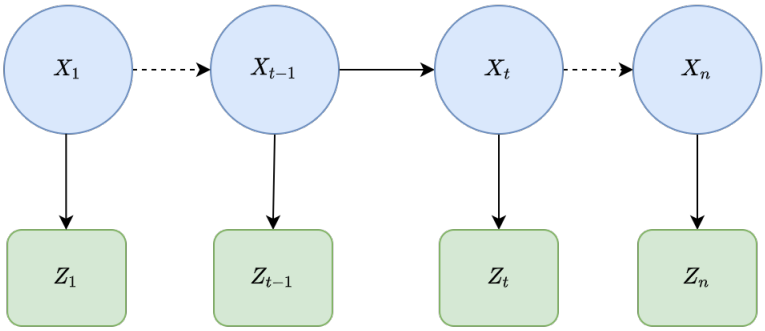


Figure 1: An illustration of the state space model

In the above figure 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 depen on past history. On the other hand, Z_t is independent of $Z_{0:t-1}$ given X_t . The model can entirely described using the following two equations:

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

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 state and observed state respectively. The covariance matrices Q and R represent the state space transition covariance matrix and the emission state covariance matrix.

2.1.1 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 $Y_{1:n}$ till 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:

$$X_t^n = E(X_t|Y_n) \tag{2}$$

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 :

$$P_t^n = E((X_t - X_t^n)(X_t - X_t^n)') \tag{3}$$

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

$$P_{t_1,t_2}^n = E((X_t - X_{t_1}^n)(X_t - X_{t_2}^n)') \tag{4}$$

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

$$P_{t,t-1}^n = E((X_t - X_t^n)(X_{t-1} - X_{t-1}^n)') \tag{5}$$

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.

2.1.2 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 [14].

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

$$\begin{aligned} X_t^{t-1} &= AX_{t-1}^{t-1} \\ P_t^{t-1} &= AP_{t-1}^{t-1}A' + Q \end{aligned} \tag{6}$$

with,

$$\begin{aligned} X_t^t &= X_t^{t-1} + K_t(Z_t - CX_t^{t-1}) \\ P_t^t &= [I - K_tC]P_t^{t-1} \end{aligned} \tag{7}$$

with,

$$K_t = P_t^{t-1}C'[CP_t^{t-1}C' + R]^{-1} \tag{8}$$

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 6. Using 7 and 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 $t > n$, 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 6, 7 and 8 please refer to [14].

2.1.3 Smoothed Estimates

For the model specified in 1 with initial conditions X_n^n and P_n^n obtained through 6, 7 and 8, for $t = n, n-1, ...1$,

$$\begin{aligned} X_{t-1}^n &= X_{t-1}^{t-1} + J_{t-1}(X_t^n - X_t^{t-1}), \\ P_{t-1}^n &= P_{t-1}^{t-1} + J_{t-1}(P_t^n - P_t^{t-1})J_{t-1}', \end{aligned} \tag{9}$$

where

$$J_{t-1} = P_{t-1}^{t-1}A'[P_t^{t-1}]^{-1} \tag{10}$$

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 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

$$P_{n,n-1}^n = (I - K_nC)AP_{n-1}^{n-1} \tag{11}$$

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

$$P_{t-1,t-2}^n = P_{t-1}^{t-1}J_{t-2}' + J_{t-1}(P_{t,t-1}^n - AP_{t-1}^{t-1})J_{t-2}' \tag{12}$$

Again, the proof for all of these properties can be found in [14].

2.2 Constant Velocity Model

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

$$\begin{aligned} x_t &= x_{t-1} + \dot{x}_{t-1}\Delta t + \eta_t' \\ \dot{x}_t &= \dot{x}_{t-1} + \eta_t'' \end{aligned} \tag{13}$$

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 $E[\dot{x}_{t+1}], E[\dot{x}_{t+2}], ..., E[\dot{x}_n] = \dot{x}_t$.

$$Z_{x_t} = x_t + \epsilon_t \tag{14}$$

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:

$$\begin{aligned} X_t &= AX_{t-1} + \eta_t \\ Z_t &= CX_t + \epsilon_t \end{aligned} \tag{15}$$

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 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.

2.2.1 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, we can approximate the next state $x_t, \dot{x}_t, y_t, \dot{y}_t$ as follows:

$$\begin{aligned} x_t &= x_{t-1} + \dot{x}_{t-1}\Delta t + \frac{\Delta t^2}{2}\eta_{x,t-1} \\ y_t &= y_{t-1} + \dot{y}_{t-1}\Delta t + \frac{\Delta t^2}{2}\eta_{y,t-1} \\ \dot{x}_t &= \dot{x}_{t-1} + \Delta t \eta_{x,t-1} \\ \dot{y}_t &= \dot{y}_{t-1} + \Delta t \eta_{y,t-1} \end{aligned} \tag{16}$$

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 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

$$\begin{aligned} Q &= E(\eta_t \eta_t^T) \\ &= E(\begin{pmatrix} \frac{\Delta t^2}{2}\eta_{x,t} \\ \eta_{x,t} \\ \frac{\Delta t^2}{2}\eta_{y,t} \\ \eta_{y,t} \end{pmatrix} \begin{pmatrix} \frac{\Delta t^2}{2}\eta_{x,t} & \eta_{x,t} & \frac{\Delta t^2}{2}\eta_{y,t} & \eta_{y,t} \end{pmatrix}) \\ &= \begin{pmatrix} \frac{\Delta t^4}{4}\sigma^2_{\eta_x} & \frac{\Delta t^3}{2}\sigma^2_{\eta_x} & 0 & 0 \\ \frac{\Delta t^3}{2}\sigma^2_{\eta_x} & \sigma^2_{\eta_x} & 0 & 0 \\ 0 & 0 & \frac{\Delta t^4}{4}\sigma^2_{\eta_y} & \frac{\Delta t^3}{2}\sigma^2_{\eta_y} \\ 0 & 0 & \frac{\Delta t^3}{2}\sigma^2_{\eta_y} & \sigma^2_{\eta_y} \end{pmatrix} \end{aligned} \tag{17}$$

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

$$Q = \sigma_\eta^2 \begin{pmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & 0 & 0 \\ \frac{\Delta t^3}{2} & 1 & 0 & 0 \\ 0 & 0 & \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} \\ 0 & 0 & \frac{\Delta t^3}{2} & 1 \end{pmatrix} \tag{18}$$

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:

$$R = \begin{pmatrix} \sigma_{\epsilon x}^2 & 0 \\ 0 & \sigma_{\epsilon y}^2 \end{pmatrix} \tag{19}$$

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 16 are known. However, in this paper, we aim to estimate some parameters such as these covariance matrices with the help of data. Then the assumption that noise affects the dynamics of motion in a specific way as in 16, can be disregarded. This also gives us more ability in our design to choose the best parameters through data-centric methods.

2.3 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:

$$\begin{aligned} X_t &= AX_{t-1} + \eta_t \\ Z_t &= CX_t + \epsilon_t \end{aligned} \tag{20}$$

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

$$\begin{aligned} \mu_0 &= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\ P_0 &= \begin{pmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{pmatrix} \end{aligned} \tag{21}$$

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:

$$\begin{aligned} -2 \log p(X_{0:n}, Z_{1:n}) = n \log |Q| &+ \sum_{i=1}^{n} (X_t - AX_{t-1})' Q^{-1} (X_t - AX_{t-1}) \\ + n \log |R| &+ \sum_{i=1}^{n} (Z_t - CX_t)' R^{-1} (Z_t - CX_t) \\ + \log |\Sigma_0| &+ (X_0 - \mu_0)' \Sigma_0^{-1} (X_0 - \mu_0) + constant \end{aligned} \tag{22}$$

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 2.1.2 2.1.3. At this stage, ideas from the Expectation Maximization (EM) [2] 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 [2] that the EM algorithm converges to the maximum likelihood estimate . Therefore, instead of calculating the complete 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

2.4 Expectation and Maximization step

2.4.1 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:

$$\begin{aligned}
-2E[\log p(X_{0:n}, Z_{1:n})|\phi_{j-1}, Z_{1:n}] &= n\log|Q| + n\log|R| + \log|\Sigma_0| \\
&+ E[\sum_{i=1}^{n}(X_t - AX_{t-1})'Q^{-1}(X_t - AX_{t-1})|\phi_{j-1}, Z_{1:n}] \\
&+ E[\sum_{i=1}^{n}(Z_t - CX_t)'R^{-1}(Z_t - CX_t)|\phi_{j-1}, Z_{1:n}] \\
&+ E[(X_0 - \mu_0)'\Sigma_0^{-1}(X_0 - \mu_0)|\phi_{j-1}, Z_{1:n}] + constant
\end{aligned} \tag{23}$$

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 equation 23 as follows:

$$\begin{aligned}
-2E[\log p(X_{0:n}, Z_{1:n})] &= n\log|Q| + n\log|R| + \log|\Sigma_0| \\
&+ E[tr(\sum_{i=1}^{n}(X_t - AX_{t-1})'Q^{-1}(X_t - AX_{t-1})] \\
&+ E[tr(\sum_{i=1}^{n}(Z_t - CX_t)'R^{-1}(Z_t - CX_t)] \\
&+ E[tr((X_0 - \mu_0)'\Sigma_0^{-1}(X_0 - \mu_0)] + constant
\end{aligned} \tag{24}$$

All the expectations above 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:

$$\begin{aligned} -2E[\log p(X_{0:n}, Z_{1:n})] &= n \log |Q| + n \log |R| + \log |\Sigma_0| \\ &+ tr(Q^{-1} E[\sum_{i=1}^{n} (X_t - AX_{t-1})(X_t - AX_{t-1})']) \\ &+ tr(R^{-1} E[\sum_{i=1}^{n} (Z_t - CX_t)(Z_t - CX_t)']) \\ &+ tr(\Sigma_0^{-1} E[(X_0 - \mu_0)(X_0 - \mu_0)']) + constant \end{aligned} \tag{25}$$

After simplifying, this can be written as:

$$\begin{aligned} -2E[\log p(X_{0:n}, Z_{1:n})] &= n \log |Q| + n \log |R| + \log |\Sigma_0| \\ &+ tr(Q^{-1} \sum_{i=1}^{n} (E(X_t X_t' - AX_{t-1}X_t' - X_t X_{t-1}' A' + AX_{t-1}X_{t-1}' A'))) \\ &+ tr(R^{-1} \sum_{i=1}^{n} (Z_t Z_t' - E[CX_t Z_t' + Z_t X_t' C' - CX_t X_t' C'])) \\ &+ tr(\Sigma_0^{-1} (\mu_0 \mu_0' + E[X_0 X_0' - \mu_0 X_0' - X_0 \mu_0'])) + constant \end{aligned} \tag{26}$$

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 equations 4 and 5, we can express $E(X_t X_t'), E(X_t X_{t-1})$ as:

$$\begin{aligned} E(X_t X_t') &= P_t^m + X_t^m X_t^{m\prime} \\ E(X_t X_{t-1}) &= P_{t,t-1}^m + X_t^m (X_{t-1}^m)' \end{aligned} \tag{27}$$

Substituting this in the above equation we get

$$\begin{aligned} -2E[\log p(X_{0:n}, Z_{1:n})] &= n \log |Q| + n \log |R| + \log |\Sigma_0| \\ &+ tr(Q^{-1} [S_{11} - AS_{10}' - S_{10} A' + AS_{00} A']) \\ &+ tr(R^{-1} (M_{ZZ} + M_{XZ} - M_{XX})) \\ &+ tr(\Sigma_0^{-1} (\mu_0 \mu_0' + E[X_0 X_0' - \mu_0 X_0' - X_0 \mu_0'])) + constant \end{aligned} \tag{28}$$

where

$$\begin{aligned} S_{11} &= \sum_{t=1}^{n} (X_t^n (X_t^n)' + P_t^n) \\ S_{10} &= \sum_{t=1}^{n} (X_t^n (X_{t-1}^n)' + P_{t,t-1}^n) \\ S_{00} &= \sum_{t=1}^{n} (X_{t-1}^n (X_{t-1}^n)' + P_{t-1}^n) \end{aligned} \tag{29}$$

and

$$
\begin{aligned}
M_{ZZ} &= \sum_{i=1}^{n} (Z_t Z_t') \\
M_{XZ} &= \sum_{i=1}^{n} Z_t (X_t^n)' C' - C X_t^n Z_t' \\
M_{XX} &= \sum_{i=1}^{n} C P_t^n C' + C X_t^n (X_t^n)' C'
\end{aligned}
\tag{30}
$$

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 2.1.2, 2.1.3. Once these smoothing steps are substituted, one can obtain update equations for Q and R as follows:

Equation 28can be differentiated with respect to Q^{-1} and R^{-1} to obtain:

$$
\begin{aligned}
\frac{\partial - 2E[\log p(X_{0:n}, Z_{1:n})]}{\partial Q^{-1}} &= nQ - [S_{11} - AS_{10}' - S_{10}A' + AS_{00}A'] \\
\frac{\partial - 2E[\log p(X_{0:n}, Z_{1:n})]}{\partial R^{-1}} &= nR - [M_{ZZ} + M_{XZ} - M_{XX}]
\end{aligned}
\tag{31}
$$

$$
\begin{aligned}
Q &= \frac{[S_{11} - AS_{10}' - S_{10}A' + AS_{00}A']}{n} \\
R &= \frac{[M_{ZZ} + M_{XZ} - M_{XX}]}{n}
\end{aligned}
\tag{32}
$$

The values of Q and R can be initialized with the heuristic covariance matrices obtained in 2.2.1, by setting σ_η^2, $\sigma_{\epsilon x}^2$ and $\sigma_{\epsilon y}^2$ as 1, 0.01 and 0.01 respectively.

$$
\begin{aligned}
Q_0 &= \sigma_\eta^2 \begin{pmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & 0 & 0 \\ \frac{\Delta t^3}{2} & 1 & 0 & 0 \\ 0 & 0 & \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} \\ 0 & 0 & \frac{\Delta t^3}{2} & 1 \end{pmatrix} \\
R_0 &= \begin{pmatrix} \sigma_{\epsilon x}^2 & 0 \\ 0 & \sigma_{\epsilon y}^2 \end{pmatrix}
\end{aligned}
\tag{33}
$$

2.4.2 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:

$$\begin{aligned} -2E[\log p(X_{1:j}, Z_{1:j})] = \sum_{i=1}^{j} (n_i \log |Q| + n_i \log |R| + \log |\Sigma_0| \\ + E[\sum_{t=1}^{n_i} (X_{i,t} - AX_{i,t-1})' Q^{-1} (X_{i,t} - AX_{i,t-1})] \\ + E[\sum_{t=1}^{n_i} (Z_{i,t} - CX_{i,t})' R^{-1} (Z_{i,t} - CX_{i,t})] \\ + E[(X_0 - \mu_0)' \Sigma_0^{-1} (X_0 - \mu_0)] + constant) \end{aligned} \tag{34}$$

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

$$\begin{aligned} Q &= \frac{\sum_{i=1}^{j} [S_{i,11} - AS'_{i,10} - S_{i,10}A' + AS_{i,00}A']}{\sum_{i=1}^{j} n_i} \\ R &= \frac{\sum_{i=1}^{j} [M_{i,ZZ} + M_{i,XZ} - M_{i,XX}]}{\sum_{i=1}^{j} n_i} \end{aligned} \tag{35}$$

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 1.

2.5 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:

$$\begin{aligned} X_t &= AX_{t-1} + \eta_t \\ Z_t &= CX_t + \epsilon_t \end{aligned} \tag{36}$$

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.

2.6 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 4.2, 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.

2.6.1 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. 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.

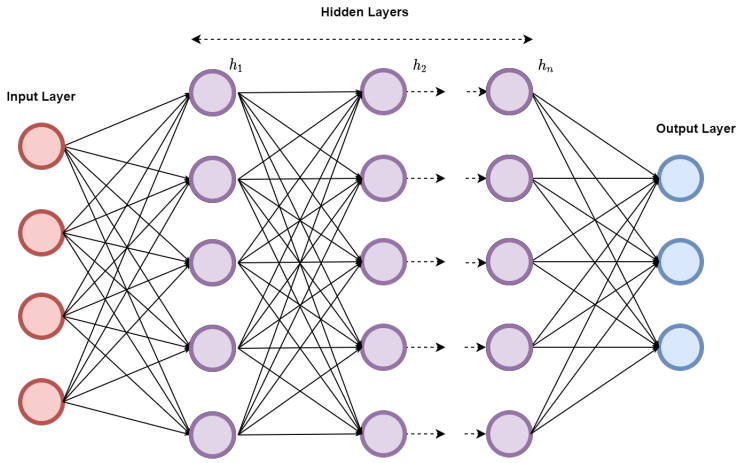


Figure 2: An illustration of neural connections in an ANN

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:

$$\begin{aligned} a_j^1 &= \sigma(\sum_{i=1}^{n_f}(X_i W_{i,j}^1) + b_j^1) \\ \sigma(x) &= \frac{1}{1+\exp(-x)} \end{aligned} \tag{37}$$

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, which has 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:

$$a_j^l = \sigma(\sum_{i=1}^{n_l}(a_i^{l-1} W_{i,j}^l) + b_j^l) \tag{38}$$

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/ may not 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. In most cases 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

$$W_{n+1} = W_n - \alpha \nabla F(W_n) \tag{39}$$

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.

$$\frac{\partial F}{\partial W^l_{j,k}} = \frac{\partial F}{\partial a^l_j} \frac{\partial a^l_j}{\partial W^l_{j,k}} \tag{40}$$

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 for the entire data) 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[8], but this gives noisy updates to the parameters. In practise, the solution is to combine these two ideas to feed-forward min-batches of data through the network. For every mini-batch, loss and gradients are computed and weights are updated. 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.

2.6.2 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. However, the 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 section. In the Kalman Filter, there is a hidden state that affects the measured output. 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 [5] introduced in 1990 which is one of the most widely used RNN variant uses this concept of a hidden state to good use.

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned} \tag{41}$$

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

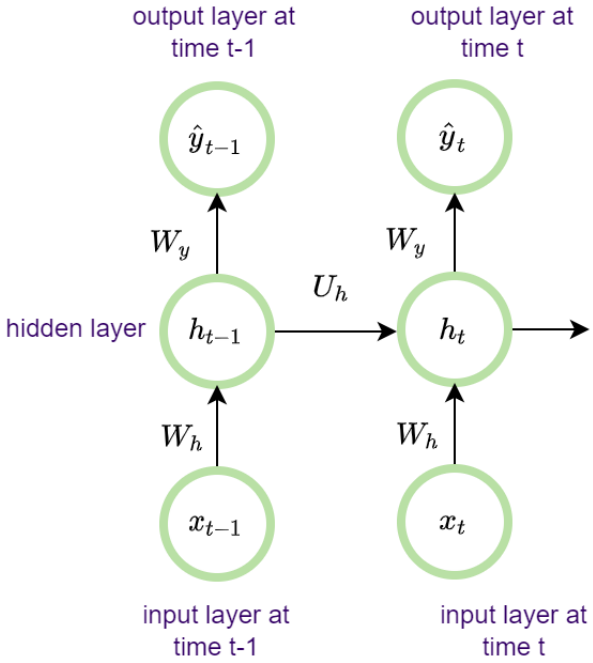


Figure 3: A time unfolded representation of the Elman Network

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 [6]. 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 [6]. In conclusion, Elman Networks were successful in solving relatively simple problems and as sequences scaled up in size and complexity, these networks frequently struggled.

2.6.3 Long Short Term Memory

Long Short Term Memory (LSTM) networks [4], 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.

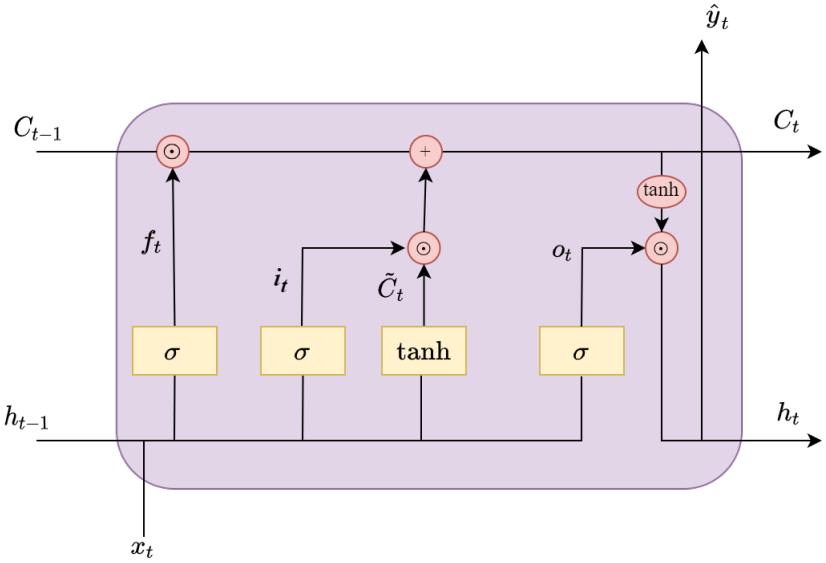


Figure 4: LSTM gated architecture

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.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{42}$$

- 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.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C_t} &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C_t} \end{aligned} \tag{43}$$

- 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.

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t * tanh(C_t) \end{aligned} \tag{44}$$

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

2.6.4 Self Attention and Transformer Encoders

Self attention was a concept first discussed in [15] , which aimed at computing similarity scores between words in a sentence. For example in the sentence "John went to the store because he wanted milk", the word "he" needs to be associated with "John". Self attention is the mechanism through which the model understands these kind of inter-sentence dependencies. 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. We propose to use the self attention mechanism in our problem through a transformer encoder architecture. The transformer encoder architecture was introduced in the same paper, 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. Through this section, this architecture and its constituent building block of self attention will be discussed in detail.

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 f feature vectors and d is the dimension of each feature vector also called as the embedding dimension.

$$\begin{aligned} Q &= W_Q X \\ K &= W_K X \\ V &= W_V X \end{aligned} \tag{45}$$

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 $f \times w$.

Next, the query matrix is multiplied with the transpose of the key matrix. To understand the intuition behind this, let us say that $f = 2$ 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 5: 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.

$$Z = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{46}$$

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}

$$Z_{net} = W_o(concat(Z_1, Z_2...Z_{natt})) \tag{47}$$

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 and layer normalization is applied. 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. 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 method section

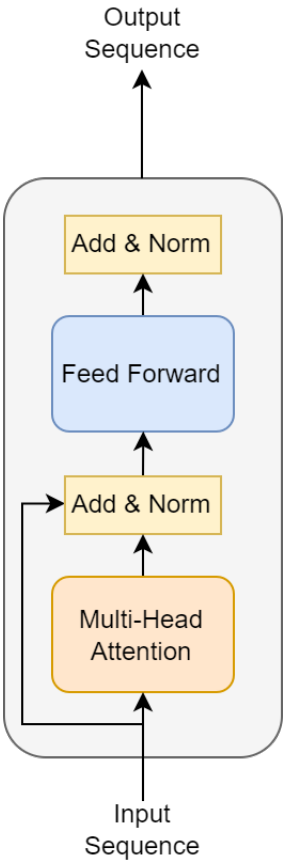


Figure 6: Transformer Architecture

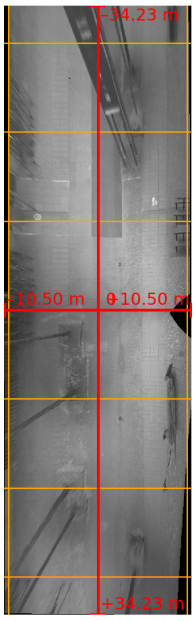
3 Data

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 is presented. Lastly, important pre-processing steps that suitably mould the data for later algorithms are enumerated.

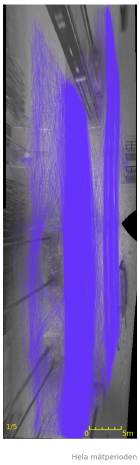
The data has been supplied by Viscando, whose technology tracks all road users to understand movement patterns, traffic flow and human behaviour. Two infrastructure stereo vision sensors were placed on the buildings of the Linköping University campus. The data collected over a week contains

trajectories of pedestrians, cyclists and the autonomous shuttles.

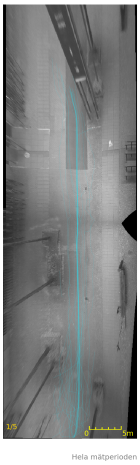
(a) Sensor's field of view



(b) Top view of the frame



(a) Cyclists



(b) Shuttles

(c) Pedestrians

Figure 8: Line plots for trajectories of road users

As can be seen from 7b, the field of vision of the sensor covers an area of 70×21 metre squared.

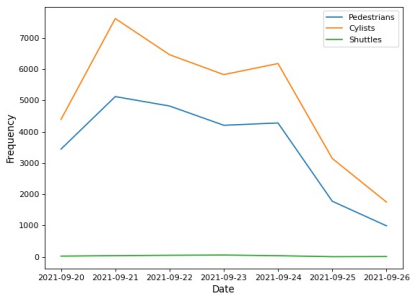


Figure 9: Top view of sensor placement and sensors' field of view

The feature names 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 1: Characteristics of each road user 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 2: Characteristics of each road user in the dataset

3.1 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. The table below shows the distribution of sampling time in the data.

Sampling Times (in seconds)	Frequency
Between 0.05 and 0.1	1407
Between 0.1 and 0.2	2133
0.24	829904
Between 0.3 and 1 seconds	6868
Greater than 1 second and less than 1.5 seconds	1656
Greater than 1.5 seconds	4472

(a) Distribution of Sampling times

Times	User 1	User 2	User 3
09:10:40.00	[x1, y1]	[x2,y2]	[x3,y3]
09:10:40.20	[x1', y1']	[x2', y2']	[x3', y3']
09:10:40.40	[x1'', y1'']	[x2'', y2'']	[x3'', y3'']
09:10:40.60	[x1''', y1''']	[x2''', y2''']	[x3''', y3''']

(b) Common time axis based 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. 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 10b, 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.

3.2 Data processing for recurrent neural networks

Every road user maps out a trajectory which keeps evolving as the road user moves through the frame. 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 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, shu) 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 called as the *Individual Trajectories* data structure will consist of two columns $[input, target]$ and the equations below help in understanding how each row of $[input, target]$ will look like 49 . The Kalman Filter EM Algorithm and the Vanilla LSTM network introduced later in 4.1, 4.2 will be trained on this data structure after applying some exclusions that is mentioned later in 3.2.1.

$$\begin{aligned} input_t^1 &= [[x_1^1, y_1^1, ped, shu], [x_2^1, y_2^1, ped, shu], ...[x_t^1, y_t^1, ped, shu]] \\ target_t^1 &= [x_{t+s}^1, y_{t+s}^1] \end{aligned} \tag{48}$$

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 49. 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 will be trained on this kind of a data structure

$$\begin{aligned} input_t &= [input_t^1, input_t^2 ... input_t^m] \\ target_t &= [target_t^1, target_t^2, ... target_t^m] \end{aligned} \tag{49}$$

where $input_t^k$ and $target_t^k$ is obtained from the equation 48

3.2.1 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 frame is also noisy, because uncertainty in measuring speed is higher than uncertainty in measuring position. Therefore, for safe measure these road users 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 making the *Frame* 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 4.3, 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 *Frame* 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. 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 is removed from the data $[input_t, target_t]$. 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.

Another interesting case that is observed in the *Frame* 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 throughout their trajectory, only one road user is considered as we assume that two road users were detected as one.

The *Individual Trajectories* data structure can be used for the Kalman Filter 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 *Frame* data structure is 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 on the same data.

4 Methodology

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

4.1 CV 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 2.1 constructs dependencies between a latent variable and an observed variable.

$$\begin{aligned} X_t &= AX_{t-1} + \eta_t \\ Z_t &= CX_t + \epsilon_t \end{aligned} \tag{50}$$

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 equation50 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 about the means of the observed states, but also 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 (insert reference) and can also be extended to prediction. This has already been done before

(insert reference) and this model is also 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 2.4. 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 21 and 33
A, C is known according to 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 2.1.2
      Run the backward smoothing step to obtain $X_t^{n_i}, P_t^{n_i}, X_{t,t-1}^{n_i}, P_{t,t-1}^{n_i}$ 2.1.3
      Compute S_{00},S_{11}, S_{10} and S_{00} 29
      Compute M_{ZZ}, M_{XZ}, M_{XX} 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 is 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 2.1.2) 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.

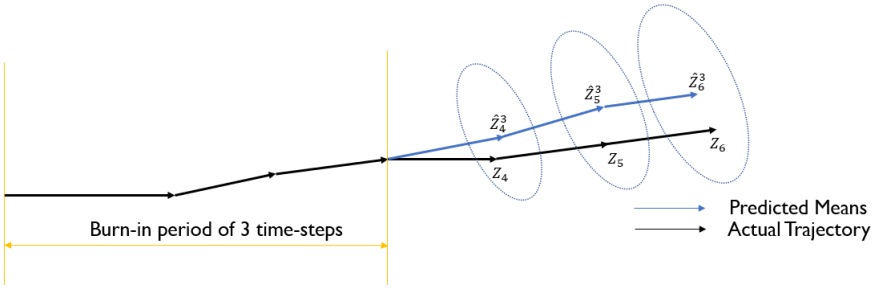


Figure 11: Illustration of predicted means and covariances

4.2 Vanilla Long Short Term Memory networks

There are two 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.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

One way of solving these limitations is to resort to non-linear state space models. However, it has been seen ([12]) 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 probabilistic 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.

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned} \tag{51}$$

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 back-propagation, 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, which 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 (insert references). 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 48.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. $b \times t_{max} \times 4$ is the dimension of the input data. 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.

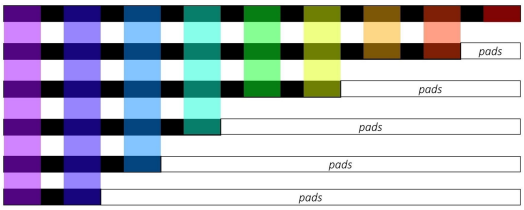


Figure 12: Padded sequences sorted by decreasing lengths

At time step 1 and 2, 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.

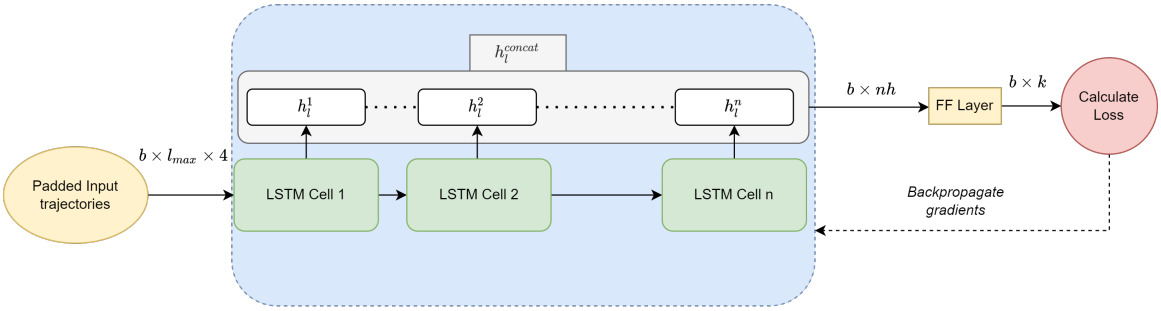


Figure 13: The architecture of the Vanilla LSTM approach

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 $t + s$.

$$MSE = \frac{\sum_{t=1}^{n}(x_t - \hat{x}_t)^2 + (y_t - \hat{y}_t)^2}{n} \tag{52}$$

  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:

$$AFDE = \frac{\sum_{t=1}^{n}\sqrt{(x_t - \hat{x}_t)^2 + (y_t - \hat{y}_t)^2}}{n} \tag{53}$$

  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

$$(\hat{x}_t, \hat{y}_t) \sim N(\mu_t, \sigma_t, \rho_t) \tag{54}$$

  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

$$\begin{aligned} log(p(x_t, y_t|\mu_t, \sigma_t, \rho_t)) &= \log(N(\mu_t, \sigma_t, \rho_t)) \\ &= \log N(\begin{pmatrix}\mu_{t,x}\\ \mu_{t,y}\end{pmatrix}, \begin{pmatrix}\sigma_{t,x}^2 & \rho_t\sigma_{t,x}\sigma_{t,y}\\ \rho_t\sigma_{t,x}\sigma_{t,y} & \sigma_{t,y}^2\end{pmatrix}) \end{aligned} \tag{55}$$

  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 55 and it can analytically be shown for that for a bivariate normal distribution,maximizing the log-likelihood is equivalent to minimizing $f(x, y, \mu_{t,x}, \mu_{t,y}, \sigma_{t,x}, \sigma_{t,y}, \rho_t)$ where f is given in the below equation.

$$\begin{aligned} f(x, y, \mu_{t,x}, \mu_{t,y}, \sigma_{t,x}, \sigma_{t,y}, \rho_t) = \log(\sigma_{t,x}\sigma_{t,y}) + 0.5[\log(1-\rho_t^2)] + 0.5(x_t - \mu_{t,x})^2\sigma_{t,y}^2 \\ + 0.5(y_t - \mu_{t,y})^2\sigma_{t,x}^2 - \rho_t(y_t - \mu_{t,y})(x_t - \mu_{t,x})\sigma_{t,x}\sigma_{t,y} \end{aligned} \tag{56}$$

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.

4.3 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 collission. 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 Social-LSTM is a recent state-of-the-art model that has shown to capture these social interactions 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 (will be added in the relevant literature section). In our method instead of using this pooling, an attention mechanism is used to capture how much attention each road user gives each other. Therefore the output representation coming from the LSTM network for each road user is then passed through an attention layer. 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 . The format of data that is passed into the network is mentioned in 49. 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 $b \times l_{max} \times 4$. 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 paper 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. Again, a feedforward network is used to bring down the dimension of the output coming from a transformer. The final predictions are made for all the road users in the frame and thus 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.

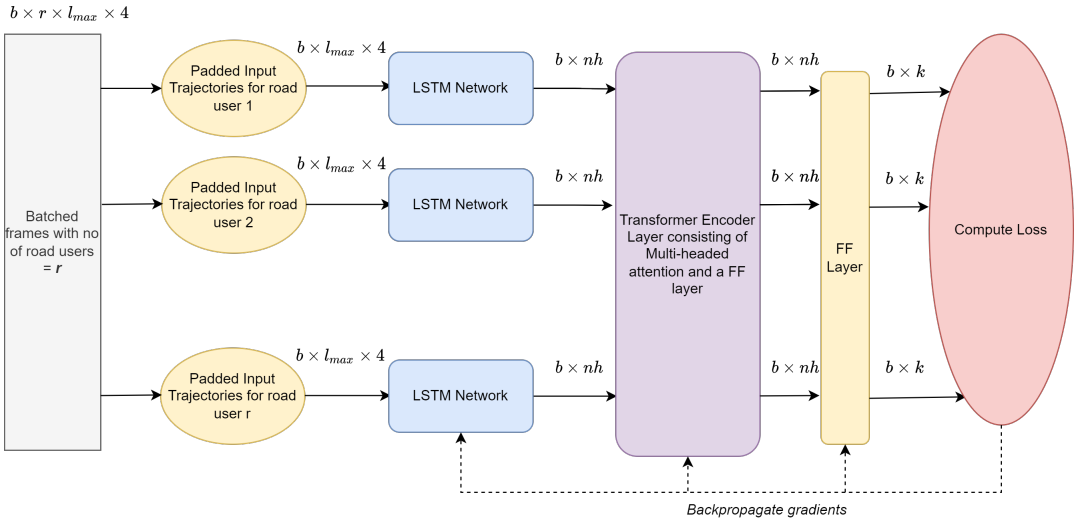


Figure 14: Architecture of the self-attention based LSTM

5 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 dataset 1 and dataset 2 respectively. The Kalman Filter constant velocity model has been trained and tested on both these datasets, whereas the neural network models have been trained and tested on dataset 2. This is because it is computationally more expensive to train the network on the dataset 1, 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 have been chosen for training. 2 days (including another weekend) of data have been

used for testing with the remaining 1 day used for validation. 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. Therefore it was decided to predict 3 seconds into the future and this setting applies to all the results mentioned here.

5.1 Constant Velocity Model

For this model, the train/test/validation split was done on dataset 1 and dataset 2. 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 on the test,validation dataset for the three different road users.

Road User	Dataset 1	Dataset 2
Pedestrian	NA	1.2
Cyclist	NA	3.7
Shuttle	NA	2.4
Total	NA	2.2

Table 3: Mean Squared Error on test

Road User	Dataset 1	Dataset 2
Pedestrian	NA	1.2
Cyclist	NA	3.6
Shuttle	NA	1.2
Total	NA	2.2

Table 4: Mean Squared Error on validation

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

$$\frac{\sum_{i=1}^{n}\sum_{t=1}^{t_i}\log(p(x_t,y_t)|\mu_{x,t},\mu_{y,t},\Sigma_t)}{\sum_{i=1}^{n}\sum_{t=1}^{t_i}[\mathbb{1}]} \tag{57}$$

Road User	Dataset 1	Dataset 2
Pedestrian	NA	-2.3
Cyclist	NA	-3.4
Shuttle	NA	-2.7
Total	NA	-2.7

Table 5: Avg log-likelihood on test

Road User	Dataset 1	Dataset 2
Pedestrian	NA	-2.3
Cyclist	NA	-3.4
Shuttle	NA	-2.4
Total	NA	-2.7

Table 6: Avg log-likelihood on validation

5.2 Hyperparameter-tuning

5.2.1 Vanilla LSTM

For the vanilla-LSTM, the number of layers and the size of hidden layers are two important hyper-parameters. To tune them, a coarse grid hyperparameter tuning was performed where the different networks were trained on different parameters. The networks' test and validation MSE is shown below:

Hidden Layer Size	Number of LSTM layers	Test_MSE	Validation_MSE
10	2	1.18	1.23
	3	1.16	1.17
	4	1.16	1.18
16	2	1.13	1.16
	3	1.13	1.15
	4	1.10	1.12
20	2	1.09	1.11
	3	1.12	1.14
	4	1.09	1.13
25	2	1.08	1.10
	3	1.09	1.09
	4	1.07	1.10
28	2	1.05	1.08
	3	1.05	1.08
	4	1.04	1.08

Figure 15: Hyperparameter tuning for the Vanilla LSTM network

As can seen from the figure above, the test MSE is very similar for the last three entries and the least complex network was chosen (highlighted in green)

5.2.2 Self Attention based LSTM

To train this network, parameters were shared from the trained LSTM network and a similar cross-validation was performed, with an additional parameter of number of attention heads.

Hidden Layer Size	Number of Layers	No of Attention heads	Test_MSE	Validation_MSE
20	2	2	1.04477	1.061322483
		4	1.03043	1.028871693
	3	2	1.04892	1.057303012
		4	1.05324	1.080438175
	4	2	1.05493	1.076258465
		4	1.05925	1.074565851
28	2	2	1.01274	1.038702496
		4	1.01432	1.05047015
	3	2	1.00774	1.029930312
		4	1.02596	1.056226594
	4	2	1.05718	1.076198589
		4	1.03758	1.06924612

Figure 16: Hyperparameter tuning for the Self Attention based LSTM network

5.3 Analyzing and comparing the different methods

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

Road user	CV	Vanilla LSTM	Self Attention LSTM
Pedestrians	1.15	0.62	0.61
Cyclists	3.65	1.66	1.57
Shuttles	2.40	2.34	2.50

Figure 17: Comparison of different models on different road users

Also, the hypothesis is that the self-attention model fares better than the Vanilla LSTM model when the number road users increase. To test this, we have plotted MSE against the number of road users and also taken a ratio of errors between the attention based vs plain architecture.

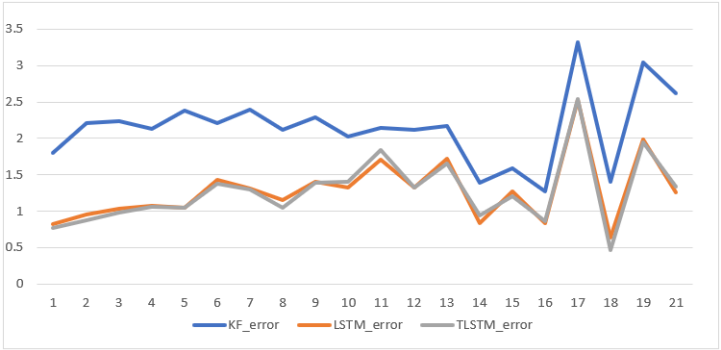


Figure 18: MSE for different number of road users in the scene

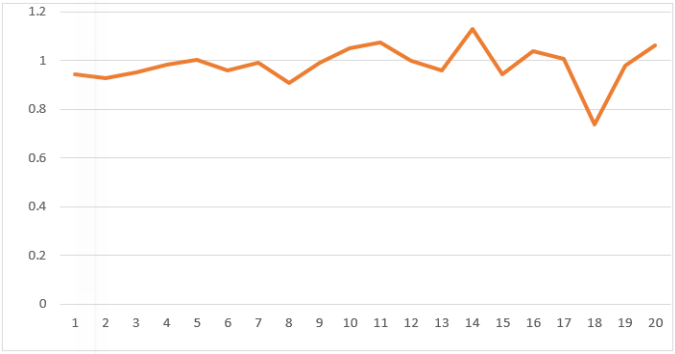


Figure 19: Ratio of MSE between the Attention based vs Vanilla architecture

References

[1] Alexandre Alahi et al. "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] 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).

[3] Tilmann Gneitting and Adrian E. Raftery. "Strictly Proper Scoring Rules, Prediction, and Estimation". In: (2007). URL: https://viterbi-web.usc.edu/~shaddin/cs699fa17/docs/GR07.pdf.

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

[5] 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.

[6] 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).

[7] 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).

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

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

[10] 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.

[11] 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.

[12] Nicolas Schneider and Dariu M. Gavrila. “Pedestrian Path Prediction with Recursive Bayesian Filters: A Comparative Study”. en. In: *Pattern Recognition*. Ed. by David Hutchison et al. 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).

[13] 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.

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

[15] Ashish Vaswani et al. “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).

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

[17] 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..