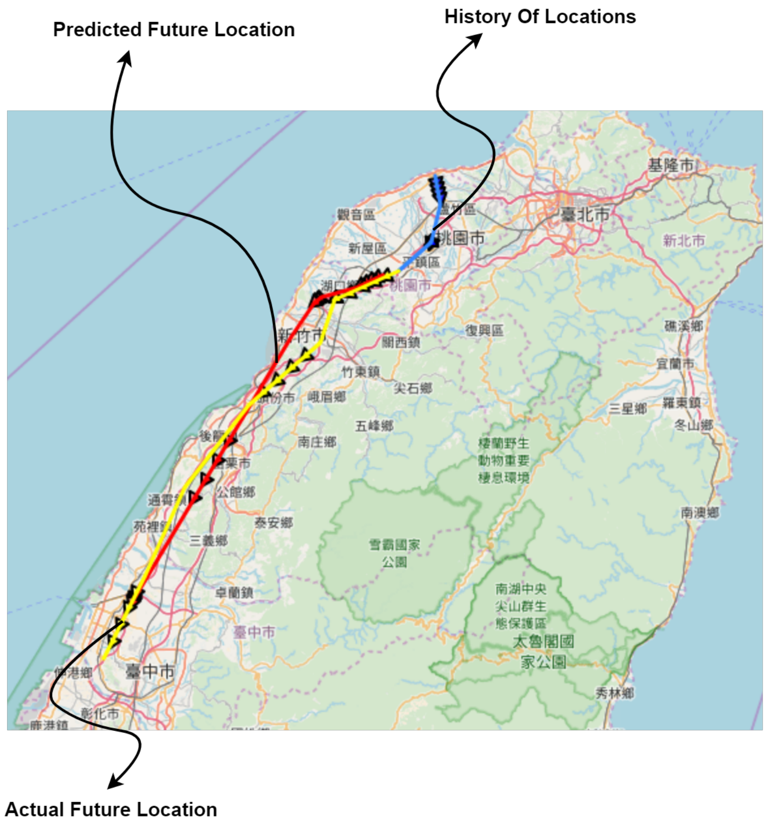
CHALMERS

UNIVERSITY OF TECHNOLOGY



Near-Time Predictions of Future Truck Locations

Using Recurrent Neural Networks

Master's thesis in Complex Adaptive Systems

ABHISHEK SRINIVASAN

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2018

Near-Time Predictions of Future Truck Locations

Using Recurrent Neural Networks

ABHISHEK SRINIVASAN

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2018

Near-Time Predictions of Future Truck Locations
Using Recurrent Neural Network
ABHISHEK SRINIVASAN

Supervisor: Jakob Rogstadius, ECCA, SCANIA CV AB.
Examiner:Mats Granath, Department of Physics,
Chalmers University of Technology

Master's Thesis 2018
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A map showing a prediction from the model.

Typeset in LaTeX
Gothenburg, Sweden 2018

Near-Time Predictions of Future Truck Locations
Using Recurrent Neural Network
Abhishek Srinivasan
Department of Physics
Chalmers University of Technology

Abstract

Location-based services are becoming an increasingly valuable service in the transportation sector. These location-based services rely on current location, but a better service can be provided by predicting the future location. So this thesis aims to investigate near-future location prediction of a specific vehicle using Recurrent Neural Networks (RNNs) and its different variants (a time-series approach). In addition, we discuss the required data pre-processing steps, the architecture of RNNs and the experiments to compare the different variants of RNNs. These experiments show that Long Short-Term Memory Networks (LSTM) have better predictions when compared to simple RNNs and Bi-directional Recurrent Neural Networks (BRNN). The results from this work will be helpful in developing a better short-term location prediction model and will provide better services to SCANIAs customers.

Keywords: location prediction, time-series, RNN, LSTM.

Acknowledgements

I would like to thank Scania for this interesting opportunity and a challenging problem, Jakob Rogstadius for all his guidance throughout this project and, family and friends for all the support thought the thick and thin of my life and the project.

Finally a special thanks to Aarthi, Diego, Hari and other friends who helped in proofreading the report.

Abhishek Srinivasan, Gothenburg, June 2018

Contents

List of Figures

List of Tables

1
Introduction

Location-based services play a major role in enhancing the engineering application involving geographical positions. These services have an application in industries like logistics, health care, etc. These services mainly depend on the current location, but if one could foresee the future location which could improve the benefits of such services. Hence, developing a proactive location-based service will help in many ways. Directing the focus towards logistic industry using trucks, these proactive services can help in advertising, planning co-missions etc. What does co-mission mean? For a vehicle, a co-mission is sharing an extra transportation mission with the regular ones, considering that they share the major part of the routes. These co-missions can be planned in a better way if the future vehicle movement is known.

1.1 Problem Statement

To achieve a proactive location-based service the information about the future location is required. Hence the present study focuses on predicting the location of the trucks in near future. The near future corresponds to a maximum of 20 hours. Finally, the developed model will be evaluated using the mean distance between the predicted and the actual location.

1.2 Delimitation

The scope of this thesis is constrained to following conditions:

- The data of the trucks from Taiwan is used, as the country is an island and the vehicles are constrained within the island.
- Model learns a specific vehicle's movement behavior.
- Prediction of location for a short duration. This thesis predicts for 13 hours in future.
- Some factors were not considered in predicting the location which includes weather, regulation e.g In Europe, a mandatory break of 45 minutes after 4.5 hours of travel is required.

1.3 Approach

This thesis solves the location prediction problem using time-series analysis. The data is transformed into a multivariate time-series and this time-series is predicted using recurrent neural networks (RNN) and its variants (bi-directional recurrent neural network (BRNN) and long-short-term memory(LSTM) [2]).

1.4 Summary

The rest of this report is organized as follows:

- Chapter 2 is a compilation of the related works which would give an intuition of prior works which were used for predicting the location and similar problems.
- Chapter 3 introduces the theory of neural networks, Recurrent Neural Network and other background required for this thesis.
- Chapter 4 explains the methods for the data representation, data pre-processing and the general architecture of the model.
- Chapter 5 describes the experiments and results required for a deeper understanding of the model is compiled.
- Chapter 6 presents a general discussion about the model, ethics and future works.
- Chapter 7 concludes the thesis with the learning from the thesis.

2

Related Work

This chapter discusses the classical, machine learning and neural network models for location prediction

2.1 Classical Location Prediction

Markovian:

Most of the classical model works with a fixed number of location space [[3], [4]]. It is the same case in the work of Burbey, where an extensive study of these methods was made and used Markov model for predicting the future locations. From the research, it is shown that the first-order Markov model with fallback works well. Similarly, Sadilek [4] has proposed two methods for learning the trend of the vehicle in a long-term using SVD (singular value decomposition) and PCA (principal component Analysis): projected eigendays and segregated eigendays model. Sadilek work shows that these model could learn the periodic behavior and believes that the performance would improve with the prior Bayesian probability.

In some others, the number of possible location is made finite by clustering the coordinates. For example, in the work of Mathew [5] the location sequence is clustered based on the time of the day and the day of the week, depending on the cluster type different Hidden Markov Model (HMM) [6] is used. The sequence of a categorical feature (triangular mesh of the world) is used in HMM. The performance of HMM with different hidden states, cluster, and geo-resolution are compared. It turns out to be the best performing algorithm with an accuracy of 13.36%. Similarly, Ashbrook [7] clusters the locations using K-means and uses Markov model to predict the future movement of a single user. Later, Ashbrook [8] performed the same experiment with multiple users. These experiments show that the model could able to learn the behavior of Global Positioning Coordinates (GPS). Similarly, Huang [9] uses transition matrix to predict the future location of the user. spatial clustering and some temporal cluster are done on the location and it is used for learning the transition matrix.

Euclidian Based:

Euclidean-based algorithm like Adaptive Resonant Theory (ART) [10] where the paths are clustered based on the Euclidean distance between the paths. Later given a path the next location is predicted by finding the cluster similar to the current sequence and uses it to predict the next location. When the prediction error is more than the threshold the current sequence is added to the cluster. These methods work even in the zero-knowledge environment.

Random Forest:

Gohil [11] uses a mobile application to collect the location data and later uses the random forest for predicting the location. It uses the results from multiple trees and uses the combined probabilities for prediction.

2.2 Neural Network based Models

Fully Connected Neural Network

A spatiotemporal semantic neural network is used for predicting the moving object [12], where the data used is from food service and other from a bus service. The data is split into trips. Then the data is temporally processed by binning, later spatially processed by interpolating the data point to the closest road and finally semantic meaning is provided by replacing with 'RoadId'. The processed sequence of the location is used in predicting the future location of the vehicle. Performance of the model is more when the sampling rate is higher and with a prediction precision of 27%.

Recurrent Neural Networks

Al-Molegi [13] used STF-RNN which comprises both spatial and temporal information in them. STF-RNN takes in one hot encoding of the position and one hot encoding of the temporal information (the hour of the day). The one hot encoding of location is obtained from the cluster centroids of the available position. This method has improved the effectiveness of the states of the art and strengthens the stand that the recurrent neural network learns the user behavior.

In telecommunication, the movement of a user is predicted so that the communication packets are rerouted efficiently. Kaaniche [14] used a fully connected network where its output recurs back into the input. The fully connected network uses k previous states (states are all possible network nodes) to predict the current state. The predicted current state recurs into the model to predict the next state. Change in weights is calculated using Back Propagation Through Time (BPTT).

In [15] X. Song et. al used an anonymized data of 1.6 million people which are collected for a period of approximately three years. In this research multi-layered LSTM is used with encoder and decoder architecture. The initial layer acts as an encoder (separate for position and type of transport), later comes to the shared hidden LSTM layers and finally the separate decoders are used for location and the transportation modes. This model seems to outperform other models like Gaussian model and time-delay neural network.

Some time series prediction models have high similarities with the location prediction as the sequence of position can be considered as the multivariate time series.Bandara [16] uses LSTMs to predict the multivariate time series. Here his team uses signal decomposition and log transformation for data-preprocessing and also uses different models based on the cluster the signal belongs. The characteristics of the signal are used to classify the signal to a different cluster. This shows that the LSTMs could learn a time-series with suitable pre-processing.

Conventional Neural Network

One of the different approach by Jianming Lv [17], is converting the location sequence into an image on the map.This image of a trajectory is used to predict the final destination of the trip using T-conv which is modified CNN (convolution neural network), to capture multi-scale features. This shows that the multi-scale features can be captured by T-conv.

2.3 Motivation for the Current Work

As seen from the above methods each had its own limitations. The methods [4, 6, 7, 8, 13] have a limitation predicting over a limited location space which is not suitable for this business case as the vehicle might be located in more than a definite number of locations. Another method [12] uses the trips which do not fit the business case as the stationary behavior of the trucks are also important to learn. The [17] converts the data into a spatial information and so the temporal information lost, the current application requires temporal information too. All of the above limitations can be covered by using the time-series analysis using recurrent neural networks (and its variants) are used to predict the future location.

3
Theory

This chapter introduces the topics of learning models, Neural Network, Recurrent network, different architecture of recurrent networks, different types of recurrent networks and hyper-parameter tuning methods.

Learning Model: Learning models are capable of learning from the data. These learning models can be classified based on the learning type or on type of problem that it solves: On the learning type it is classified as either supervised or unsupervised and on the type of problem as classification or regression problem. The supervised model tries to learn a statistical relation between the input and the output, which is similar to a teacher supervising a student and giving feedback which answers are right and which are not right. While, the unsupervised model does not require labels for learning. Some of the supervised learning models used for predicting future location are Recurrent neural network, it's variants and fully connected networks.

3.1 Artificial Neural Network

Neural network is the general term for artificial neural network. A neural network is a simplistic representation of the biological neuron which also capable of learning. There are different types of neurons in the human brain and as so in the artificial neural network. Example Include Hopfields, Kohonnen, feed forward neural network and recurrent neural network. This project focuses more on feed-forward and recurrent neural networks.

3.1.1 Fully Connected Neural Network

Fully Connected neural network goes by many different names like feed-froward networks, perceptron etc. This mimic the most common form of neurons in the brain[18]. Any type of neural network can be represented in the form of directed acyclic graphs (DAG). The nodes in the graph are the neurons and the edges are the weights. In a fully connected neural network the neurons are arranged in different layers as in figure 3.1. The neurons can have forward connections only to the neurons in the next layer (towards the output layer).

Each neuron has two parts; a summing unit that sums the inputs and an activation unit (described in the section 3.2). In any type of neural network, learning is to find an optimal set of weight where the loss of the network is minimized. This learning process consists of two main parts: feed-forward and back-propagation. In feed forward the inputs are fed to the input layer and the output is calculated. Equation 3.2 is the feed forward procedure for the network shown in figure 3.1

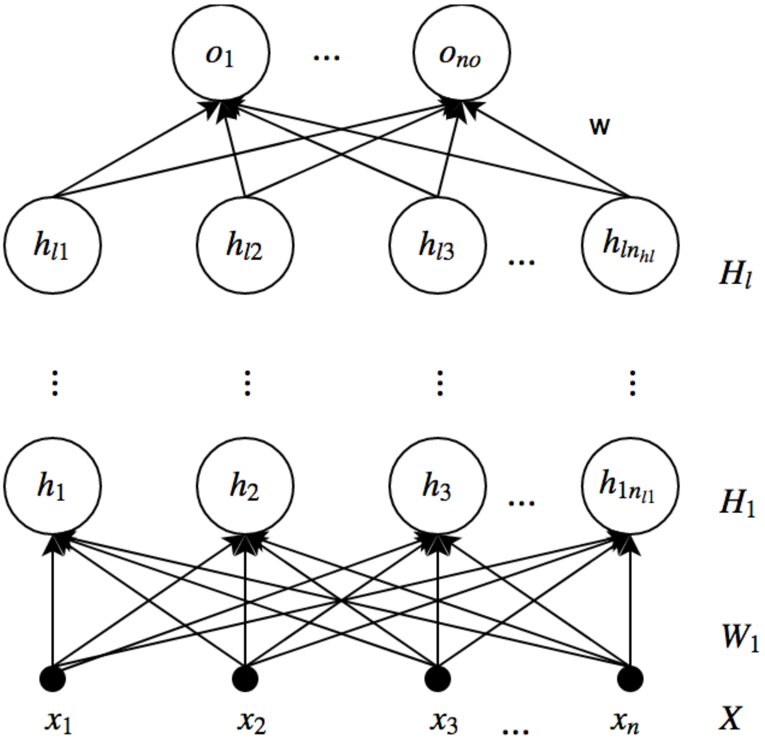


Figure 3.1: Fully Connected Neural Network

All the values can be represented as a vector as shown in equation 3.1

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \qquad H_i = \begin{pmatrix} h_{i1} \\ h_{i2} \\ h_{i3} \\ \vdots \\ h_{in_{li}} \end{pmatrix} \qquad O = \begin{pmatrix} o_1 \\ o_2 \\ o_3 \\ \vdots \\ o_{no} \end{pmatrix} \tag{3.1}$$

$$\begin{gathered} H_0 = XH_i = g(W_i.H_{i-1} + B_i) \\ O = g(w.H_l + b) \end{gathered} \tag{3.2}$$

In equation 3.2, $g(.)$ represents the activation function. W and w are the weight matrices, H is the hidden unit vector, X is the input,vector B and b are the bias, i is the hidden layer number, l is the total number of hidden layer and O is the output. There are two notations for counting the number of layers in the fully connected network, one includes the input layer and other omits it[19]. This thesis uses the latter one.

Once the output is calculated the forward propagation is done and the loss is calculated for the current outputs and this is used to calculate the gradients which is used for optimization (further explained in section 3.3).

3.1.2 Recurrent Neural Network

Recurrent neural networks are capable of understanding the order of the sequences e.g., a sentence in any language does not make any sense if the order of the words is not correct. So in some cases it is important for the network to understand the order of in which the data is presented. The recurrent neural networks captures characteristics of a sequence using the recurrent connections.

3.1.2.1 Simple Recurrent Networks

Recurrent neural network is quite different from the fully connected neural network by having recurring connection and connections to the other neurons in the same layer. For example, the figure 3.2 shows the neural network with a recurring layer which is represented across time (unrolling the network through time). At each time step the network takes in input and the hidden states from the previous time step. The part of the network which preserves the information through time is called the memory cell. A simple cell is described in the equation 3.3.

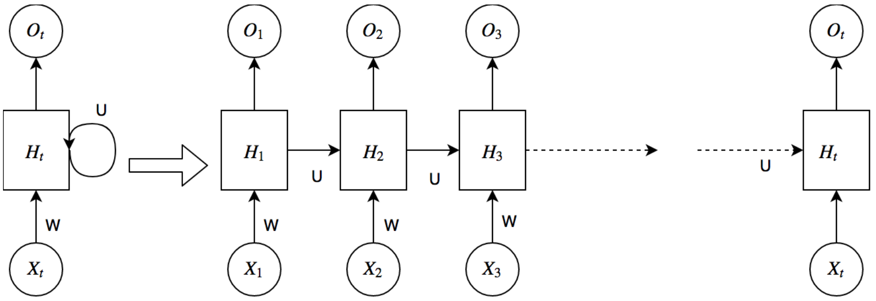


Figure 3.2: Recurrent Neural Network (Right:Rolled form,Left:Unrolled form)

In the figure 3.2, X represents the inputs, H is the hidden inputs which carry information through time and W, U are the weights for the input and the hidden inputs. The forward propagation is given by the equation:

$$H(t) = f(W \cdot X(t) + U \cdot H(t-1)) \tag{3.3}$$

where $f(\cdot)$ is the activation function. In some cases the hidden state is directly the output and in others these states are used as the input to the single-layer neural network layer which is used as the output.

The parameters are updated using a method called back propagation through time [20].

3.1.2.2 Long Short-Term Memory (LSTM)

The LSTM [2] is a variant of RNN. It is similar to simple RNN with a small difference of having a cell state which can carry long-term dependencies. During the forward propagation there are three different information gates. The forget gate determines which information from the previous time step is to be carried forward. The output gate determines the amount of current cell state to be the output. The input gate is responsible of updating the cell state. The forward propagation of the LSTM network is given in equation 3.4.

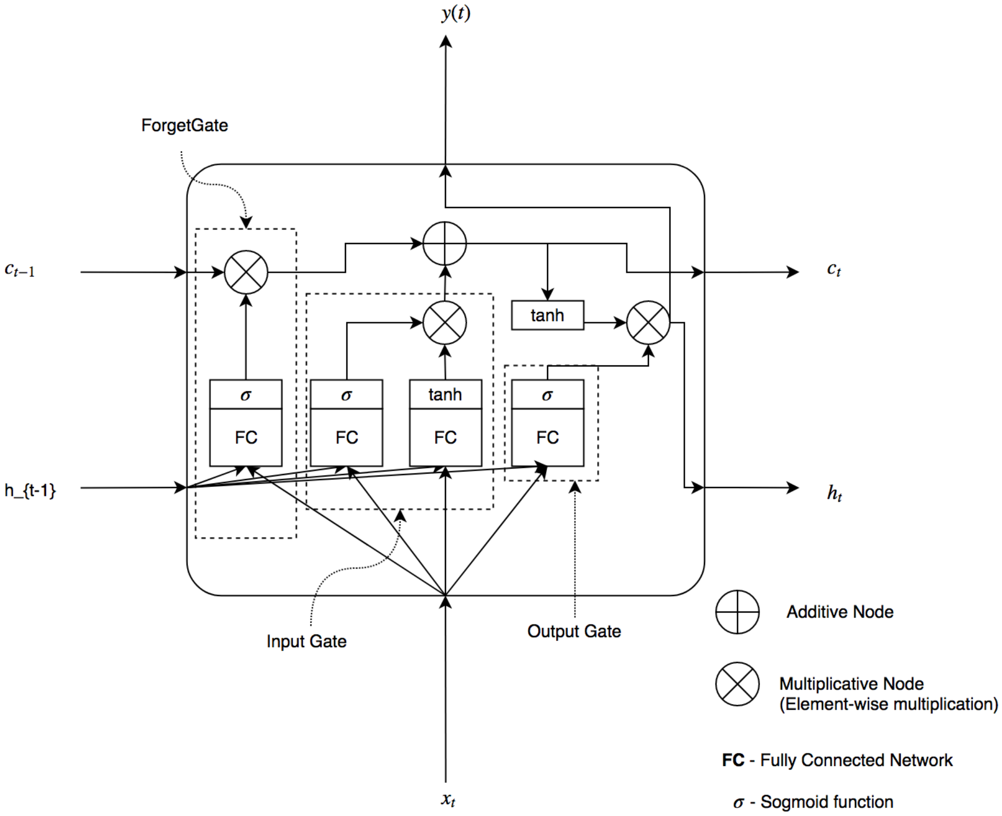


Figure 3.3: Long and Short-Term Memory Neural Network

$$\begin{aligned}
i_t &= \sigma(W_{xi}^T.x_t + W_{hi}^T.h_{t-1} + b_i)\\
f_t &= \sigma(W_{xf}^T.x_t + W_{hf}^T.h_{t-1} + b_f)\\
o_t &= \sigma(W_{xo}^T.x_t + W_{ho}^T.h_{t-1} + b_o)\\
g_t &= tanh(W_{xg}^T.x_t + W_{hg}^T.h_{t-1} + b_g)\\
c_t &= f_t * c_t - 1 + i_t * g_t\\
y_t &= h_t = o_t * tanh(c_t)
\end{aligned} \tag{3.4}$$

c_t is the cell state at time t. h_t is the hidden state at time t. x_t input at time t. W_{xo}, W_{xi}, W_{xf} and W_{xg} are the weights connected to the input, $W_{ho}, W_{hi}, W_{hf}, W_{hg}$ are the weights connected to the hidden states, b_o, b_i, b_g and b_f are biases terms and i_t, f_t, o_t are the outputs from input, forget and output gates.

3.1.2.3 Vanishing Gradient Problem

In the case of deep neural network (a neural network is called deep neural network if the hidden layers are present), during the parameter update step the gradient has to flow back till the input layer (in the process of back-propagation). As these network has a huge number of layers, the gradient vanishes [2] (or diminishes) when the gradient reaches the input layer. In some other cases the gradient grows exponentially. This problem is referred to as the vanishing gradient and the exploding gradient problem.

The problem of vanishing gradients can also be seen in the simple recurrent neural network. Consider a unrolled RNN (as in figure 3.2), where the possibility of vanishing gradients increases as it does the number of time steps. This problem can be overcome by using a variant of RNN, which is LSTM. LSTM has multiple states flowing through time, one is the hidden state for remembering the short-term dependencies and the other is the cell state for remembering the long-term dependencies. The cell state is a connection flowing from left to right without much change on it and so the gradient diminishes/ explodes much slower.

3.1.2.4 Different Architecture

The different variant of the recurrent neural network can be used in the different architecture depending on the requirement of the problem. These different architectures can be seen in figure 3.4.

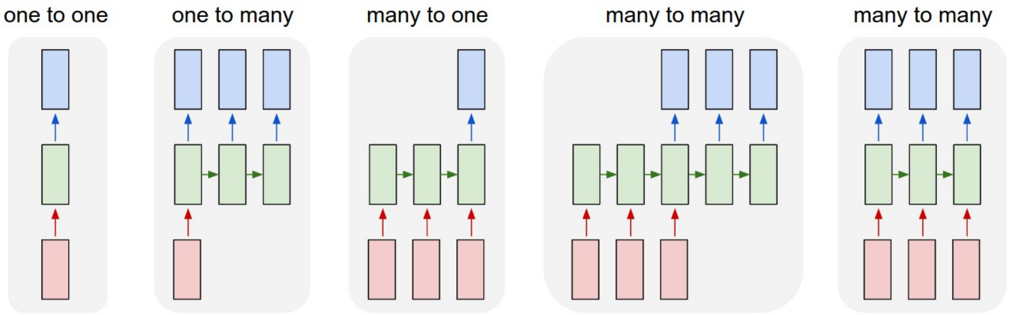


Figure 3.4: Different RNN Architectures [1]

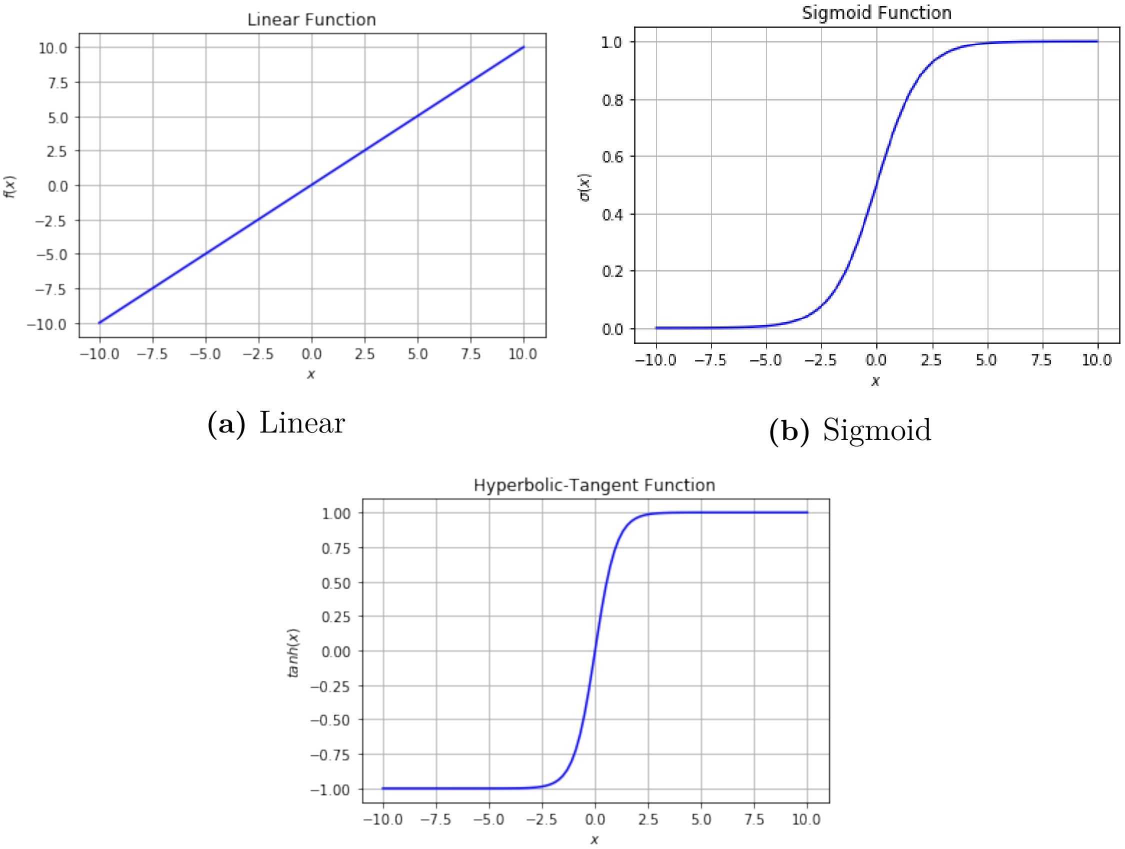
As in the figure there are four different architectures. One to one is the same as the feed-forward neural network since the network does not carry any information in time. Many to one architecture is an architecture where the network looks at the inputs and gives a single output, e.g. like sentiment classification on sentences. There are two types of many to many architecture: encode and decode type, and predicting one output for every time step. For example, the encode and decode type is used for the the language translations and the other one is used to classify if a word is a name or not.

3.2 Activation Function

Activation function is a part of the neurons. Most of cases they act as a squashing function so that the output of the neural network does not explode. In other words, the activation function decides which neuron should be active for the given input. There are different types of activation functions. This section discusses about linear, sigmoid and tanh functions.

Consider a linear activation function (shown in figure 3.5a) as in equation 3.5, which is equivalent of not having an activation function (the input is the output). Here, any combination of neurons cannot represent a non-linear function (a combination of a linear function presented in a linear manner is still a linear function). So this activation function can only capture the linear properties.

$$f(x) = x \tag{3.5}$$



(a) Linear

(b) Sigmoid

(c) Hyperbolic-Tangent

Figure 3.5: Activation Functions

So to represent non-linear functions, a non-linear activation function should be used. Sigmoid and tanh are some of the examples of non-linear activation functions. Sigmoid function is shown in figure 3.5b and mathematically represented in equation

3.6. This function squashes the input to the range of 0 to 1. It can be used for classification problem where the value from the sigmoid function is used as a confidence score.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.6}$$

Tanh (shown in figure 3.5c) is also activation function of which squashing function which squashes the input, but from 1 to -1. Its mathematical form is given by equation 3.7.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.7}$$

Some other notable activation functions are RELU, modified RELU, logit, softmax, piecewise linear and complementary log-log [21].

3.3 Optimization

Optimization is the process of finding the value at which a function reaches its minimum or maximum value. One of the applications of optimization algorithms is in neural networks, where it optimizes the weights so that the network has minimum error. Some of the different optimization methods that could be used are gradient descent, stochastic gradient descent and mini batch gradient descent. In a supervised learning model, the model takes an input and forward propagates it through the model. It predicts the output and the losses are computed from these predictions. These losses are used to calculate the gradients. These gradients are in turn used for the weight update such that the loss is minimized.

The loss function is quite different from the metric that is used to evaluate the model. The loss function can be conceived as a close representation of the metric such that the function is convex and so the process of optimization is efficient. Some of the notable optimization algorithms are simple gradient descent, Adagrad, AdaDelta and ADAM [22, 23].

Gradient decent optimization works by moving the parameter towards the steepest gradient by some factor of learning rate. In equation 3.8 x is the parameter of the function, $f(x)$ is the function to optimize, $\nabla f(x)$ is the derivative of the function and η is the learning rate. The learning rate η should be carefully fixed: if it is too small the optimization might settle in a local optimum and if it is too high the optimization might diverge instead of converging to the optimum parameter.

$$x = x - \eta \cdot \nabla f(x) \tag{3.8}$$

Adam[22] is a recent stochastic optimization algorithm which uses two moments for updating the parameter. This algorithm optimizes based on the first gradient of the objective function. The first moment is updated based on the gradient and the second one is updated based on the squared gradient. Here The parameter update takes place as expressed in equation 3.9.

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \\ \hat{m_t} &= m_t/(1 - \beta_1^t) \\ \hat{v_t} &= v_t/(1 - \beta_1^t) \\ \theta_t &= \theta_{t-1} - \alpha \cdot \hat{m_t}/(\sqrt{\hat{v_t}} + \epsilon) \end{aligned} \tag{3.9}$$

In equation 3.9 β_1 and β_2 are the exponential decay of moments, m_t and v_t are the first and second moments at time t respectively, $\hat{m_t}$ and $\hat{v_t}$ are the bias correction terms at time t. For a machine learning problem $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$ are recommended [22].

3.4 Dropouts and Regularization

3.4.1 Dropouts

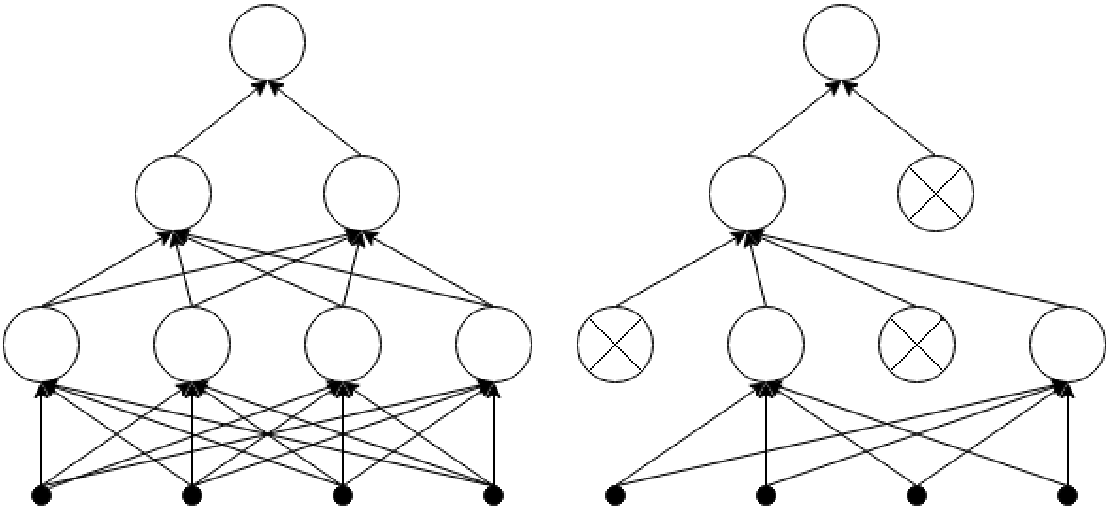
When a neural network is trained the model will at some point over-fit the training data and so the error in the test data starts increasing. This problem can be solved using dropouts and regularization to generalize the model.

Drop-outs [24] is a process of simply ignoring some neurons for the process of training. These neurons are selected randomly with a probability of $1 - p$, where p is the **keep probability**. In each training iteration the nodes are selected with a probability of $1 - p$ and dropped. During the test phase all the neurons are used and reduced by a factor of p. The reason for the drop-out is to avoid over-fitting as each neuron learn to act independently of others.

3.4.2 Regularization

During the training the value of the weights increases and this in turn over-fits the training data. To solve this problem regularization can be used. There are two types of regularization $L1$ and $L2$. The L2 regularization [25] term is given by the second term added to the loss int he equation 3.10, where w is the weight and λ is the regularization parameter. When λ is small the contribution of the regularization term is small and when it is large, the regularization is large and that makes the weights are small.

$$L2_{loss} = loss + \frac{\lambda \cdot w^2}{2} \tag{3.10}$$



(a) Fully connected Neural network **(b)** Neural network with dropouts

Figure 3.6: Difference between models

3.5 Hyper-parameter Tuning

There are many parameters which could affect the performance of the model like learning rate, regularization term, number of neurons, number of layers and when the model should stop the training (i.e. number of iterations). Some of the common methods for hyper-parameter tuning are grid search and random search. In grid search methods a grid parameter is selected and the model is tested over it. In random search multiple tests of random combinations of parameters in the suitable range are tested. An advantage of this type of test is that we are able to find the parameter which affects the performance most[26].

3.6 Cross-Validation

During the learning process the data is split into train and test. The model learns on the training data and validated on the test data. This process is not an overhead in computation. This can help in making a choice in selecting the model. This can also be used to detect when the model over-fits the training.

4
Methods

This chapter covers the evaluation, the methods used for proof of concept and predicting the future location.

4.1 Time-Series prediction

Time-series is a set of data points which are ordered in time. There are two different types of time series uni-variate and multivariate. A univariate time series is dependent solely on time whereas a multivariate time-series is dependent on multiple interdependent variables at each time-step. In simpler words, the multivariate time-series can be considered as a higher dimensional time-series.

A simple time function like sine is an example of uni-variate time-series, while the location prediction would be an example multivariate time series as the future location latitude and longitude are dependent on previous latitude and longitude (interdependent variables).

4.2 Evaluation Metrics

The evaluation metrics play a crucial role in measuring and comparing the models that are developed. The metric used for the location prediction problem is to compare the true locations against the predicted future locations. The comparison can be made by calculating the distance between the points. The distance between two locations (lat1, lon1) and (lat2, lon2) can be measured using the haversine distance. The method for calculating the haversine distance can be found in the appendix A.1.

The table 4.1, represents a sample prediction, its true location and the error of that particular prediction, the units of latitude and longitude are in degrees and the unit of error is kilometers (Kms).

True lat	**True lon**	**Pred lat**	**Pred lon**	**Error(kms)**
24.502	120.793	24.509	120.809	1.50
⋮	⋮	⋮	⋮	⋮

Table 4.1: Evaluation: Mock Error Calculation

To evaluate a model, the average error of the prediction of is calculated. Later this error is used for comparing the model. Less the error better the model is at predicting the future location.

4.3 Baseline Model

A baseline model is a simplest possible model used as a benchmark for the prediction problem. For example, it can be a random guess, a heuristic model or a statistical model. For location prediction problem, the developed baseline model works by predicting all points in the shorter future as the mean of the inputs. This can be seen in the figure 4.1 below that the baseline model takes the k time steps as an input and predict the next k time step as the average of the input steps.In other words, the output of the model will be the average of the latitudes and longitudes provided. Later the performance of this model is evaluated using the metrics defined in the section 4.2 and this metric is used to compare other machine learning models.

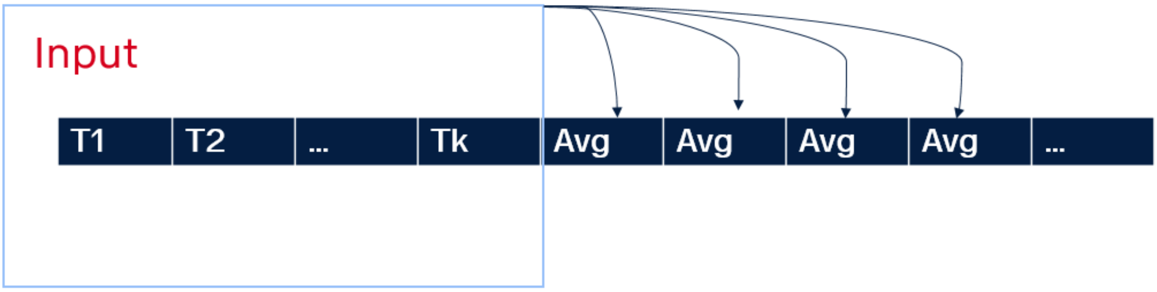


Figure 4.1: Illustration of Baseline Model

4.4 The General Architecture

The figure 4.2 shows the unrolled stacked recurrent neural network (general architecture). 'A' can be any variant of the recurrent neural network. m is the number of layers, n is the length of the input(i.e the number of input steps). Here the output of each cell 'A' is a vector and its size depends on the number of neurons (denoted by N) of the cell. Finally the dense layer (or fully connected layer) transforms the output to the required dimension.

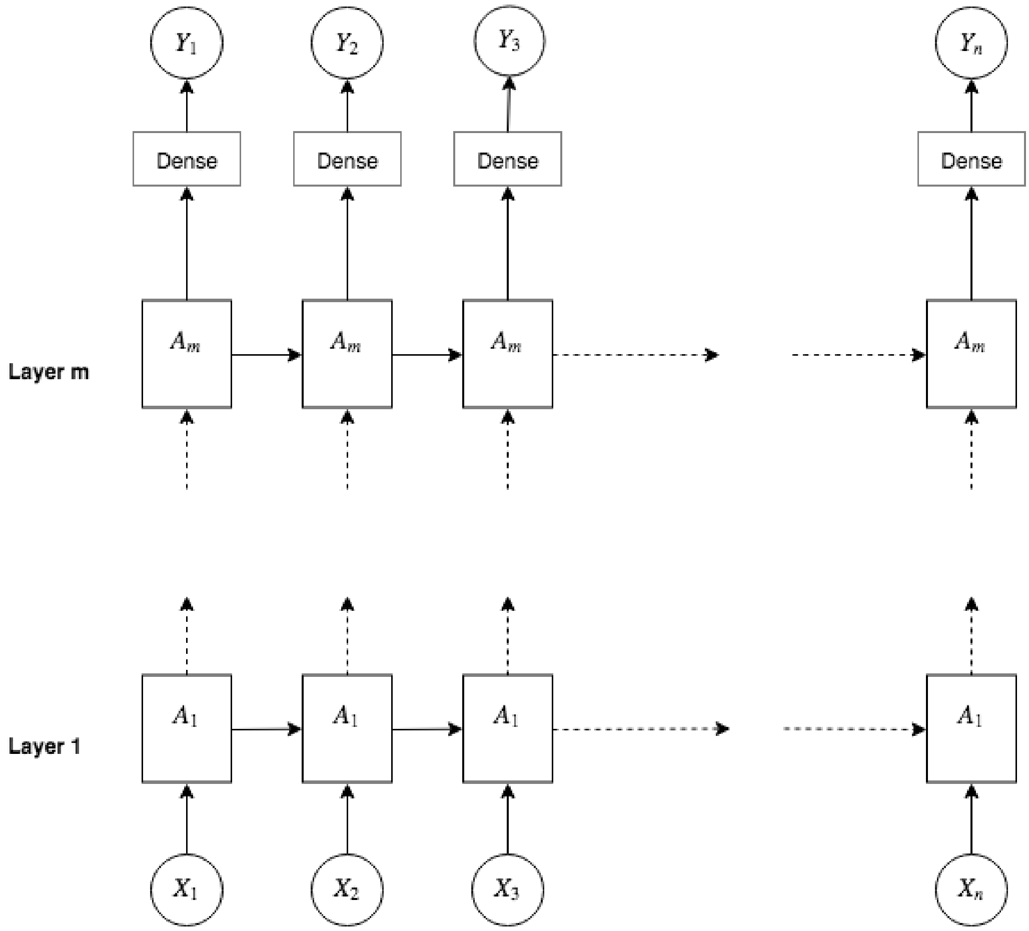


Figure 4.2: General Architecture: Recurrent Neural Network Architecture

The X_i is the input where the i is the time-step, A_k can be any variant of the recurrent network where k is the layer number.

Training: During the training, the inputs are forward propagated to get the outputs Y. It is used to calculate the mean square error (MSE) expressed in the equation 4.1, which acts as a cost. This cost is minimized by updating the trainable parameter with the help of the optimizer (ADAM is used in this project).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.1}$$

In equation 4.1, Y_i is the output predicted and $\hat{Y}_i$ is the actual value.

Testing: During the testing process the outputs are obtained by forward propagation and then evaluated with the help of metrics discussed in the section 4.2.

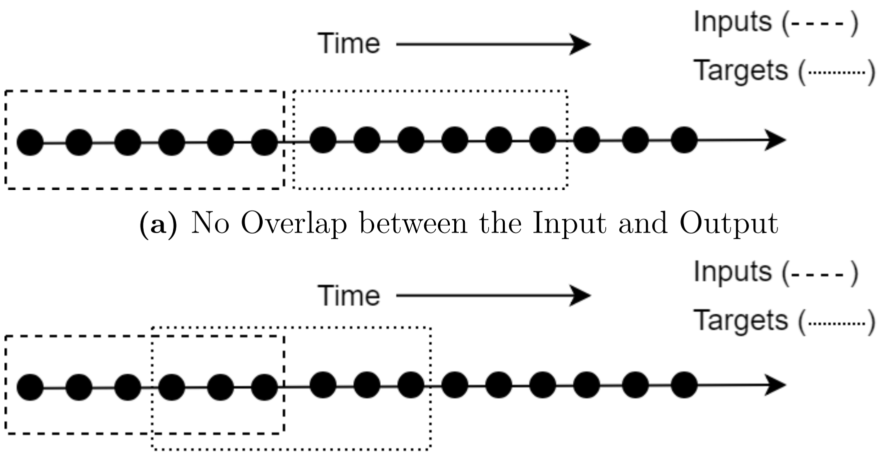
4.5 Data Representation for the Model

The general architecture explained in section 4.4 takes the inputs and output of the same size. To predict a time-series using this architecture the data has to be represented in a particular way. Some terms like **overlap** and **instances** are explained in this section.

4.5.1 Overlap

Overlap is the percent of time-steps which is common between input and output. The figure 4.3a shows the input and outputs with zero overlap and the figure 4.3b shows an example with a 50% overlap.

When the data with no overlap (shown in figure 4.3a) is used for training, the cells learn to predict the k steps ahead. Whereas when the data with 50% overlap (shown in figure 4.3b), the cells learn to predict k/2 steps ahead.



(a) No Overlap between the Input and Output

(b) Overlap of 50% between Inputs and Outputs

Figure 4.3: Different possibilities of Input and Output

4.5.2 Instances

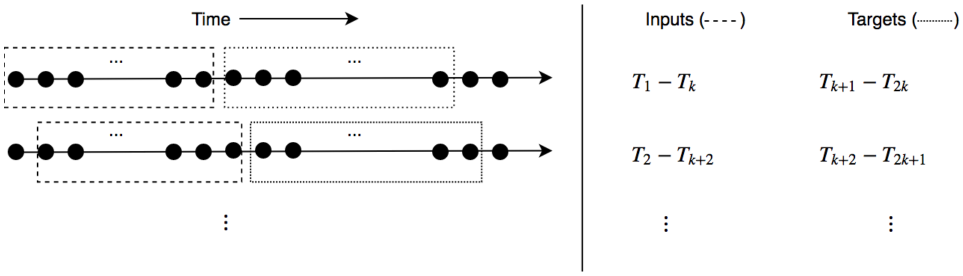


Figure 4.4: Visualization of Instances

A pair of input and target (which the model learns to predict) makes an instance. The instances used for the training is created from the time series. These instances used for training and testing the model. This image shows the example for creating the instances for the case where there is no overlap. The first instance starts from the beginning of the time series (the beginning of the train/test data).
The time series is split into train and test parts. For training and testing, the instance are generated from the respective parts of the time-series.

4.6 Recurrent Networks in learning simpler Time-series

As a proof of concept to show the recurrent neural network could learn a time-series function, the suggested architecture (as in the section 4.4) is implemented. The implemented model is tested on the synthetically generated time-series given in the equation 4.2 and shown in the figure 4.5

$$signal(t) = \frac{t \times sin(t)}{3} + 2 \times sin(5t) \tag{4.2}$$

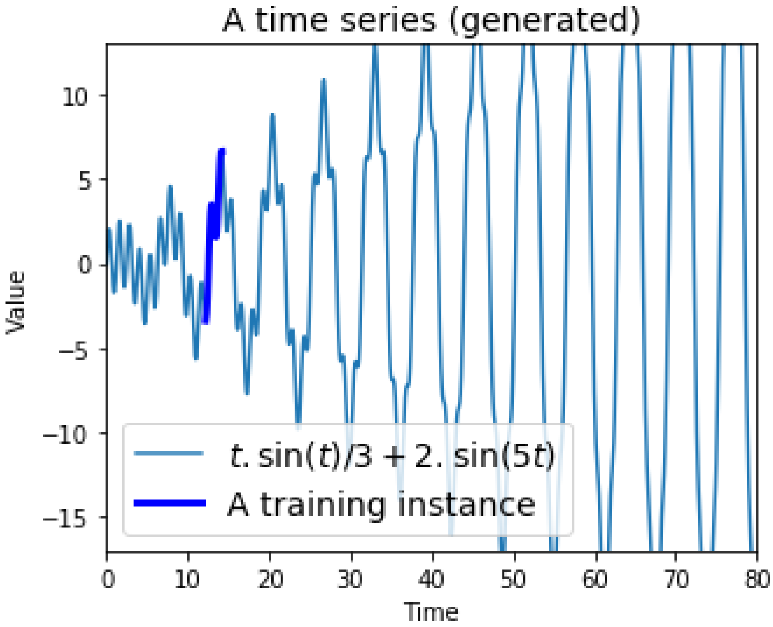


Figure 4.5: Plot of Synthetically Generated Time Series

The function in the equation 4.2 is a uni-variate time-series as the function is completely dependent only on time. This model is also evaluated using mean square error(MSE) as shown in the equation4.1.

During the training process the instances from the train part of the time-series is used. Also during this process, each cell A_k learns to predict some steps ahead depending on created instances.

4.7 RNN for Location Prediction

Generally,some pre-processing is required when a model is trained on the practical data . So this section explains the common pre-processing to be considered for the location prediction problem.

4.7.1 Data Pre-processing

Initially, the samples are grouped by vehicle and the duplicates with respect to the time-stamp are removed. If there are false logs and excess data other than the data required (i.e latitude, longitude, and the time-stamp are the required ones) are removed. Then samples are interpolated so that the samples are spaced with equal time interval. Also, some features like the day of the week and the time of the day are created. The **day of the week** feature is a code based on the weekday, where Monday is 0 and Sunday corresponds to 6 (as shown in the table 4.2a) and **time of the day** feature corresponds 15 minute bin of the current day to which the sample's timestamp belongs (as shown in the table 4.2b). All the features are normalized to unit variance and zero mean. The training data is used to calculate the transformation variables for normalization and these variables are used in the test data. At the end of this process, each vehicle's data would be time-series of its own.

Days of the week	Code
Monday	0
Tuesday	1
Wednesday	2
Thursday	3
Friday	4
Saturday	5
Sunday	6

(a) Days Of the Week

Time of the day	Bin
00:01 – 00:15	1
00:16 – 00:30	2
00:31 – 00:45	3
00:46 – 01:00	4
01:01 – 01:15	5
⋮	⋮

(b) Time of the Day

Table 4.2: Look-up Table: Feature Generation

4.7.2 The Location Prediction Model

As discussed in the section 3.6, cross-validation helps in preventing the over-fitting of train data. The data spans over four months which is 121 days and the division of train/test set is about 90%/10% which corresponds to 109 days of training data and 12 days of test data.

The model is trained to learn movement behavior of specific vehicle, as the movement of each vehicle is a time series of its own. It is logical to represent the data as time-series as the information is ordered. After data pre-processing, each time step comprises four different features: latitude, longitude, the day of the week and the time of the day. The input part of instances created will have all the four features, while the target part will have latitude and longitude. This is an example of multivariate time series prediction (as mentioned in section 4.1).

The architecture described in section 4.4 is used for predicting the future location, but different variants of RNN can solve this problem. So the experiments are done for selecting the suitable one for the practical use case. Then the model is evaluated and compared using the evaluation function.

5

Experiments and Results

This chapter compiles and summarizes the experiments performed and the results obtained. Starting with the proof of concept and then the iterations which lead to the final version of the model.

5.1 Experimental setup

The two main environment that requires to be set up for development and execution of the experiments i.e. hardware and software. Hardware used is an hadoop cluster, where multiple model were trained parallelly if required. The software includes the programming environment and the libraries. Models are implemented in Python 3.4.5 and using libraries like pyspark, tensorflow, scikit-learn, math, matplotlib, plotly, pandas, folium and pickle. `Tensorflow` for the machine learning model and training the models. `numpy` and `pandas` are tools for handling the data. `math` for mathematical operations. `matplotlib`, `plotly` and `folium` for visualizing the data. `scikit-learn` for data-preprocessing. `pickle` is used for serializing and de-serializing the data. `pyspark` is a python version of Spark which exposes to Resilient Distributed Datasets (RDD).

For the purpose of repeatable results, a random seed is set and it is 42.

5.2 Tool: Visualization on the Map

A visualization tool is developed in python using a library called folium (folium is a python wrapper for the leaflet.js). This library allows the binding of the data to the map. This tool is developed to take in the trajectory of paths and plots it on the map. For example, the figure 5.1 shows a mock data plotted on the map.

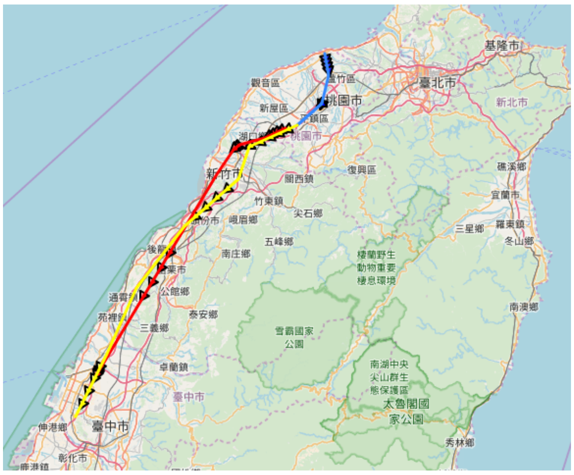


Figure 5.1: A mock Trajectory on a Map created using Visualization Tool

The different colors are used to represent different things like input, target and the output. The arrows denote the direction of the trajectory.

5.3 Learning a Simpler Time-Series

As described in section 4.7.2, the location sequence is a higher dimensional time-series and so it is important to test if the Recurrent neural nets are able to learn the simpler ones as a proof of concept.

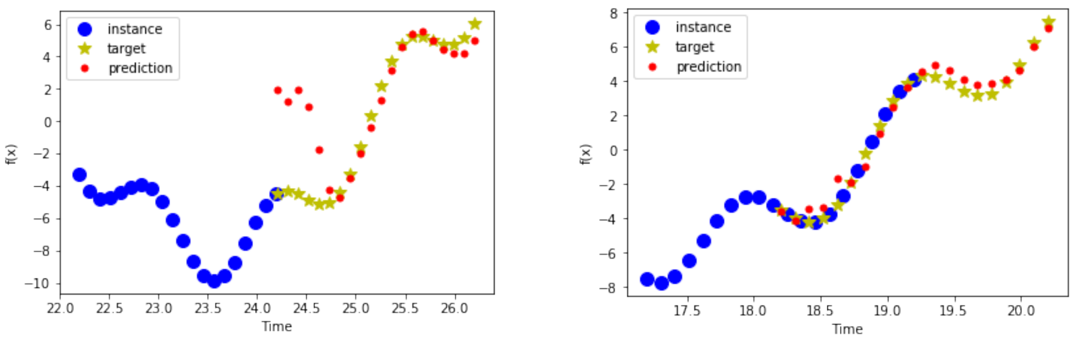
5.3.1 Experiment

The general architecture with simple RNN cell is used as describes in the section 4.6 and the different experiments are conducted with different overlap percentages and other parameters are hand tuned. Experiment1: the number of layer $m = 1$,number of neurons $N = 100$, the number of steps $n = 20$, **overlap** of 0% and a learning rate $\eta = 0.01$.

Experiment 2: the number of layer $m = 1$,number of neurons $N = 100$, the number of steps $n = 20$, **overlap** of 50% and a learning rate $\eta = 0.01$. The difference between the experiment 1 and 2 is testing different **overlap** percentage.

5.3.2 Result

The results were promising that the model was able to learn the time-series function as the errors were low. The models from both the experiments were tested on a random batch of 50 instances and the mean square error is about 5.232 and 1.051 for the model without overlap (experiment 1) and with overlap (experiment 2) respectively. In the case of experiment 2, the part of the output which is not overlapping with the input is used for evaluation.



(a) Experiment 1: With an Overlap of 0% between Inputs and Outputs

(b) Experiment 2: With an Overlap of 50% between Inputs and Outputs

Figure 5.2: Plot on Model's Prediction for simple Time-Series

The mean square error does not give any subjective idea of how better the model performs and so to understand well, one of the samples is plotted as shown in figure 4.5. This shows that the model learns the function and also the initial part of the output is the major contributor to error in the case of experiment 1. An instance of the function is given as the input which is shown as blue large dots in the figure 5.2, the target (the one the model learns to predict) are shown as red small dots and the prediction are shown as the yellow star. The plot 5.2a shows that the initial error is high because the recurrent neural network requires some steps to build the internal states. In the second experiment the region that is overlapping is used for developing the internal states. This experiment proves that the recurrent neural networks are capable of learning a time series function and the overlap helps in building the internal states.

5.4 Data Set

SCANIA provides fleet management services for more than 315,000 vehicles. The services include fleet monitoring, remote diagnostics and many more. The quality of these services are improved by data analytics which requires data like position message (latitude, longitude, time-stamp, vehicle id, ignition, fuel level, ignition event, odometer and more location related data) and data of sensors are collected. The frequency of the position message varies as a function of vehicle operating characteristics.

In this thesis, the models are tested with some vehicles in Taiwan and studies from those are used to generalize the behavior for all the vehicles. The range of time used for this study is four months behavior. For this study, the data is converted into a ***.csv** to make the accessing of data simpler.

A small test is performed just to make sure that the movement performed by the trucks are not random. Sometimes, it is hard to find patterns in the naked eye, where the tool like periodogram is helpful in finding them. It works also for an unevenly sampled data, the figure 5.3 shows a vehicle that has a periodicity of three and half days. A manual test was made to make sure that the samples of vehicle used for the project are periodic on the periodogram scale.

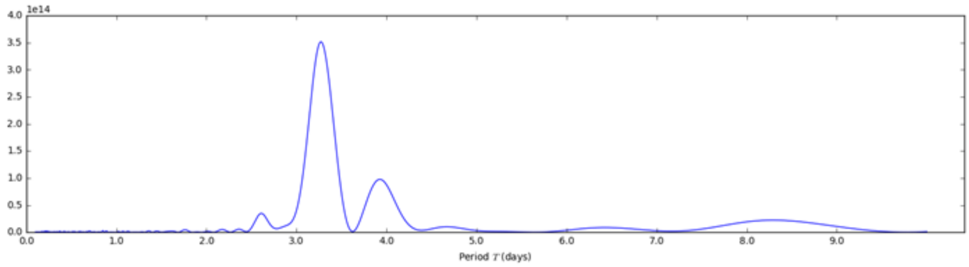


Figure 5.3: Periodogram of One Vehicle Behavior

The pre-processing and split of the data is mentioned in the section 4.7

5.5 Finding the Right Variant (Focusing Bias Problem)

The general architecture described in the section 4.4 is used with the different variants of RNN like simple RNN, BRNN(Bi-directional recurrent neural networks), and LSTM (Long and short-term memory) and its performance is tested. A set of vehicles with different behavior are sampled for this purpose. Also, this experiment aims only to consider the high bias problem (tries to fit the training data) and so the training data is same as the test data. In addition, the results are compared to the baseline model to make sure the machine learning model could learn more.

5.5.1 Experiment

The different variants of RNN like simple RNN, BRNN (Bi-directional recurrent neural networks), and LSTM (Long and short-term memory) are tested. For all the variants the number of time-steps was 100. The hyper-parameters used for simple RNN variant are the number of layer $m = 1$, number of neurons $N = 100$, the number of steps $n = 100$, learning rate $\eta = 0.001$, decay rate 0.3 per 500 iterations and a **overlap** of 50%.

The BRNN variant uses the parameter setting of the number of layer $m = 1$, number of neurons $N = 100$, the number of steps $n = 100$, learning rate $\eta = 0.001$, decay rate 0.5 per 1000 iterations and a **overlap** of 50%.

The LSTM has a hyper-parameter setting of the number of layer $m = 2$, number of neurons $N = 100$, the number of steps $n = 100$, learning rate $\eta = 0.001$, decay rate 0.9 per 5000 iterations and a **overlap** of 50%. These parameters are achieved by manual tuning.

The baseline model takes in 100 time steps and predicts 100 steps.

5.5.2 Results

The model predicts 100 time-steps with an overlap of 50%, which is 50 time-steps that are not common. The 50 time-steps corresponds to 12.5 hours on which the evaluation is made. The performance of the different variant for the different vehicle is compiled into a table. The different variants are RNN, BRNN, LSTM and the baseline. In the results, the vehicle IDs are anonymized for safety reasons.

Vehicle ID	Baseline	Simple RNN	BRNN	LSTM Stacked
1728564	55.068	19.847	26.251	3.548
1773712	23.923	14.755	9.029	5.554
1758104	23.307	13.468	12.863	2.504
1656852	93.393	6.437	8.345	4.686
1686720	43.361	15.359	13.395	6.216
1664121	30.061	19.102	21.474	5.177
1779491	88.371	57.753	66.990	19.661

Table 5.1: Table showing the Performance of different Models (error in kms)

The table 5.1 has listed a set of vehicles and its performance (error in Kms).LSTMs have 6.7 Km average error for predicting the future location of the vehicle. LSTMs have better than the other variants where simple RNNs have 26.5 Km and Bi-directional Recurrent Neural Networks (BRNN) have 26.04 Km of average error. It is clear that the RNN and its variants perform better than the baseline model. Also, it is to be noted that the LSTM variant performs quite best in the current experiment.

5.5.3 Visualization of Results

The tool developed in the section 5.2 can be used to visualize the behavior of different model on the map. It provides better insight of the prediction of the model. The prediction of the RNN, BRNN and LSTM models are visualized on the map. The blue is the input to the model, black is the actual future path and, yellow, orange and red are the prediction by simple RNN, BRNN, and LSTM respectively.

In the figure 5.4, it can be seen that the path taken by LSTM is much closer to the actual path (similar inference as from the table 5.1). Also, it is to be noted that the only a part of input and the output is plotted as plotting the whole trajectory would be hard to visualize here.

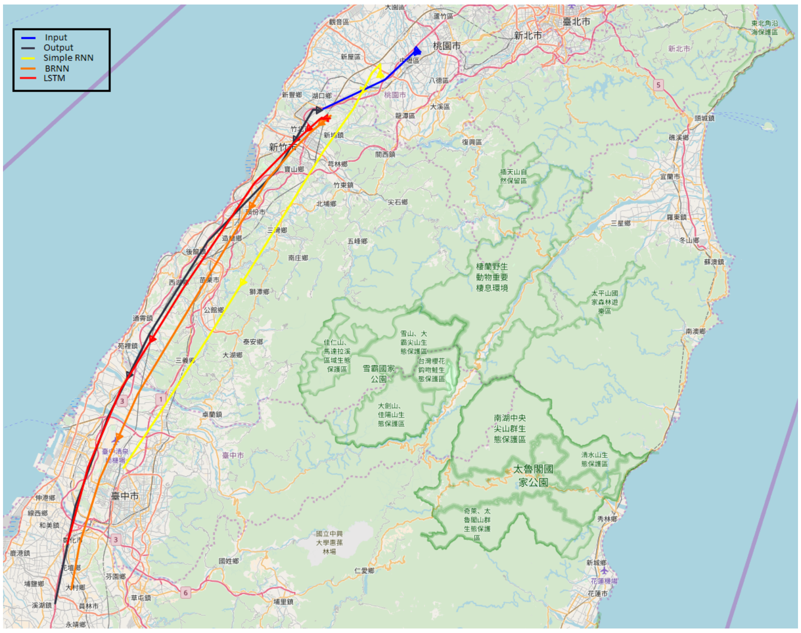
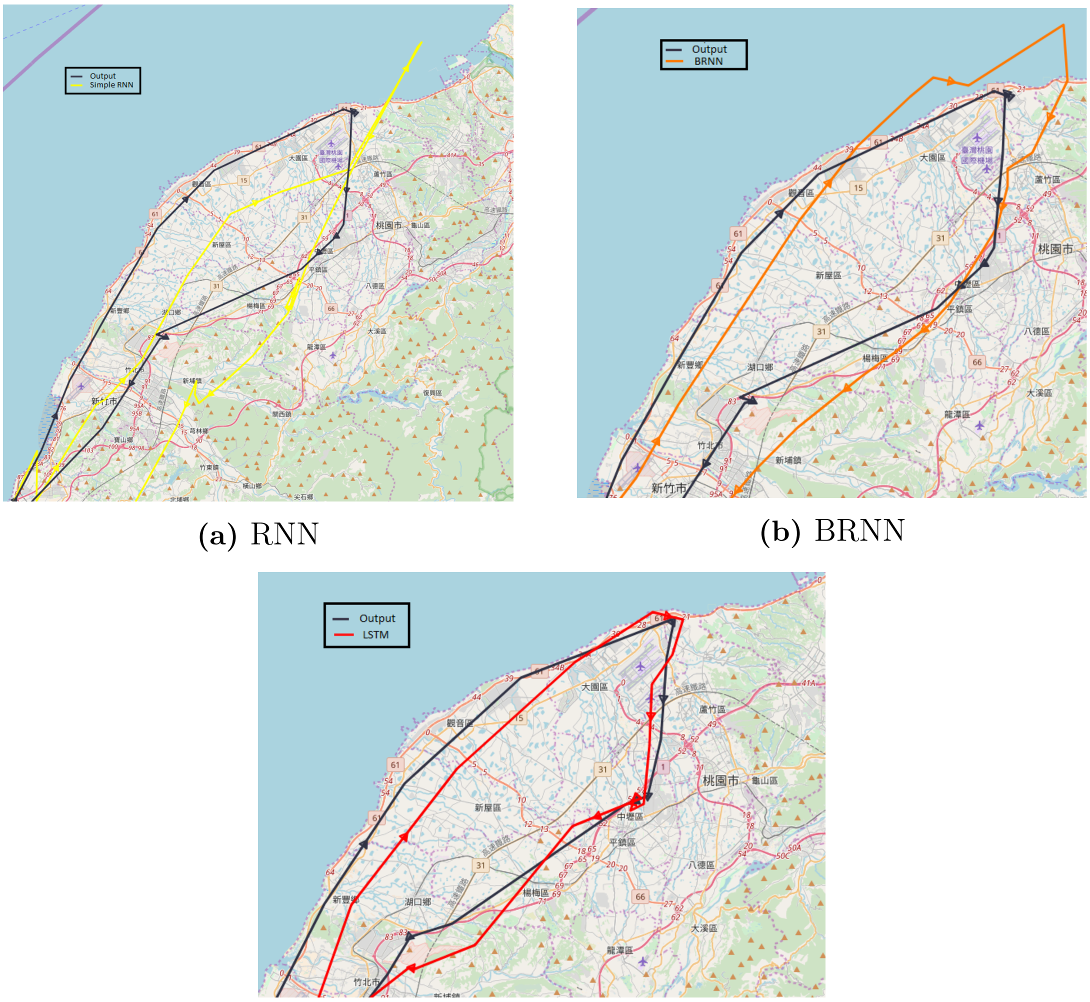


Figure 5.4: Prediction of different models on Map



(a) RNN

(b) BRNN

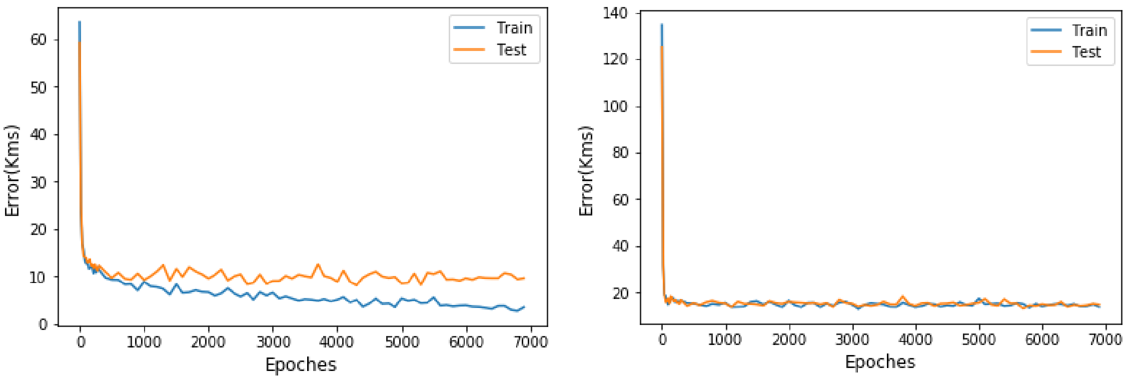
(c) LSTM

Figure 5.5: Complex Behaviour captured by Different Models

Another interesting behavior is shown in the figure 5.5 where the color schema is same as the figure 5.4, the actual path is much complex compared to the one in the figure 5.4. LSTM have closer prediction compared to the simple RNN and BRNN. Sometimes, when the actual point is close to the coast the BRNN and simple RNN has a higher chance of predicting the point in the sea as they have a higher error as inferred from the table 5.1.

5.6 Solving Variance Problem

Once the bias problem is solved, the problem of high variance (solving the problem of over-fitting) can be identified when the test and train data are used. The figure 5.6 shows the training and the test error of the epochs. The first figure we could see that there is an over-fitting in the process of learning. Later when the regularization and dropouts are introduced the learning decreases so that model generalizes (in other words preventing overshoot of test error). The regularization parameter is crucial and so it has to be tuned properly.



(a) Training and the Test loss without Regularization

(b) Training and the Test loss with Regularization

Figure 5.6: Error Vs Iterations for Models with and without Regularisation

5.7 Behavior for Different Delta-time

Delta-time is the time difference between the last input and the predicted point. This plot shows how error varies increase in delta-time.

Each box plot is the samples that are predicted (does not include the sample that overlap with inputs) and they are 15 minutes apart in time. In a box plot, the lines below and above the box are the wiskers, the top and bottom end of the box represent the upper and lower quartile ,and the line in the middle of the box is median. The plot in figure 5.7 shows that the median of error is almost same for all the delta-time for an LSTM location prediction model. It can be observed that the median of error varies between 10% of the whole range for the LSTM model.

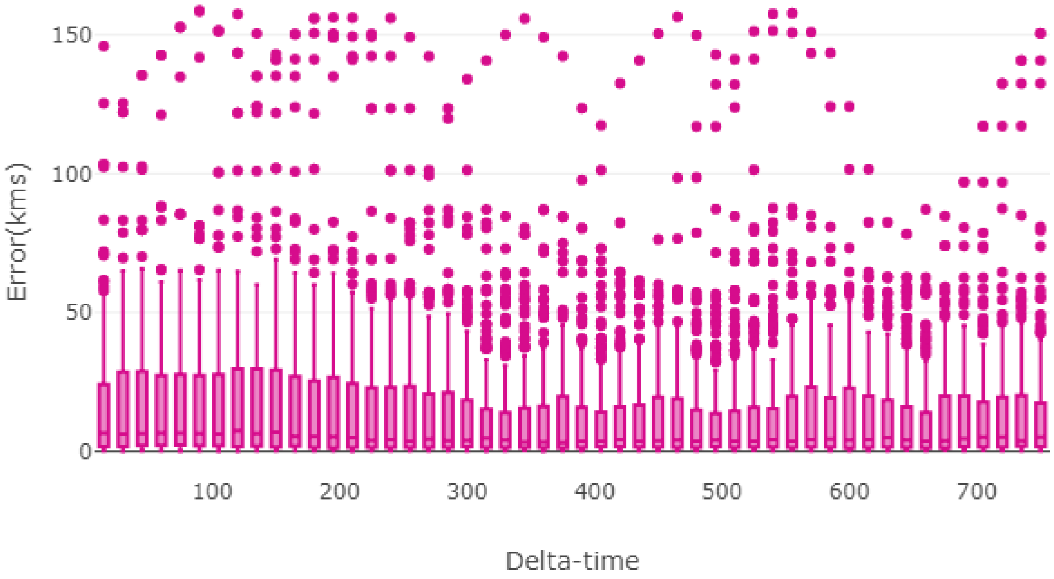


Figure 5.7: Box-plot of Error Vs the Delta-Time

5.8 General Observation on the Performance

From the table 5.1, we can see that the performance of the vehicles varies drastically. The lower error does not mean that the model performs good, maybe the vehicle travel within a small range of distance and the error is huge compared to the range. So to answer the question, how good the model performs for different vehicles. The number of unique geo-hash cells visited is calculated for all the vehicles. The geo-hash with a precision 5 is used, the precision is chosen such that the trajectory is continuous (more about the geo-hash in appendix A.2).

The figure 5.8 shows error in kilometer plotted across the number of different Geo-hash cells the vehicle has visited. The number of Geo-hash cell is a proxy for the area that the vehicle covers. It can be observed in the figure 5.8 that the relation is linear. This shows for a vehicle that covers larger area, more gets the error.

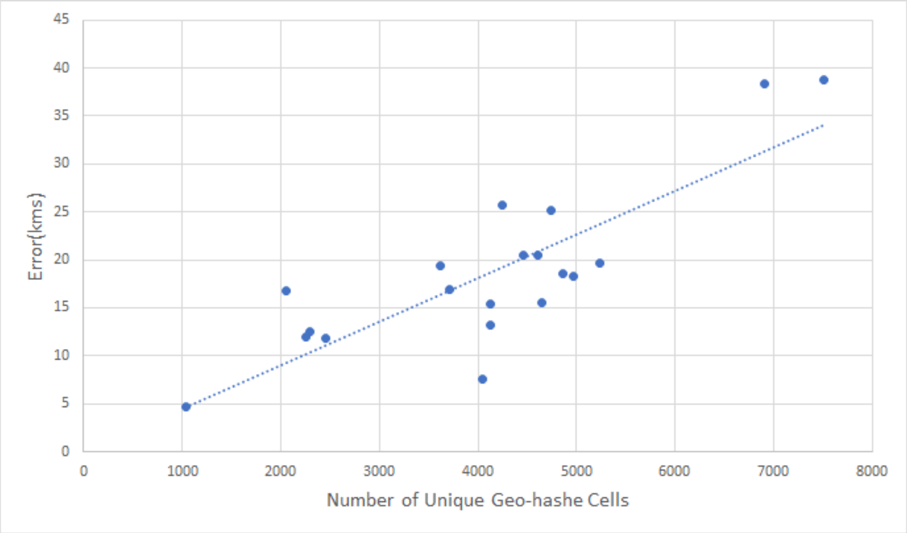


Figure 5.8: Plot of Error Vs the Number of Geo-hash Cells

5.9 Hyper-Parameter Tuning

5.9.1 Experiment

Using the distributed computing, the parameters like number of layers, number of neurons, initial learning rate, regularization parameter, number of iterations and decay rate are tuned using grid search method.
To reduce the overheads the hyper-parameter search grid was minimized to:

$$
\begin{aligned}
\text{Learning Rate} &: 0.1, 0.01 \text{ and } 0.001\\
\text{Number of Neurons} &: 10, 100 \text{ and } 200\\
\text{Number of Layers} &: 1, 2 \text{ and } 3
\end{aligned}
$$

5.9.2 Result

The table 5.2 shows the results from hyper-parameter tuning. Out of 20 different vehicles (i.e. 20 different model) trained, 15 vehicles had improvement in their results. In the table 5.2 if the improvement are positive than the value is shown in bold. The results had a maximum of 23% improvement and a minimum of -8% depreciation.

Vehicle ID	**Without Tuning**	**With Tuning**	**Percent Improved**
194193	20.511	19.461	**5.117**
194474	18.570	16.052	**13.563**
694405	19.641	21.259	−8.240
1439614	20.525	19.500	**4.993**
1521807	7.522	7.594	−0.949
1550355	11.897	12.211	−2.636
1660947	38.776	32.030	**17.396**
1662439	12.562	12.259	**2.414**
1670794	25.155	23.648	**5.991**
1671090	4.736	4.153	**12.306**
1686668	11.759	12.210	−3.840
1686720	25.667	22.331	**12.997**
1707174	15.495	15.035	**2.965**
1708408	15.398	15.242	**1.010**
1708723	19.416	16.577	**14.624**
1712282	16.888	14.600	**13.546**
1728564	38.342	31.010	**19.122**
1757724	18.267	17.012	**6.870**
1757785	13.147	13.928	−5.941
1773712	16.748	12.763	**23.794**

Table 5.2: Table showing the Model performance with and without Hyper-Parameter Tuning

6

Discussion

This chapter discusses the results, comparison of current work with the prior works, business cases for the problem and finally the ideas that the current work could be transformed into.

6.1 Discussion about the Model

Variants experiment: Based on the experimental results, LSTM had better performance when compared to other variants from the table 5.1. The better performance of LSTMs was because of its cell state which enabled it for remembering the dependencies longer through time [2]. Other fact that from the table 5.1 is that the bi-direction variants did not perform better than the simpler version. The reason for no improvement in performance is because the forward direction cell has a higher error at the initial part whereas the reverse cells have at the final part and the final dense layer averages out together.

Hyperparameter experiment: In the table 5.2, it can be seen that sometimes the performance degrades rather improving. One of the reasons behind this is that search space might be small and this could be improved by using random search algorithm.

Delta-time Vs Error: From the figure 5.7, we noted that the error is constant with different delta time. A hypothesis is that the error should increase with an increase in delta-time, e.g. given the history of the location, one could predict location at the next minute more accurate than predicting after ten hours. This hypothesis should be re-tested for the model.

Dataset: In the existing dataset, the most commonly found interval between the sample is 10 minutes which is not good enough to capture some behaviors. For instance, a vehicle which distributes goods within a city would misses more activity at this resolution. This problem can be overcome by increase in the sampling rate. In other words it can be said as the higher sampling rate allows learning of high-resolution behaviors of the truck.

General: Some of the notable observation during the thesis: normalization of the data could significantly improve the results, different architecture of RNN where the training step is in the architecture **many in many out** with the same number of input/output (rightmost in the figure 3.4) and predicted as **many in many out** can have different number of input/output (second from the right in the figure 3.4), the results were worse than the baseline. The problem with this model is that the output converges to one location after some time steps during prediction and so this model could only predict a couple of steps ahead .

6.2 Bench-marking against Prior Work

Most of the methods described in chapter 2 are designed to predict only one-time step ahead, the work [12] used the one-time step prediction model to predict multiple time steps but the model predicts nonmeaningful location after some time as the model learns based on trips.

To compare the results with the prior work, the metric needs to be the same. One of which has the similar metric is the Mathew's [5] model, this method has an average error 143.5 km for predicting the future location. It can be clearly seen that the results from the LSTMs (which had a maximum of 38.77 and minimum of 4.15 km of average error, form the table 5.1) are much better than the Mathew's model. Also, other factor like the dataset plays a huge role in the performance or the complexity of the problem. The work [5] did not mention about the characteristics of the data (like frequency, periodicity etc.), whereas the data used by Fan Wu [12] has a sampling rate of 0.03HZ which is around one sample per 33 seconds. Fan Wu developed an STS-LSTM model which has an average deviation distance of 3.35 meters for MTA-bus data set with the customized pre-processing, which is much better than the performance from the LSTM model developed in this thesis (referring the table 5.1) . There are also considerably many differences for further thinking like the methods cited here tries only to predict one step into the future and the model developed in this thesis tries to predict many steps into future.

One of the major difference between the MTA bus data and the data of the SCANIA's truck is that the movement of the bus is periodic whereas for the truck may or may not be periodic.

6.3 Utilization of Results

Some of the possible business cases are listed below

- Assuming that there is an on-board system capable of predicting faults in near future. Combining with this fault detection system, the short-term future prediction can help in finding the closest workshop in the direction of the future movement depending on the seriousness of the problem.
- In case of a cargo that requires an immediate transport from point A to point B, then the location prediction can be used to find out the suitable vehicle which can deliver according to the immediate needs.
- As the future movement of the vehicle known, the vehicle can be used as the moving advertisement.

- If the movement is quite different from the predicted movement then the detected movement anomaly is notified.

6.4 Ethics

There are many different ethical questions that revolve around location prediction. Some of them are

- As the model predicts for a specific vehicle and in most of the cases the vehicle is driven by the same driver, so this kind of prediction compromises his/her privacy.
- As discussed in the business cases most of them deal with the immediate responses. If the model has a wrong prediction who would bare the costs of the wrong decision.

6.5 Future Works

Let us start with the data set, this project was constrained to the Taiwan data and so it has to be tested on the other data sets as well with different geographical distribution. Then the existing model has several open questions that are needed to be answered

- How the model performance changes with the different overlap? The current models do not motivate the 50% overlap between the inputs and the targets. More tests would help in understanding its influence much better.
- How long in future could the model predict? More tests have to be conducted to define that it can predict for **X** hours and the prediction after which the above the reasonable error limit.
- How new features influence the learning? The current model takes in location as latitude and longitude and it does not know more about the context of the location. For instance, when a truck parked in different parking spots of a warehouse; contextually it is in the same place but with different coordinates. This kind of problem can be considered.

Some general future works other than the current developed,

- A generalized model that captures the general patterns of all vehicles could be built. This generalized models prediction can be used when the vehicle travels in a new route which is different from its regular behavior.
- The behavior of the vehicles change over time. These factors can be incorporated in the future model when developed.

7
Conclusion

In this thesis, an analysis and implementation of the near-time future location prediction of trucks were presented. This problem was approached by considering the ordered location data as a multivariate time-series.

As a proof of concept, a simpler time-series is learned using RNN and the promising results from the proof of concept lead to the further implementation of location prediction model. In evaluations (average distance between the actual and predicted point), LSTM outperformed baseline, simple RNN, and BRNN on location prediction. The prediction error was on average 6.7 km for LSTM. These developed models learn the movement behavior of one specific vehicle. Analyzing the learning of many different vehicles it is observed that vehicles that travel across larger geographic areas, and thus have a more complex movement pattern, also have a higher prediction error. The problem of high variance is approached by using regularization and dropouts.

Further efforts are required in developing a stronger hyper-parameter tuning, in investigating the behavior of the model with increasing delta-time, in testing model with different data (maybe the data from other geographical area) and in making the solution friendlier for regular usage.

Bibliography

[1] A. Karpathy, “The unreasonable effectiveness of recurrent neural networks,” *Andrej Karpathy blog*, 2015.

[2] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] I. E. Burbey, *Predicting Future Locations and Arrival Times of Individuals.* PhD thesis, Virginia Polytechnic Institute, 2011.

[4] A. Sadilek and J. Krumm, “Far out: Predicting long-term human mobility.,” in *AAAI*, 2012.

[5] W. Mathew, R. Raposo, and B. Martins, “Predicting future locations with hidden markov models,” in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp ’12, (New York, NY, USA), pp. 911–918, ACM, 2012.

[6] B. Schuster-Böckler and A. Bateman, “An introduction to hidden markov models,” *Current protocols in bioinformatics*, vol. 18, no. 1, pp. A–3A, 2007.

[7] D. Ashbrook and T. Starner, “Learning significant locations and predicting user movement with gps,” in *Proceedings. Sixth International Symposium on Wearable Computers*, pp. 101–108, 2002.

[8] D. Ashbrook and T. Starner, “Using gps to learn significant locations and predict movement across multiple users,” *Personal Ubiquitous Comput.*, vol. 7, pp. 275–286, Oct. 2003.

[9] Q. Huang, “Mining online footprints to predict user’s next location,” *International Journal of Geographical Information Science*, vol. 31, no. 3, pp. 523–541, 2017.

[10] T. Anagnostopoulos, C. Anagnostopoulos, and S. Hadjiefthymiades, “An adaptive machine learning algorithm for location prediction,” *International Journal of Wireless Information Networks*, vol. 18, no. 2, pp. 88–99, 2011.

[11] M. H. Gohil and S. V. Patel, “Mobile location prediction: Profound study using multi-class random forest predictors,” in *International Journal of Scientific and Engineering Research*, vol. 8, (New York, NY, USA), pp. 67–74, 2017.

[12] F. Wu, K. Fu, Y. Wang, Z. Xiao, and X. Fu, “A spatial-temporal-semantic neural network algorithm for location prediction on moving objects,” *Algorithms*, vol. 10, p. 37, 2017.

[13] A. Al-Molegi, M. Jabreel, and B. Ghaleb, “Stf-rnn: Space time features-based recurrent neural network for predicting people next location,” in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–7, Dec 2016.

[14] H. Kaaniche and F. Kamoun, “Mobility prediction in wireless ad hoc networks using neural networks,” *Journal Of Telecomunications*, vol. 2, 2010.

[15] X. Song, H. Kanasugi, and R. Shibasaki, "Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level," in *International Joint Conference on Artificial Intelligence*, pp. 2618–2624, 2016.

[16] K. Bandara, C. Bergmeir, and S. Smyl, "Forecasting across time series databases using long short-term memory networks on groups of similar series," *CoRR*, vol. abs/1710.03222, 2017.

[17] J. Lv, Q. Li, and X. Wang, "T-CONV: A convolutional neural network for multi-scale taxi trajectory prediction," *CoRR*, vol. abs/1611.07635, 2016.

[18] J. A. Hertz, *Introduction to the theory of neural computation.* CRC Press, 1990.

[19] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd ed., 1998.

[20] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," *Backpropagation: Theory, architectures, and applications*, vol. 1, pp. 433–486, 1995.

[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[23] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[25] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in neural information processing systems*, pp. 950–957, 1992.

[26] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

A

Appendix 1

A.1 Havesine Distance

Haversine distances determines to find the circle distance given the latitudes and the longitudes. It is solved using spherical geometry.

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{A.1}$$

where d is the distance, r is the radius of the sphere(the earth in the case of measuring geographical distances) and (ϕ_i, λ_i) is the latitude and the longitude of the location i.

A.2 Geo-hash

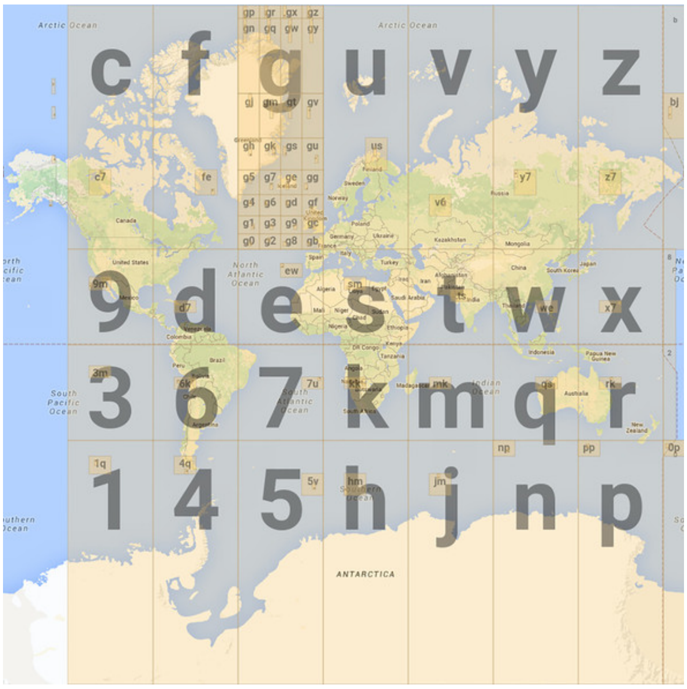


Figure A.1: Illustration Geo-Hash

Geo-hash is away of representing location with cells. There are different levels of precision in geo-hash. The geo-hash of precision one is denoted by on alphabet and each alphabet added with increase in precision. Each alphabet can denote one in 32 locations. The figure A.1 shows how the geo-hash the divides the world into different regions.