Forecasting the next position of the user element using deep learning algorithms

UNIVERSITY OF SURREY

Ponlasit Poopipatpol

Supervised by Professor Ning Wang

ID: 6606435

Abstract

User mobility forecasting is one of the well-known challenging problems in a wide range of modern technologies, particularly the fifth-generation (5G) mobile communication systems.

The use of forecasting user mobility is critical for allocating resources of the beam as well as optimizing the beam pattern to always stay connected with the user element (UE) and provide the highest possible throughput to the UE.

To address this issue, the deep learning method was used. Deep Learning is one area of machine learning in artificial intelligence that enables the ability to learn and improve automatically through the use of data. Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU) were employed to forecast the mobility pattern of the UE based on historical sequential samples of coordinate data generated by GPS and simulator.

To effectively extract optimal performance from machine learning algorithms, evaluation metrics such as accuracy, mean square error and root mean square error was chosen to assess machine learning quality.

The deep learning algorithms are conducted on my laptop and implemented in Keras, a deep learning API written in Python, running on top of TensorFlow framework.

Once implementation framework of all deep learning models presented in this dissertation has been trained on both realistic dataset and simulated dataset, It was found that the bidirectional LSTM and GRU and RNN are the best approach in term of accuracy and fast training and they are capable of forecasting next position of the user element with one step forward and multi-step prediction.

Keywords: Deep learning, Time series Forecasting, User Mobility Prediction, Long Short-Term Memory, Recurrent Neural Networks, Gated Recurrent Unit

Acknowledgements

I would like to express my gratitude to my industrial supervisors, Mr. Hequn Zhang and Mr. Kexuan Sun at Viavi solutions company for giving me this opportunity to work on this project. They provided constructive guidance to make this project goes in the right direction and I would like to express gratitude and thank my supervisor, Professor Ning Wang for his kind support, expertise and excellent supervision in motivating me to accomplish this dissertation. This dissertation would not possible to achieve without his continuous guidance and support.

I would also like to thank my friends and my family for their support throughout the research.

Ponlasit Poopipatpol, August 2021

Contents

Table of Contents

1.Introduction

An emerging mobile communication technology fourth generation (4G) mobile technology and fifth generation (5G) mobile technology has become the mainstream communication technology, allowing many people all over the world to connect over long distances. As time goes on, mobile phones are increasingly being developed as minicomputers that can be used for a variety of purposes, especially when combined with 5G, which provides higher data rates with seamless roaming. As a result of the widespread use of smartphones and location-based services all over the world, there has been a massive and rapid increase in mobility data over time. Many organizations, operators, and vendors are looking for ways to optimize their telecommunication equipment in order to support a diverse set of mobility data. To meet these increased data demands, identifying the characteristics of user mobility pattern predictions has been proposed as one method of introducing a new method of supporting in allocating beam resources as well as optimizing the beam pattern to always stay connected with the user element (UE) and provide the highest possible throughput to UE for the growing volume of mobility data. User mobility forecasting is one of the well-known challenging problems in a wide range of modern technologies, such as handover management in mobile networks [1], resource planning [2], individualize recommendation systems, urban planning [3], mobility management in mobile communication system and smart transportation. In general, the use of mobility forecasting differs depending on the application scenario that the organisation wishes to achieve. In particular, in the case of mobile communications, forecasting the next position of user elements in the near future greatly aids firms in the telecommunications fields in order to plan for mobility management and beam allocation to provide the highest possible throughput to mobile users. Also, this prediction application can be viewed as the problem of time series in which the trajectory refers to a data points in time series of positions separated by a fixed sampling time interval.

Numerous approaches for predicting movement have been developed, including frequent patterns mining [4], Markov models [5], and other mathematical methods. However, most of these methods are focused on extracting the location prediction, which is indeed a problem in multi-classification. This is because trajectories data can be interpreted as location extraction from Global Positioning System (GPS) format, and locations may remain the same for numerous consecutive time-steps when the sampling time interval is small, but may change between two adjacent time-steps when the sampling time interval is big [6]. As a result, they are unable to accurately reflect user movement patterns in the real-time. Furthermore, trajectories with continuous position coordinates make it difficult to specify the level of discretization, and when there are a large number of data points, it's possible that it could have an impact on the forecast's accuracy, causing many people to concern about it. Nevertheless, the benefits of tracking user movement patterns in the majority of scenarios are enormous, and this has inspired many businesses to do so.

To address this issue, deep learning is a key. Deep Learning is one area of machine learning in artificial intelligence that enables the ability to learn and improve automatically through the use of data. As the trajectories composed of continuous coordinates and this sort of problem is related to time series regression prediction problem with non-linear methods, Recurrent Neural

Networks (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU) have demonstrated their excellence in a diversity of time series problems not only in natural language processing (e.g., sentiment analytics [7], speech emotion recognition [8]), but also in some other fields (i.e. statistical machine learning used in integrated anti-spam system [9], stock market analysis using supervised machine learning [10]). These models are interesting algorithms for user element forecasting problem and they are employed to forecast the mobility pattern of the UE based on historical sequential samples of coordinate data generated by GPS and simulator provided from Viavi in this dissertation.

2 Description of the problem

2.1 Aim

The primary goal of this dissertation is to design and explore a deep learning algorithm to forecast the next position of the User element, which can be used for forecasting the next position of the User element, which can then be used to optimize the beamform for maximum coverage. The model used sequential samples of coordinate data generated by the simulator as input and forecasted the next position of the User element based on the historical data or the input. In this dissertation, the deep learning model was used as the main model, and it was attempted to optimize the model with optimal hyperparameters to achieve the best accuracy. The deep learning algorithms are conducted on my laptop and implemented in Keras, a deep learning API written in Python, running on top of TensorFlow framework.

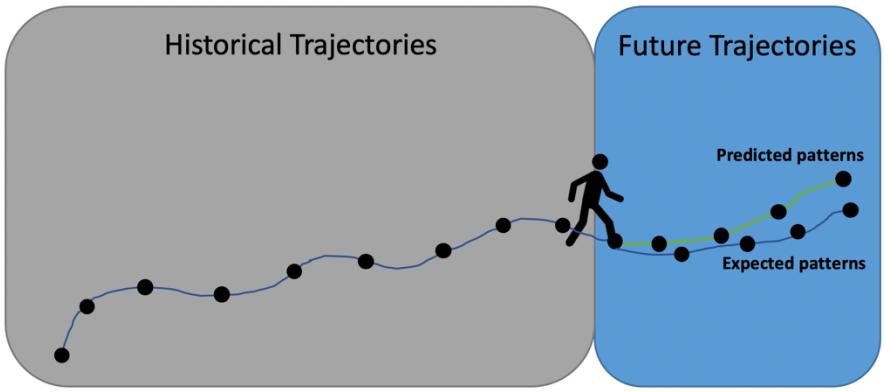


Figure 2.1: Problem description over trajectory forecasting. The goal is to obtain trajectory predication as close to the expected trajectory patterns as possible. Predictions are based on the historical data of x and y coordinates as inputs.

2.2 Objectives

The main scope is to explore the use of deep learning algorithms such as Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN) and Gated Recurrent Unit (GRU) for forecasting the next position of the UE using the training data consisting of the x and y

coordinates collected from logged data over 1 month period and the dataset generated from simulator provided by Viavi company. The dissertation work is limited to human behavior and does not cover every possible scenario. The aim of the dissertation is not to predict when the cell site should hand over the UE to other cells. Instead, the beam selection will use machine learning prediction data to optimize the beam to transmit the best signal power to UE. The model simply generates the next position of the UE based on a mathematical function that fits the model to the training dataset. The simulation environment is not taken into account in this dissertation. The models only concern the next point of predictions.

The objectives are divided into the 4 categories listed below.

1. To adapt the theoretical techniques of machine learning models with application to forecast the next position of user element in the future.
2. To optimize, tune and train the model in order to forecast the next position of user element as accurately as possible.
3. To evaluate and compare the performance of deep learning models in forecasting the possible position of user element in the future.
4. To develop the machine learning model to forecast multi-step sequences of position utilizing numerous sets of historical sequences of the position as input.

2.3 Workflow

The study on which this dissertation is based used deep learning models to anticipate the next position of the User element. Following some preliminary research on the use of deep learning models, trajectory forecasting and time series literatures, it was decided that the chosen deep learning models, such as LSTM, RNN, GRU, stacked LSTM, and stacked GRU models, be implemented in the work and that further research be conducted to investigate how these models can be applied with time series problems in order to forecast the next position of the UE, with the goal of identifying suitable models by comparing the results of these machine learning models. TensorFlow, a popular and user-friendly framework, will be utilized to construct deep learning models. This study adheres to the cycles of the cross-industry standard process for data mining (CRISP-DM) methodology. Begin with identifying the study topic, then attempt to comprehend and analyse the datasets before doing data preparation, constructing the models, train and evaluating them. After that, the machine learning environment was set up in which we could train and evaluate our deep learning models implementation. Ultimately, the analytical results, assessment of overall success and how each experiment contributed to the accomplishment of primary objectives, limitations and future work on forecasting UE's next position were concluded.

2.4 Structure of the dissertation

The dissertation is divided into six chapters: Introduction, theory, methods, results, discussion, conclusion and future work. The first chapter of the thesis discusses the introduction, problem description, aim, objectives and related work. The second chapter discusses the background of

this dissertation and theoretical basis as well as research in time series forecasting, mobility prediction, and neural networks. The Third chapter describes the research methods used in the study, followed by models used in the experimental method. Chapter Fifth is a section to present the results of the research methods. Then, the results are analyzed and discussed in Chapter sixth. Finally, chapter seventh concludes the dissertation's conclusion and future work.

3 Background & Related Work

3.1 Statistical learning

Statistical learning is a machine learning framework derived from statistics and mathematical analysis to provide a foundation for machine learning and this idea was introduced in the late 1960s [11]. Statistical learning was thought to be a theoretical topic of estimating the given set of data until the 1990s [11]. Then, new forms of learning algorithms emerged in the mid-1990s. As a result, statistical learning theory became not just a tool for theoretical study, but also for developing practical algorithms for estimating multidimensional functions [11]. There are a variety of applications that have made use of the statistical learning theory. For example, computer vision, speech recognition, bioinformatics, and regression estimation.
Its primary objective of statistical learning is to provide a framework application from understanding the data, making predictions, making decisions or create the models from a set of data to solve the inference problem [12]. This learning may be broken down into several different areas such as supervised learning, unsupervised learning, online learning, as well reinforcement learning. The best practical learning algorithm falls in the supervised learning category which will be explain in detail in section 3.2.

3.2 Machine Learning

Machine learning algorithms are classified into 3 categories. They are:

1. Supervised Learning: This system requires input data as independent variables and output data or the target variable as a dependent variable. Using all of these variables, the algorithm can generate a mathematical function that learns how to map the input to the output. In order to achieve the desired level of accuracy on training data, the algorithm would require appropriate parameters as well as the training time to train the model on the training dataset. The term supervised learning is referred to as a teacher supervising the learning process [13]. In other words, the system will make the predictions based on training data and the system will be corrected by the correct answers or the outcome variables, just as a teacher would. The model examples of supervised Learning are Random Forest, Decision Tree, linear regression, Logistic Regression, and KNN [14].

   Supervised Learning can be divided into 2 categories:

a) Classification: This algorithm is used to predict when the output data is recognized as specific categories. For instance, classifying whether the shape is "circle" or "rectangle".
b) Regression: This algorithm is used to predict the relationship between dependent and independent variables when the output data is to real value. For example, predicting house sale prices

3. Unsupervised Learning: This system does not require any target or output labeled data to predict. In contrast to the supervised learning described above, which requires the outcome variables to correct answers, this algorithm uses only the input data to segment the population into different groups. In other words, The algorithm must gain insights from the input and discover its own to demonstrate the interesting structure in the data [13].
   Unsupervised Learning can be divided into 2 categories :
   a) Clustering: The model is used to solve the problem associated with segmenting the data group. Example: Clustering customers who brought the products based on the Recency, Frequency, Monetary Value.
   b) Association: The model is used to solve the problem associated with discovering rules that describe a large portion of data. Example: The people who play "golf" tend to be "rich".

3. Reinforcement Learning: Reinforcement Learning is one of the areas in Machine Learning where focusing on taking suitable action corresponding to maximize reward[13]. The agent is used by the algorithm to learn how to find the best possible behavior or path in an uncertain, potentially complex environment by employing trial and error to solve the problem and get the rewards or penalties which follows the design policies for the actions it performs[15]. It could be said that this type of learning is bound to learn from its experience. For example, we consider the scenario of playing chess as an example of the reinforcement algorithm, the chess pieces represent an agent while the optimal movement of the chess pieces is determined based on previous experience of getting a maximum reward.

3.3 Artificial Intelligence

An artificial intelligence (AI) is a relatively new field, having only been around for about 60 years, and it is a collection of disciplines, ideas, and methodologies (mathematical logical and probabilistic reasoning; statistics and probability; computational neurobiology and computer science) that aims to mimic the cognitive abilities of an individual human [16]. As a result of its development during the Second World War era, computers can now do more complicated jobs that could previously only be performed by a person [16].

In the early 1950s, John Von Neumann and Alan Turing did not invent the phrase artificial intelligence, but they did lay the foundations for the technology underlying it: they made the shift from computers to 19th century decimal logic which dealt with values from 0 to 9 and

machines to binary logic in 1950 which rely on Boolean algebra, dealing with more or less important chains of 0 or 1 [16]. By formalising the design of our modern computers, the two researchers proved they were universal machines that could execute anything was programmed into them. However, in his renowned 1950 article "Computer Machinery and Intelligence," Turing proposed the topic of machine intelligence for the first time, "Can Machines Think?". Turing also introduced the "game of imitation," in which an individual should be capable of telling the difference between talking to another human and talking to an artificial intelligence when communicating via telephone [17]. It is frequently recognised as the genesis of the questioning of the border between the human and the computer, and it formed the primary objective basis for Artificial Intelligence [17].

3.4 Artificial Neural Networks (ANN)

Perceptrons

The perceptron, which is composed of a single neuron, is the basic building unit of an ANN. Equation 3.1 describes the perceptron mathematically:

$$y = \sigma\left(\sum_{j=1}^{n} \omega_j x_j + b \right), \sigma(x) = \begin{cases} 1, & \text{if } x>0 \\ 0, & \text{otherwise} \end{cases} \quad (3.1)$$

It takes the sum of each element x_j of a real-valued input vector, and multiplies it by the weight vector element ω_j. As a result of this, the neuron's output y is determined by a bias b that is added to the sum and provided as input to an activation function σ (x) that yields one if the input is higher than zero and zero otherwise. Figure 3.1 is a diagram of the same perceptron in its schematic form.

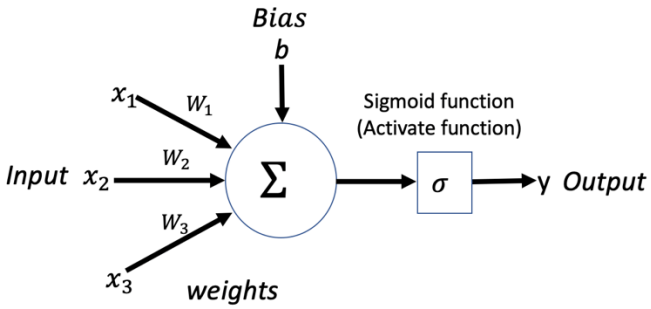


Figure 3.1: Diagram of a basic perceptron with a single input layer.

3.5 Deep Learning

Deep learning neural networks, also known as artificial neural networks, seek to imitate the human brain using a mix of data inputs, weights, and bias. Deep learning is made up of several layers of interconnected nodes, each of which builds on the preceding layer to enhance and optimise the prediction or categorization [18]. There is no exact definition of how many layered are required in deep learning networks. A deep neural network's visible layers are its input and output layers. While, a hidden layer is a layer between the input and output layers where neurons take in a collection of weighted inputs and create an output via an activation function. The data is ingested for processing by the deep learning model in the input layer, and the final prediction or classification is generated in the output layer.

Neural networks have several advantages, including the ability to store data on the entire network, the ability to work with insufficient knowledge, the ability to train the machine, the ability to parallel process as these networks grow in size, gradual corruption , and the corruption of one or more artificial neural network cells has no effect on output production which makes deep learning networks better at tolerating faults [19]. Furthermore, the fact that each layer can learn various abstractions of the data by its own is the exact reason why deep networks outperform some shallow networks [20, 21, 22].

The downside is that it requires a large amount of data to perform better than other techniques, the computational complexity grows with network depth, which is not ideal [23]. Additionally, it does not have the right standard theory and approaches in selecting the proper deep learning tools. It necessitates that the developers must have a thorough understanding of topology, training techniques, and other factors in order to construct a strong deep learning model [23].

3.6 Multi-layer Neural Network

A multi-layered neural network builds by employing numerous neurons that are linked to one another and arranged in layers, as shown in Figure 3.2. Circles and edges depict neurons and their connections, respectively. The output of each neuron is computed in the same way that perception is.

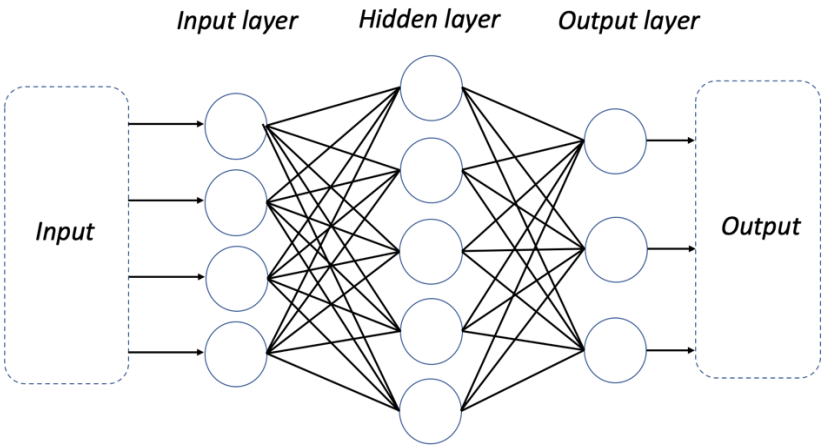


Figure 3.2: Architecture of Neural Network.

Input Layer: It is responsible for transferring input data to the subsequent levels.

Hidden Layer: All computations are performed in this layer, and the results are sent to the next levels.

Output Layer: This layer employs the activation function, which is in charge of converting the output response to the proper format.

3.7 Activation Functions

Activation functions are used within ANN neurons to transform the induced local field V,

$$V \equiv \sum_{j=1}^{n} \omega_j x_j + b \tag{3.2}$$

Activation functions are used to compress the local field to a certain interval, usually between zero and one. As a result, each neuron's output produces the output within the same range. All neurons in a network utilise the same activation function except for the output layer. Some prominent non-linear activation functions for the hidden layers are listed below.

Sigmoid

$$\emptyset_{sigmoid}(x) \equiv \sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

Hyperbolic Tangent

$$\emptyset_{tanh}(x) \equiv tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.4}$$

ReLU

$$\emptyset_{ReLU}(x) = \max(0, x) \tag{3.5}$$

Non-linearity is an important requirement for activation functions. Non-linearity improves a neural network's capacity to approximate complicated functions. Differentiability is also another requirement that must be present in activation functions.

The activation function for the output layer is determined by the type of output required, as the range of an ANN's output must match the intended value range. A linear activation function is commonly used in regression.

Linear

$$\emptyset_{linear}(x) = cx \tag{3.6}$$

The softmax activation function is frequently used for classification, where the output maps the probability of belonging to a certain class.

Softmax

$$\emptyset_{softmaxi}(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{3.7}$$

where $\varphi_{softmaxi}$ is the probability of belonging to class i.

3.8 Backpropagation

Backpropagation is the most widely used algorithm for training ANNs. To minimise an objective function, this technique employs the gradient decent optimization process. Backpropagation is most often employed in supervised learning since it needs knowing the objective function's derivative with respect to the output, which can only be computed if the intended output value is known. The algorithm is divided into two stages: forward and reverse. Data propagates across the network and creates an output during the forward stage. The output is compared to the desired value, and the error is computed. The gradients of each weight in the network are determined during the backward step. A modest amount of negative gradient is then applied to all weights. When the weight is updated, the learning rate determines how much weight is updated. It is faster for the network to learn if its learning rate is high, and it is faster for the network to learn with a high learning rate and more accurate with a low learning rate.

3.9 Weight Initialization

Before starting the training of an ANN, the network weights must be initialized. The gradient descent optimization methods descend down the steepest gradient of a function to find the local minimum. Unfortunately the function which is optimized when training an ANN is not convex, which means that the starting value of the model parameters matter. Depending on their initialization the optimization algorithm will find different minima. One naive way to initialize the weight would be to set all of them to zero. Unfortunately, setting all the weights to the same value will make the network unable to learn. This is due to the fact that the error backpropagated during learning is proportional to the weights of the network. Initializing all the weight to the same value will therefore make the backpropagated errors identical and accordingly all the weights, when updated in the same iteration, will be updated the same magnitude. This is why weights are randomly initialized when creating a new ANN. The random initialization of the weights are sampled from some distribution, usually a Gaussian or a uniform distribution with zero mean and small variance to avoid large weight initialization. Another way to initialize the parameters for an ANN is to use a pre-trained ANN which has completed training for another task and start the learning on the new task from there.

3.10 Vanishing and exploding gradient

The phenomenon of vanishing and exploding gradients is a typical problem while training neural networks, particularly deep neural networks. The gradients of the weights in the first layers of a regular feed forward network are reliant on the gradients and values of the weights in the subsequent layers of the network. The gradients in the network's layers near the input layer are calculated by multiplying multiple gradients and weight values at the network's end.

This issue stems from the backpropagation algorithm's usage of the chain rule. If, the unrolled RNN for long sequences is so deep and the chain rule for backpropagation involves partial derivative products, the gradient at early time slices is the product of many partial derivatives. In reality, the number of components in the product is proportional to the length of the input-output sequence for early slices [24]. This is an issue because, unless the partial derivatives are all near to one, their product will either become extremely tiny, i.e. vanishing, when the partial derivatives are less than one, or very huge, i.e. exploding, when the partial derivatives are greater than one. As a result, learning becomes either incredibly slow in the vanishing scenario or highly unstable in the exploding case [24].

This is a specific problem with recurrent neural networks; for lengthy input-output sequences, RNNs have difficulty modelling long-term dependencies, or the relationship between elements in the sequence separated by extended periods of time but it is solvable by employing Long Short-Term Memory Cells.

3.11 Data Normalization (Min-Max Scalar)

As the range of raw data varies greatly, the trained model will not operate effectively without data normalisation. If we do not normalise the data, the model will be dominated by variables with a higher scale, which will have a negative impact on model performance. This necessitates data normalisation. The range of independent data may be normalised using Min-Max Scaling. It is also known as data normalisation in data processing and is usually conducted during the data preparation stage.

The simplest and most basic technique of rescaling is min-max scalar, often known as min-max normalization. The feature range to scale in [0, 1] for defaults and [1, 1]. As a result, we have smaller standard deviations, which can dampen the influence of outliers. The target range is chosen based on the nature of the data. The general formula for a min-max value of [0, 1] is as follows:

$$x_{normalized} = \frac{(x - \min(x))}{(\max(x) - \min(x))} \tag{3.8}$$

where $\min(x)$ is the minimum value of variable x over every sample and $\max(x)$ is the maximum value variable x over every sample.

The formula for rescaling a range between two values [a, b] is:

$$x_{normalized} = a + \frac{(x - \min(x))\,(b - a)}{(\max(x) - \min(x))} \tag{3.9}$$

where a and b are the min and max values of the scale.

3.12 Time Series Forecasting

Time Series Forecasting is the process of forecasting the future observations of a given phenomenon solely based on previous patterns of the same event [25]. It is extensively applied in holistic decision-making analysis [26]. Time series analysis has become increasingly important in recent decades in a wide range of applications, including, forecasting the spread of COVID-19, forecasting the prices of Bitcoin, and forecasting the depletion level of stocks in stores.

Time series data is defined as a sequence of data patterns that are collected and stored in time order over a period of time. There are two major components in time series data: time stamp and a numerical value for each time stamp.

The advantage of time series data is that it provides the researcher with the necessary information without imposing any minimum or maximum time constraints during data

collection. This has resulted in the extensive use of time series data across many organizations because it allows for the prediction of future values using the historical data.

Time series data can be classified into four categories which are deterministic time series, non-deterministic time series, stationary time series, and non-stationary time series.

A deterministic time series can be described in terms of mathematical as a Taylor series expansion. It has no probabilistic nor random elements. The values of past and future derivatives at the time can defined always forecast its future pattern and past pattern [27].

A non-deterministic time series is one that has a random aspect preventing it from being explicitly described. In terms of statistical, this can be defined by the data, for instance, averages of various types and probability distributions [27].

A stationary time series is one that means, variance, and autocorrelation structures do not change over time [28]. In other words, a stationary time series can be described as a flat looking series with no trend, no seasonality, constant variance and constant autocorrelation structure over time [28]. Its future pattern is simply predicted as the statistical properties is the same as its past.

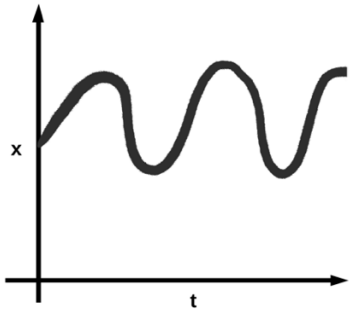


Figure 3.3: Stationary time series

A non-stationary time series is the opposite of a stationary time series. Its statistical properties change over time due to the presence of numerous factors such as trend, cyclical, seasonal, and random components [27].

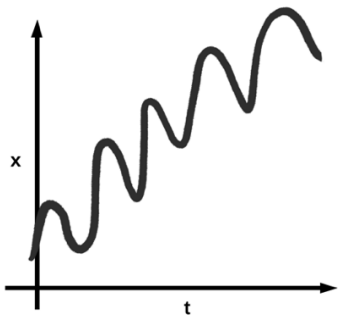


Figure 3.4: Non-stationary time series

Time series analysis methods can also be classified as linear or non-linear, as well as univariate or multivariate.

A linear time series is one in which each data point can be thought of as the equation for a straight line or a linear combination of past or future values or differences. It is an equation with a maximum degree of 1.

While, nonlinear time series does not form a straight line, but rather a curve. A nonlinear equation has a degree of 2 or greater, but not less than 2. It has characteristics that linear algorithms cannot model.

A univariate time series has only one time dependent variable. It consists of a single time observation collected in a series over equal time intervals.

Month	Sales
1960-01	6550
1960-02	8728
1960-03	12026
1960-04	14395

Figure 3.5: Example of Univariate Time Series

A multivariate time series has multiple time dependent variables. Variables in a multivariate time series data set are affected not only by their previous values but also by other variables in the data set.

No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	cbwd	Iws	Is	Ir
1	2010	1	1	0	NA	-21	-11	1021	NW	1.79	0	0
2	2010	1	1	1	NA	-21	-12	1020	NW	4.92	0	0
3	2010	1	1	2	NA	-21	-11	1019	NW	6.71	0	0
4	2010	1	1	3	NA	-21	-14	1019	NW	9.84	0	0

Figure 3.5: Example of Multivariate Time Series

There are two types of time series forecasting: one-step forward forecasting and multi-step forward forecasting.

One step forward forecasting is a prediction of a sequence of values in a time series based on the historical data at the next time step because only a one time step is to be predicted.

Multi-step forward forecasting is a prediction of a sequence of values in a time series based on the historical data for multiple time steps rather than the next time step.

3.13 Literature Review & Related Work

Some theoretical mobility models, such as the Density-EPR model [29], Markov Diary Generator [29], the Random Walk model [30], Gauss-Markov model [31], Levy-Walk model [32], and others, have been proposed to imitate the movement of human mobility and generate synthetic trajectory patterns corresponding to mathematical algorithms and theoretical methods of a single moving object and human behavior. However, these mathematical models have several limitations, and they may not be able to cover all of the trajectory cases of various movements in a complex and real-world context, making their applications in critical practice unfeasible.

Furthermore, most of the researchers have been concentrating on the development of machine learning algorithms using decision tree and K-nearest neighbor (KNN) that can forecast the human route in a certain area using GPS data acquired from volunteers and then mapping it out with the location of a place on the map [33], [34]. However, these strategies need discrete locations and they only reveal approximate patterns of human movement that is not appropriate for continuous coordinate trajectories with short sampling intervals. They may not be ideal for critical cases that require the exact position of the users. Markov-based models [35], [36] are another extensively used mobility prediction approach. A hidden Markov model (HMM)-based trajectory prediction is a technique for tracking transition states from one position to another position.

The work in [37] applies the algorithm of deep learning model like recurrent LSTM neural networks in predicting vehicle trajectory. The proposed method is designed to create a model that takes sequential samples of logged ego vehicle sensor data as input and outputs trajectory predictions for the ego vehicle's future path. Additionally, the work aims to forecast the lane change based on statistical fitting on the provided training data, with the expectation of ending up in the same lane as the ground truth trajectories. However, the proposed method is specifically designed for the vehicle trajectories in a lane change scenario. Alahi et al. [38] propose a social LSTM network for predicting pedestrian trajectory. However, it is only capable of forecasting human trajectories using images from video recorder in a limited range scene, such as square and intersections.

Another study [39] used time series theory combined with a deep learning model to predict Hotspots in order to manage the beam for adaptive virtual small cells in 5G networks. This paper extensively used a deep learning model to forecast population density in a virtual small cell, which was then used to adjust the beam forming as highly directional transmission as possible to cover those cells. Correspondingly, an interesting work [40] used deep learning methods with time series theory to predict user mobility by finding suitable deep learning models to predict the next 96 eNBs based on available historical data and determining which model for each user element provides better prediction in term of accuracy.

Time series theory and trajectory prediction offer a wide range of applications in 5G networks, including radio resource allocation [41], caching decision at the wireless edge [42], mobility management [43], and more. For instance, consider the application of machine learning models in forecasting the future signal strength in order to optimize the handover state in mobile systems, which corresponds to a time series issue that uses previous signal values for each individual user element to predict the future signal value [44]. Furthermore, knowing the intentions of adjacent automobiles through trajectory prediction application is crucial due to the stringent safety standards of autonomous driving and advanced driver aid systems [45], [46]. As a result, trajectory prediction is a topic that has to be fully researched. Nonetheless, the benefits of this is substantial in the vast majority of cases, and this has motivated numerous firms.

4 Methods

This chapter describes the research methodology used for the dissertation, including the choices of learning models in the experiment and the methodology involved in training and mapping the model with historical sequences of coordinate samples of trajectory to forecast the next trajectory of user mobility. This section begins by preparing and exploring the datasets collected from the Python simulator provided from Viavi and datasets of Geolife GPS provided from Microsoft Research Asia that were used to train the models. It then describes the pre-processing and filtering of the dataset, as well as how the data was split into training and validation sets. The network architecture of the proposed models is then demonstrated. Furthermore, the training procedures for the model are presented in terms of the loss function and hyperparameter optimization. In the final section, which establishes evaluation methods and performance metrics, the models are evaluated.

4.1 Experiment

4.1.1 Software Environment

The machine learning models were developed using Jupyter Notebook, an open-source, interactive web application known as a computational notebook that allows researchers to combine their software code, computational output, explanatory text, and multimedia resources in a single document [47]. It has the ipython kernel to execute any software code developing under Python programming language [48]. The open-source libraries TensorFlow and Pandas, Matplotlib, Sklearn, Numpy, Keras, Unit, and Haversine were employed in the Jupyter Notebook to gain the insight from the dataset and build machine learning models to forecast the next movement of the user mobility. Keras, an open-source library of deep learning components written in Python, serves as the primary library for developing machine learning models associated with various types of neural networks in the dissertation.

4.1.2 Hardware Environment

The machine learning models were developed in the MacBook pro 16 system consisting of Intel Core i9 processor 2.3 GHz 8-Core and 16 GB 2667 MHz DDR4. The system is running on macOS Catalina 64-bit operating system.

4.1.3 Datasets

To assess the performance of the machine learning models for forecasting the next movement of the user mobility, two types of datasets, which are a dataset generated by a simulator and a realistic dataset, are used in this case. It is critical to test with two types of datasets in order to validate and compare the results of the machine learning model performance in a realistic and test environment.

- Dataset generated from simulator provided by Viavi:

The dataset produced by the Python simulator is based on the cubic spline interpolation theory, which is a mathematical approach widely used to create new points inside the bounds of a collection of known points. These new points are function values of an interpolation function also known as a spline, which is made up of several cubic piecewise polynomials.
This simulator imitates a type of user mobility pattern in 2D position (x, y) which are the coordinates in a simulation area of 2000m × 2000m with 10-way points. Figure 4.1displays a user element's simulation area and one example route. The dataset generated by this simulator consists of four columns which are x coordinate in meter unit, y coordinate in meter unit, speed in m/s unit and timestep in second unit.

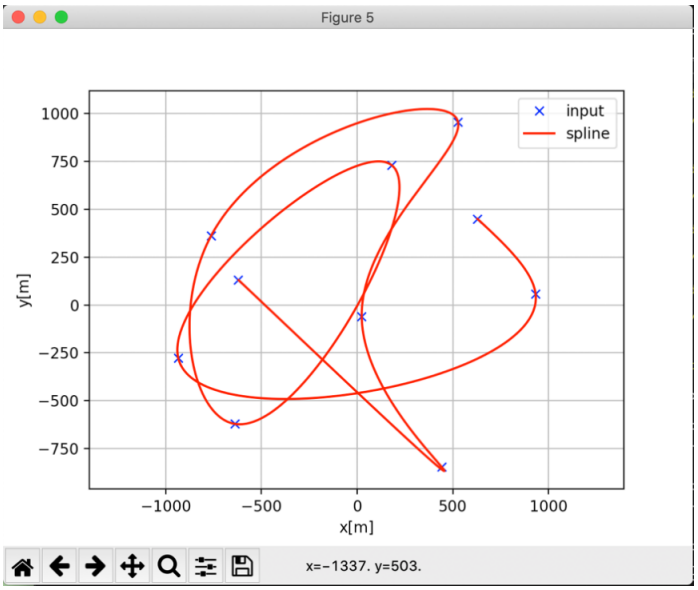


Figure 4.1: An example of a single UE trajectory generated by the simulator where the data in the x-coordinate lies in the x-axis and the data in the y-coordinate lies in the y-axis.

- Geolife dataset collected by Microsoft Research Asia:

The huge real-life GPS trajectory dataset gathered by Microsoft Research Asia from the Geolife project in Beijing, consisting of 182 users and includes 17,621 trajectories with a total distance of 1,292,951kilometers and a total duration of 50,176 hours with varied sampling rates during a five-year period from April 2007 to August 2012 [49]. There are several timestamp points for each trajectory, and GPS-functioned phones capture the latitude, longitude, altitude path and label data. This dataset recoded a wide range of users' outside movements, covering not just daily routines like going home and going to work, but also certain recreational and sporting activities like shopping, sightseeing, eating, hiking, and cycling. This trajectory dataset may be used to a wide range of research topics, including mobility pattern mining, user activity identification, location-based social networks, location privacy, and location suggestion. To work with large and untidy raw datasets, a preprocessing phase is required.

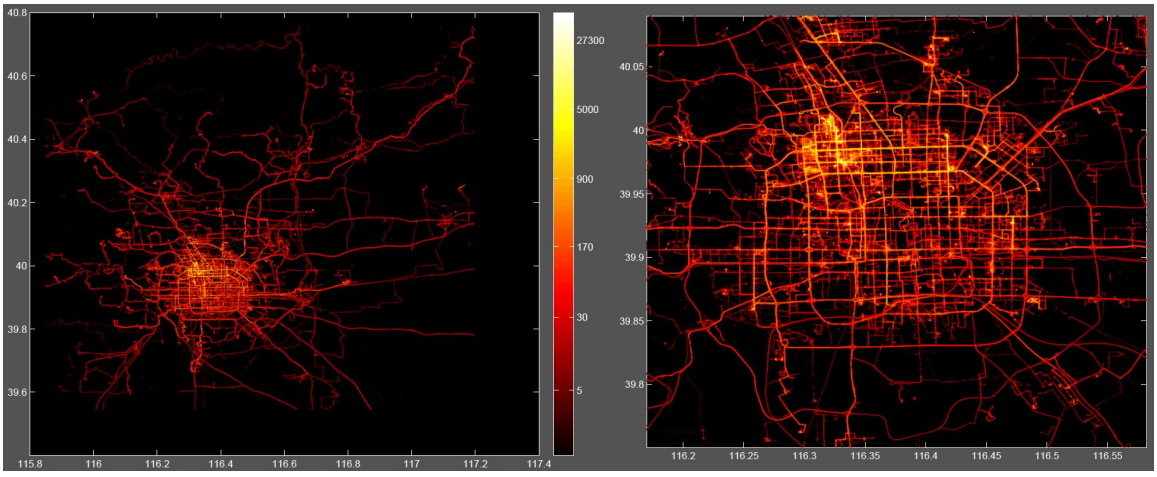


Figure 4.2: Human trajectories overview in Beijing and the Human trajectories within the 5th Ring Road of Beijing

4.1.4 Data Preprocessing

The model will be trained on sequences of features built up of samples from real data logged from GPS and simulated data from python simulator.
The raw data of the realistic dataset contains seven columns. The selected logged data is the result of filtering one person, which we choose user id 167 from all during an 8-day period in April 2008. The dataset consists of about 3,387,303 recorded files where each file contains roughly 5 seconds of moving.
Input features comprise: Longitude, Latitude, Altitude, Date_Time, Id_user, Id_perc and Label.

The raw data of the simulated dataset, on the other hand, has four columns. The logged data consists of about 11,079 recorded files, with each file representing one timestep of movement.
Input features comprise: X coordinate, Y coordinate, Speed and Time.

Once the raw data of the realistic dataset is extracted from the parser function and the simulated data is gathered, they are to be analyzed and the features that are required for the time series are to be selected and transform them into supervised problem. The flow of data preprocessing operations in each stage is depicted in Figure 4.3 below.

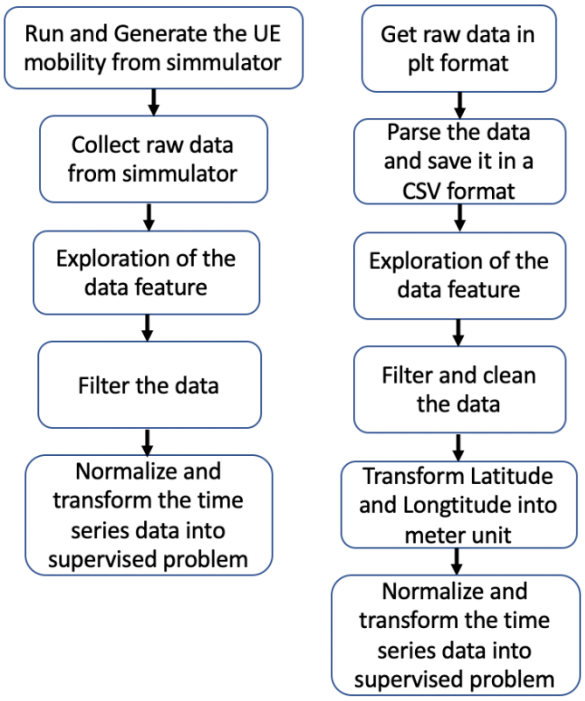


Figure 4.3: Flow of data preprocessing processes, where the left picture representing the process for Simulator dataset and the right image representing the process for realistic dataset.

```
traj.tail()
```

	Latitude	Longitude	Altitude	Date_Time	Id_user	Id_perc	Label
3387298	40.070186	116.314153	-45.0	2008-11-29 02:01:31	179	20081129015805.plt	bus
3387299	40.070193	116.314041	-48.0	2008-11-29 02:01:33	179	20081129015805.plt	bus
3387300	40.070224	116.313923	-51.0	2008-11-29 02:01:35	179	20081129015805.plt	bus

```
T1.head()
```

	X	Y	Speed	Time
0	148.832079	-240.485966	4.517157	1
1	148.664153	-235.971996	4.517093	2

Figure 4.4: Raw dataset, where the upper picture representing the realistic dataset and the bottom image representing the simulated dataset.

The essential features for conversion into a supervised problem corresponding time series problem are latitude and longitude of the realistic dataset, which will then be converted from

the geographic coordinates represented by latitude and longitude into two-dimensional plane coordinates by spatial coordinate projection in meter units as x and y coordinates. While, the essential features for simulated data are in the form of x and y coordinates already.

4.1.5 Split training and testing datasets

To train and evaluate the performance of machine learning models, the data generated by a function that turns the original sequence of data into supervised learning problem is divided into training and testing datasets. In the instance of the realistic dataset, the data consists of 12,836 time slots collected over an 8-day period commencing on April 5, 2008 and ending on April 29, 2008. While the simulator data combines three trajectories into one, the total number of points of trajectory is 11,080 time periods. The test dataset for validating the models will be divided accounts for 20% of the total, whereas the train dataset that goes into machine learning for training the models accounts for 80% of the whole.

4.1.6 Models Implemented

In this study, six deep learning models were employed for testing, however the LSTM model was chosen as the main model.

The reason choice for choosing the LSTM model as the main model is that

LSTM (Long Short-Term Memory) was developed base on a Recurrent Neural Network (RNN) architecture and it is widely used in natural language processing and time series forecasting. It has computational power and the ability to work with larger datasets more effectively [50]. It even outperforms other models when dealing with sequence prediction problems due to its ability to feedback and store the past information to its model. Its model promises to learn, rather than having this context pre-specified and fixed, the context needed to generate predictions in time series forecasting issues [51]. LSTM can also be used to handle the problems regarding vanishing gradients that arise as a result of repeated weight adjustments as a neural network trains.

Advantages:

1. It is capable of addressing the vanishing gradient problem, which makes network training difficult for long sequences of information.

2. It has the ability to deal with larger datasets more effectively than other models

3. The performance of LSTM is largely dependent on the selection of a number of hyper-parameters, which must be done with care in order to achieve decent results [52].

4. LSTM is capable of learning nonlinearities which means that it does not require data to be in the form of stationary for forecasting.

Disadvantages:

1. In some cases, it fails to completely remove vanishing gradients.

2. It requires a lot of resources and time to train the model.

3. It gets influenced by different random weight initializations and thus behaves similarly to a feed-forward neural net.

4. It is prone to overfitting if it has to deal with a small dataset and using the dropout algorithm to mitigate this problem is difficult.

4.1.7 Recurrent Neural Networks

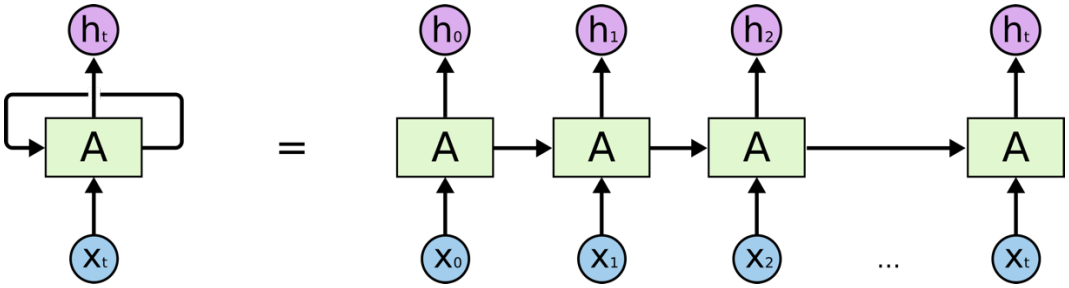


Figure 4.5: The repeating module in a standard RNN. http://colah.github. io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png

The recurrent neural network (RNN) is an extension of the traditional feed-forward neural network where the output from the previous variable-length sequences are fed as the input to the current variable-length sequences. The RNN can handle variable-length sequences by employing a recurrent hidden state, the activation of which is dependent on the preceding time [53]. A generative RNN's output is the probability distribution of the next element in a sequence given its current state. RNN is a logical extension of feed-forward neural networks [53].

[53] Junyoung Chung et al. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. 2014. arXiv: 1412.3555 [cs.NE].

$$h_t = \sigma_h(i_t) = \sigma_h(U_h x_t + V_h h_{t-1} + b_f) \quad (4.1)$$

$$y_t = \sigma_y(a_t) = \sigma_y(W_y h_t + b_f) \quad (4.2)$$

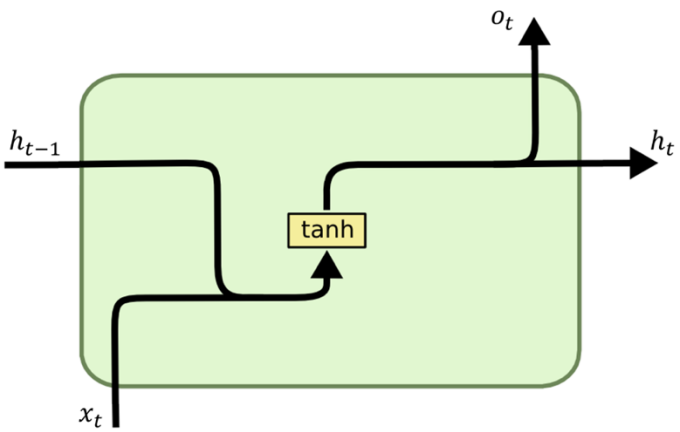


Figure 4.6: Standard RNN. http://dprogrammer.org/rnn-lstm-gru

4.1.8 Long Short-Term Memory Cells (LSTM)

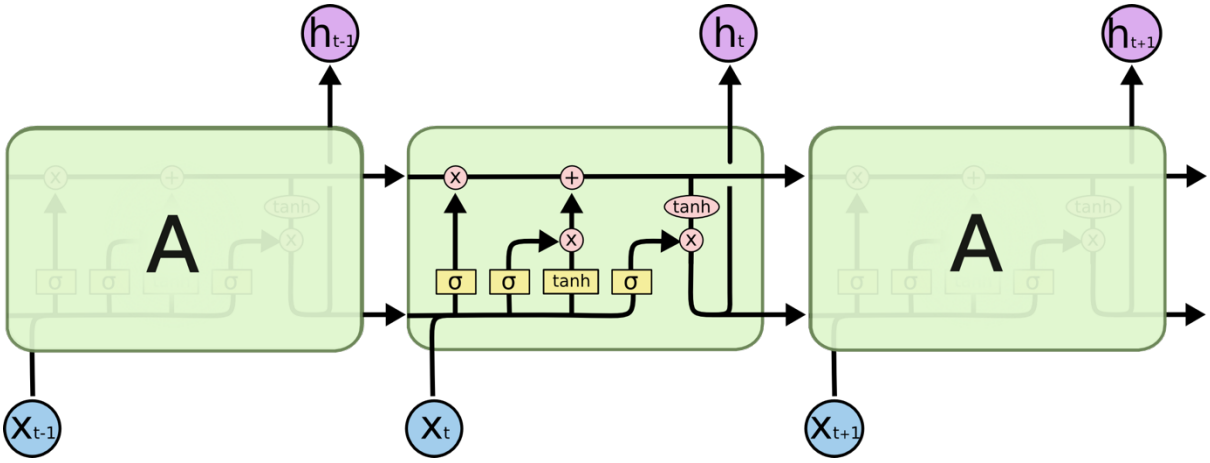


Figure 4.7: The repeating module in an LSTM containing four interacting networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/ LSTM3-chain.png

To tackle the vanishing gradient problem, Sepp Hochreiter and Jürgen Schmidhuber [54] created an RNN called long short-term memory (LSTM) in 1997. LSTMs make long term memory possible by eliminating the problem of disappearing or expanding gradients. To overcome this problem, LSTMs turn every recurrent network node into a memory cell instead of an activation node Like RNNs, LSTMs use both their prior and present states to calculate their output, as shown in Figure 4.7.

C = Cell state
f = Forget gate
i = Input gate
o = Output gate
h = Hidden state

σ = Sigmoid function
tanh = Hyperbolic Tangent function

The LSTM consists of a cell state, a forget gate, an input gate, and an output gate.

Cell state: This cell state is in charge of long-term memory and used to decide whether to forget previous cell state and add the new value to the current cell state or not. The forget gate changes the cell state, whereas the input modulation gate adjusts the cell state. According to the equation, the forget gate multiplies the previous cell state and adds the input gate multiplied by the current cell state.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4.3}$$

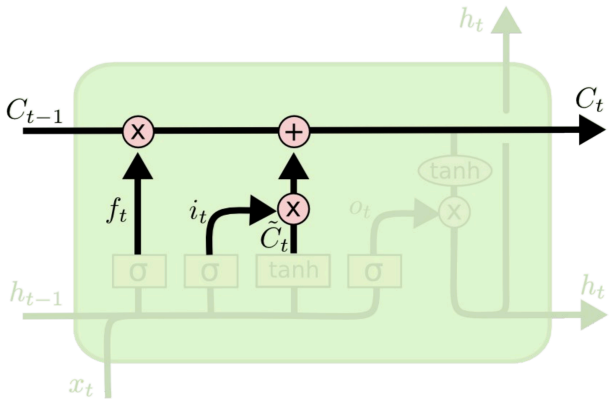


Figure 4.8: Cell state architecture. http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-C-line.png

Forget gate: This gate is where the LSTM decide whether to keep or get rid of cell state. It looks at hidden state of $t-1$ and input which is x_t, and outputs a number between 0 and 1 by sigmoid function for each number in the cell state C_{t-1}. By this, 1 represents "completely keep this" while a 0 represents "completely get rid of this."

$$f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f) \tag{4.4}$$

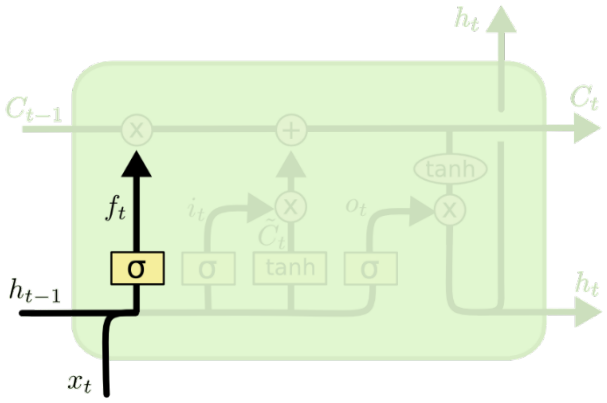


Figure 4.9: Forget state architecture. http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-focus-f.png

Input gate: This input gate is used to decide what new information to be stored in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values to be updated. Second, a tanh layer creates a vector of new candidate values, C_t at t time, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4.5}$$

$$C_t = tanh (W_c \cdot [h_{t-1}, x_t] + b_c) \tag{4.6}$$

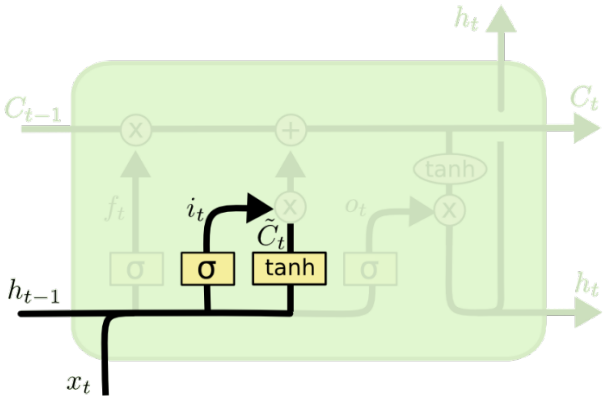


Figure 4.10: Input state architecture. http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-focus-i.png

Output gate: This gate handles the information about which value from the matrix is to be transferred to the next hidden state from all available values.

$$o_t = (\sigma (w_o \cdot [h_{t-1}, x_t] + b_o) \quad (4.7)$$

$$h_t = o_t * tanh(C_t) \quad (4.8)$$

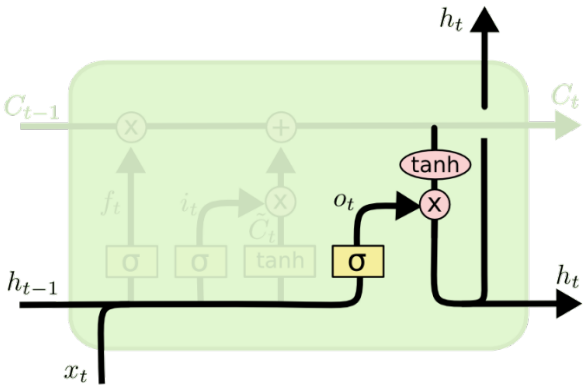


Figure 4.11: Output state architecture. http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-focus-o.png

LSTM has been utilised in sophisticated machine learning applications such as classification application, time series analysis, text recognition, speech recognition, and so on.

4.1.9 Stacked LSTMs

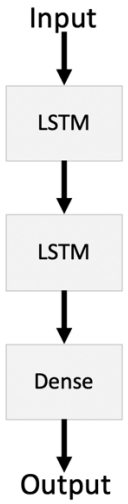


Figure 4.12: Stacked LSTM architecture.

Stacked LSTMs or Deep LSTMs are an expansion of the usage of the LSTM model in multiple layers, with multiple hidden LSTM layers, each of which contains multiple memory cells [55]. It is a reliable technique in solving sequence prediction problems. Having stacked LSTMs increases model complexity and accuracy. Additionally, when the input sequence has already

been processed by a prior LSTM layer, the current LSTM can provide a more sophisticated feature representation of the current input.

4.1.10 Bi-directional LSTM

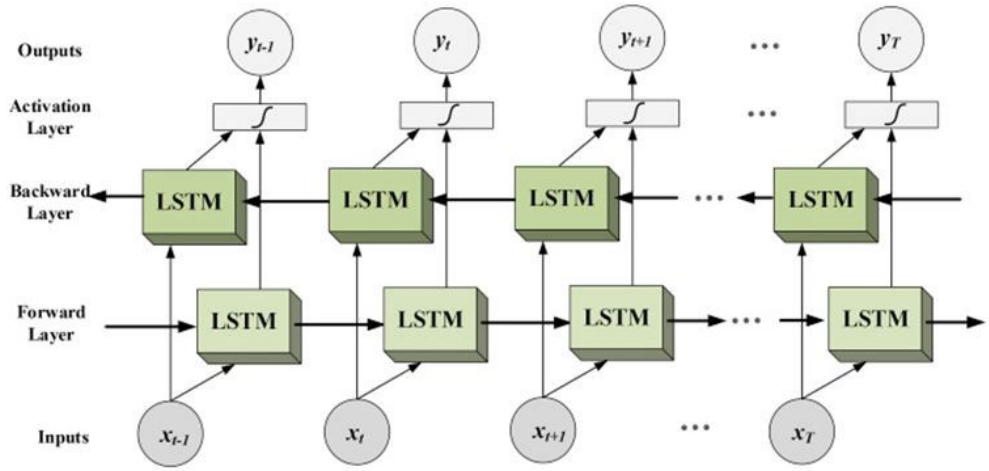


Figure 4.13: The Bidirectional LSTM architecture.
https://miro.medium.com/max/4800/0*ZsTT3zzTNGF-6OsR.jpg

Bi-directional LSTM is a hybrid of Bi-Directional RNN and LSTM. It entails replicating the network's first recurrent layer such that there are now two layers side by side, then feeding the input sequence as-is to the first layer and a reversed duplicate of the input sequence to the second [56]. Bidirectional LSTM is not suitable for all sequence prediction tasks, although they do give superior outcomes in particular areas. Using bidirectional will run inputs in two directions, one from past to future and one from future to past.

4.1.11 Gated Recurrent Unit (GRU)

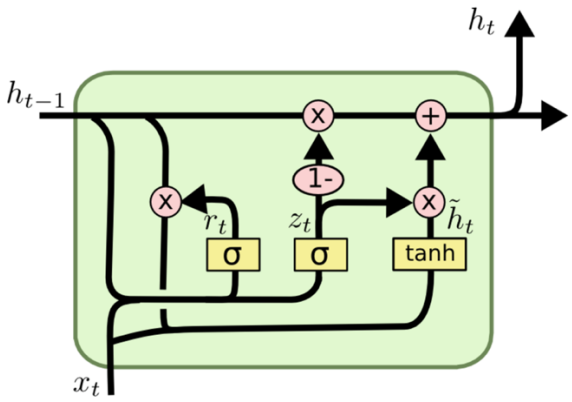


Figure 4.14: The GRU architecture.
https://miro.medium.com/max/1724/1*GSZ0ZQZPvcWm TVatAeOiIw.png

Chung, Junyoung suggested a Gated Recurrent Unit (GRU) in [57,58]. A variation of LSTM, it is one of the most often used. Update gate was created by combining the forget gate and the input gates, as well as integrating cell state and hidden state. To control information flow inside the GRU, it includes gating units, but no distinct memory cells [53]. The model is unique because it can be trained to retain information from a long time ago without being washed away by time or to eliminate information that is irrelevant to the prediction.

x_t = input vector
z_t = update gate vector
r_t = reset gate vector
h_t = output vector
σ = sigmoid function
tanh = Hyperbolic Tangent function

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{4.9}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{4.10}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t \cdot h_{t-1}, x_t]) \tag{4.11}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{4.12}$$

The model determines the information from past stages that has to be passed on to the future by update gate z_t. Likewise, the model chooses how much previous data to forget with the aid of the reset gate r_t.

4.1.12 Stacked GRUs

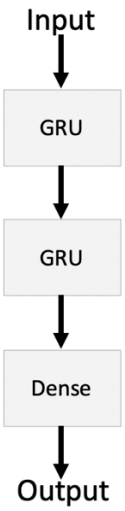


Figure 4.15: Stacked GRU architecture.

Stacked GRUs, also known as Deep GRUs, are an extension of the GRU model's use in multiple layers, with several hidden GRU layers, each of which has multiple memory cells.

The same justification for stacking an LSTM model can be applied to the Stacked GRUs approach, since it enhances model complexity, accuracy and is a clearly reliable approach for dealing with sequence prediction difficulties

4.2.1 Framework Design

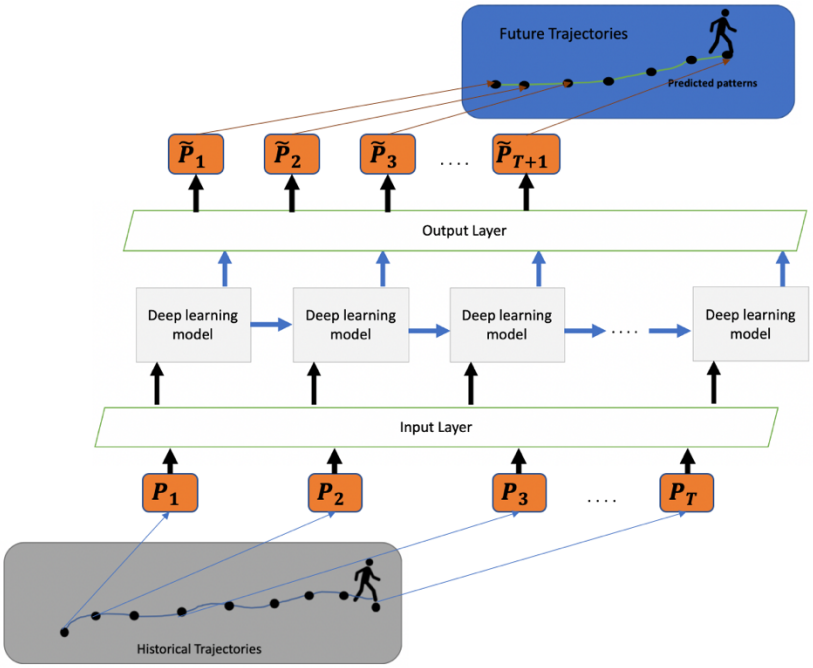


Figure 4.16: Basic deep learning framework for User element prediction.

Figure 4.16 depicts the establishment of a deep learning design framework for a user. This deep learning system perfectly captures the user element's paths based on previous trajectories. Three key phases are involved in the prediction process. First, a fully connected input layer using neuron networks processes the supplied trajectory, mapping each x and y coordinate to a deep learning model or in case of multi-step, each x or y coordinate is mapped with each deep learning model. The processed sequence is then transmitted to prediction core, a deep recurrent neural network depending on the choice of model such as LSTM, stacked LSTMs, bi-directional LSTM, GRU, stacked GRUs and RNN receives the historical sequences as input and feeds its output to the next layer. Finally, a fully connected output layer maps the output of the deep learning model at each time-step T to a two-dimensional coordinate $\tilde{p}_{T+1}$ as the predicted position of the next time-step, and the prediction sets of sequence model $\tilde{p} = \{\tilde{p}_2, \tilde{p}_3, \tilde{p}_4 \dots, \tilde{p}_{T+1}\}$ is then produced from the deep learning.

4.2.2 Loss Function

The loss function determines if a trajectory forecasting is excellent or terrible by comparing the output of the prediction with the correct answer. The loss function utilised in this study is batch of the *MAE* of a predicted trajectory, which is defined as

$$loss\ function = MAE = \frac{1}{n_b}\left(\sum_{j=1}^{n_b}\left(\left|\ {y_i}^j - \ \tilde{y}_i{}^j\right|\right) + \sum_{j=1}^{n_b}\left(\left|{x_i}^j - \ \tilde{x}_i{}^j\right|\right)\right) \quad (4.13)$$

where n_b is the batch size, n is the number of points predicted in the trajectory, (y_i , x_i) are the ground truth coordinates and ($\tilde{y}_i$, $\tilde{x}_i$) are the predicted coordinates.

4.2.3 Hyperparameters

The number of layers and the number of hidden units were determined by the usage of hyperparameters. In each model, the hyperparameters we pick vary based on how well the model has been tuned. For example, two LSTM cells connected in series as stacked LSTMs model with 64 units in the first layer and 128 units in second layer. The learning rate we choose is the default values. As large batch size sped up the computing time but compromised the convergence of the network which lead to the use of a batch size of 72.

4.2.4 Weight initialization and Optimization

According to section 3.9, weights are derived from a distribution and TensorFlow default, which we apply in our models. In backpropagation, the weights of the network are updated in order to minimise the loss function, which drives ($\tilde{y}_i$, $\tilde{x}_i$) to approach (y_i , x_i), causing the model to provide predictions that are close to correct values. We have opted to utilise Adaptive Moment Estimation (Adam), a common gradient descent method. While Adam's algorithm is based on gradient descent, it has been tweaked to find a local minimum faster.

4.3 Model evaluation

The performance metric employed determines how well a deep learning model works. In general, the performance of ANNs is measured in accuracy metric.

4.3.1 Accuracy Metric

The model's performance is determined by the model's accuracy, which indicates if the model is being taught correctly. In this dissertation, accuracy metric measures how well the model or algorithm predicts the next position of the user element based on the historical sequences of data.

4.3.2 Performance Metric

The RMSE will be used as performance metric to assess the error distance between prediction position with the ground truth position.

4.3.3 A High-Level Overview of the Implementation Process Flow

This study complies to the CRISP-DM methodology's cycles. The first step is to choose the study subject, followed by an attempt to understand and analyse the datasets, preparing the data before feeding into machine learning model, then building the models, training and assessing them.

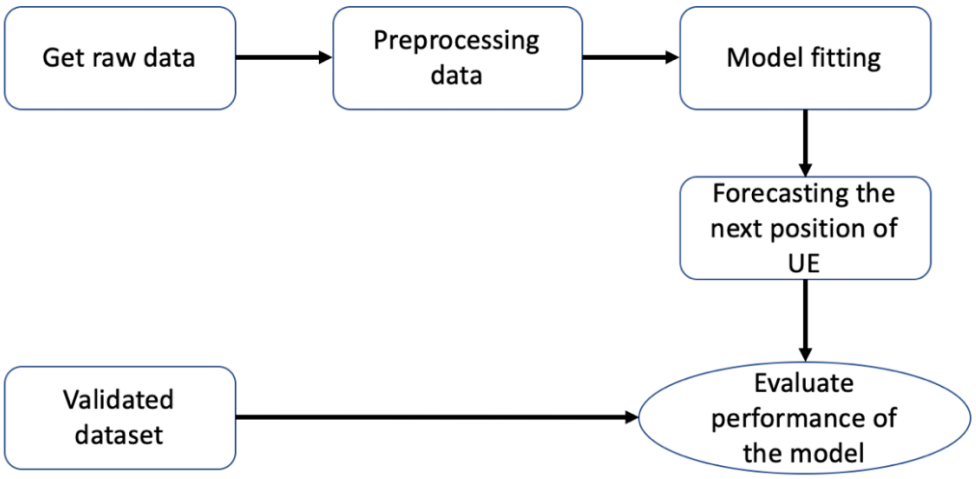


Figure 4.17: Structure of Process Flow

5 Results

This chapter presents all of the findings from the dissertation work. We begin by showing each steps of data preparation results, displaying the hyper parameters that were used in each model, as well as presenting the training and validation outcomes of user mobility trajectory prediction models. Following that, an examination of the overview prediction and distance error by deep learning models are shown. In the following section, we evaluate the performance of deep learning models. At the end of this chapter, we specified our findings to current works

5.1 Experimental Results

5.1.1 Data Exploration and Data Preprocessing

- Data preprocessing for realistic dataset

After parsing and compiling the dataset, the very first step is to visualise it in order to filter the target trajectory out from all trajectories. This is accomplished by providing command to filter user id and display value_counts which help us in knowing how many sequences of data points there are in each user id as shown in Figure 5.2.

```
traj.describe()
```

	Latitude	Longitude	Altitude	Id_user
count	3.387303e+06	3.387303e+06	3.387303e+06	3.387303e+06
mean	3.961094e+01	1.142307e+02	4.284805e+02	9.501306e+01
std	2.397093e+00	1.856440e+01	1.971133e+03	4.728128e+01
min	1.824990e+01	-1.799696e+02	-2.306100e+04	1.000000e+01
25%	3.993521e+01	1.163118e+02	0.000000e+00	6.500000e+01
50%	3.997549e+01	1.163314e+02	1.310000e+02	8.500000e+01
75%	3.999186e+01	1.164181e+02	1.903000e+02	1.280000e+02
max	4.001667e+02	1.799969e+02	4.252950e+04	1.790000e+02

Figure 5.1: Description of realistic dataset

```
In [5]: traj['Id_user'].value_counts()

Out[5]: 68     449860
        128    382820
        167    349425
        85     261023
        10     254722
        62     216681
        65     192035
```

Figure 5.2: Step to filter user id from raw dataset

After observing data point sequences in each user id, we decide to limit the month to April 2008 and show the data points in each day of user id 167 using the command in Figure 5.3. The raw data has now been reduced in size as a consequence of filtering to only include the trajectory of user id 167 in April 2008 as shown in Figure 5.4.

```
In [6]: tr167 = traj.loc[traj.Id_user==167,:]
        tr167['month'] = pd.DatetimeIndex(tr167['Date_Time']).month
        tr167 = tr167.loc[tr167.month==4,:]

        tr167['date'] = pd.DatetimeIndex(tr167['Date_Time']).date
        tr167 = tr167.loc[tr167.month==4,:]
        print(tr167['date'].value_counts())

        format = '%Y-%m-%d %H:%M:%S'

        tr167['Date_Time'] = pd.to_datetime(tr167['Date_Time'], format=format)

        tr167 = tr167.reset_index()
        tr167 = tr167.drop(['index'], axis=1)
        tr167.head()

2008-04-29    3928
2008-04-26    3630
2008-04-27    2356
2008-04-28    1241
2008-04-30    1169
2008-04-25     500
2008-04-04      10
2008-04-05       3
Name: date, dtype: int64
```

Figure 5.3: Step to limit the trajectory of user id 167 in April 2008 from raw dataset

	Latitude	Longitude	Altitude	Date_Time	Id_user	Id_perc	Label	month	date
0	39.981150	116.227850	236.220472	2008-04-04 01:52:55	167	20080404005918.plt	walk	4	2008-04-04
1	39.980967	116.227000	190.288714	2008-04-04 01:54:36	167	20080404005918.plt	walk	4	2008-04-04
2	39.980667	116.226500	196.850394	2008-04-04 02:06:00	167	20080404005918.plt	walk	4	2008-04-04
3	39.979833	116.226467	183.727034	2008-04-04 02:07:56	167	20080404005918.plt	walk	4	2008-04-04
4	39.979700	116.226050	206.692913	2008-04-04 02:16:37	167	20080404005918.plt	walk	4	2008-04-04

Figure 5.4: Data of the trajectory of user id 167 in April 2008

Below is a figure that shows the plots of values of both latitude and longitude for trajectory of user id 167 in April 2008. To correspond with the scenario we set, the geographic coordinates represented by latitude and longitude must be converted into two-dimensional plane by spatial coordinate projection in meter units as x and y coordinates.

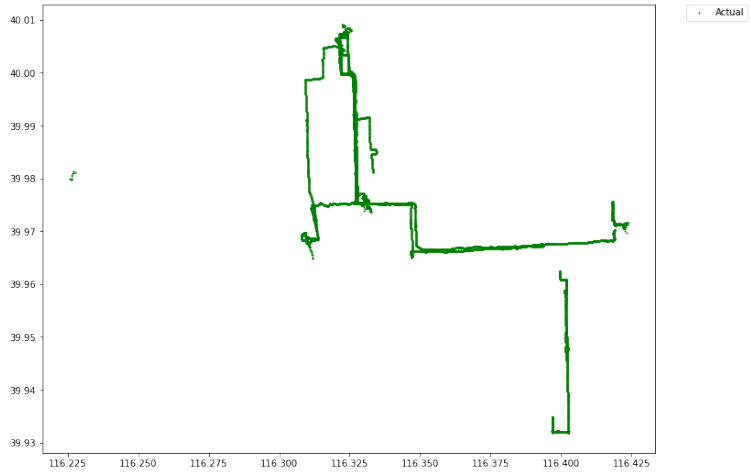


Figure 5.5: Plot of trajectory of user id 167 in April 2008

We adopt the haversine formula to calculate the x and y axis boundaries of latitude and longitude into meter units. Normalization was chosen as the approach for scaling the data into the range of defined distances from 0 to the x and y axis boundaries.

```
In [8]: # calculate haversine distance and based on that calculate calculated speed
        from haversine import haversine, Unit
        s1 = tr167.Latitude.min(),tr167.Longitude.min()
        s2 = tr167.Latitude.max(),tr167.Longitude.min()
        d1 = haversine(s1,s2)*1000
        print(d1)

        s1 = tr167.Latitude.min(),tr167.Longitude.min()
        s2 = tr167.Latitude.min(),tr167.Longitude.max()
        d2 = haversine(s1,s2)*1000
        print(d2)

        8579.590000659398
        16857.044197489
```

```
In [9]: df1 = tr167[['Longitude']]
        values = df1.values
        # ensure all data is float
        values = values.astype('float32')
        # normalize features
        scaler = MinMaxScaler(feature_range=(0, d2))
        df1 = scaler.fit_transform(values)
        df1

Out[9]: array([[ 169.],
               [  96.],
               [  54.],
               ...,
               [8041.],
               [8039.],
               [8037.]], dtype=float32)
```

```
In [10]: df2 = tr167[['Latitude']]
         values = df2.values
         # ensure all data is float
         values = values.astype('float32')
         # normalize features
         scaler = MinMaxScaler(feature_range=(0, d1))
         df2 = scaler.fit_transform(values)
         df2

Out[10]: array([[5472.5],
                [5452. ],
                [5418.5],
                ...,
                [8265. ],
                [8264.5],
                [8262.5]], dtype=float32)
```

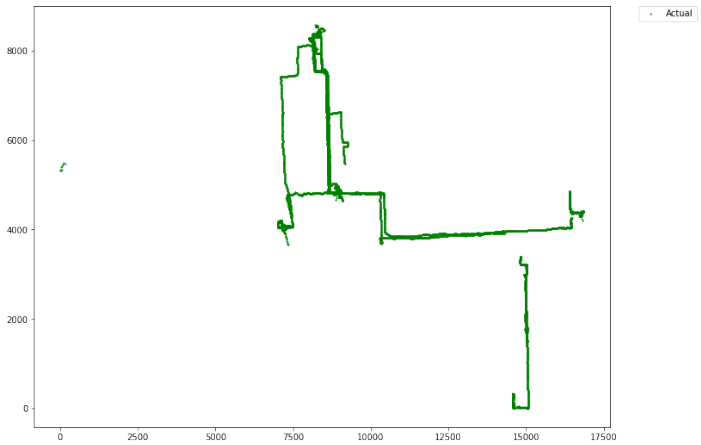
Figure 5.6: Step to convert the latitude and longitude into meter units

```
In [416]: dataset = pd.DataFrame(df1,columns=['X'])
          dataset['Y'] = pd.DataFrame(df2)
          dataset
```

Out[416]:

	X	Y
0	169.0	5472.5
1	96.0	5452.0
2	54.0	5418.5
3	51.0	5325.5

Figure 5.7: Data of both the latitude and longitude in meter units



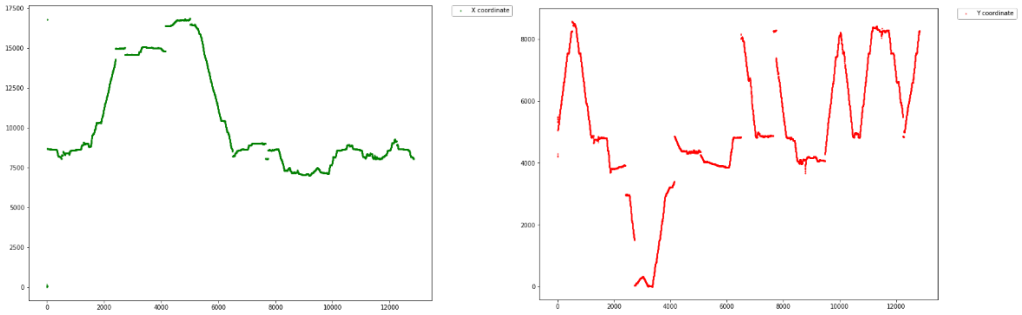


Figure 5.8: Plot of trajectory of user id 167 in April 2008 in meter units, with the upper image displaying the plot of both X and Y coordinates. While the bottom left picture represents X coordinate data points over time, the bottom right image represents Y coordinate data points over time.

Once this was complete the next step was to transform data into supervised problem which corresponds the time series problem and normalize the transformed data into the range of -1 to 1. The rationale behind turning the data into a supervised problem is that data at t-1 time can be used as feature variables or input data given to the machine learning model, while data at t time can be utilized as label or target variables or output data of the machine learning model. The deep learning model will then fit the supervised learning problem corresponding to this type of data structure. The figure below is the transformed and scaled data for column X and Y coordinates.

	Feature Varables		Target Varables	
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	-0.979949	0.275715	-0.988610	0.270937
2	-0.988610	0.270937	-0.993593	0.263127
3	-0.993593	0.263127	-0.993949	0.241448
4	-0.993949	0.241448	-0.998102	0.237951
5	-0.998102	0.237951	-1.000000	0.245760

Figure 5.9: Transformed data corresponding the time series problem

Split the converted data into training and testing datasets so that the deep learning models can use them to train and test the model's performance as shown in figure below.

```
In [419]: # split into train and test sets
          values = reframed.values
          n_train = round(reframed.shape[0]*0.2)
          train = values[:n_train, :]
          test = values[n_train:, :]
          # split into input and outputs
          train_X, train_y = train[:, :-2], train[:, 2:]
          test_X, test_y = test[:, :-2], test[:, 2:]
          # reshape input to be 3D [samples, timesteps, features]
          train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
          test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
          print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

          (2567, 1, 2) (2567, 2) (10269, 1, 2) (10269, 2)
```

Figure 5.10: Step to split the training dataset and testing dataset

The approach for transforming data for multi-step prediction differs somewhat from one-step forward prediction in that the deep learning model will only fit 5 sequences of historical data and produce 3 sequences of future position of one coordinate. This indicates that the X and Y coordinates will be trained in separate models.

X coordinate dataset

	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)	var1(t+1)	var1(t+2)
5	-0.979949	-0.988610	-0.993593	-0.993949	-0.998102	-1.000000	-0.995492	-0.995492
6	-0.988610	-0.993593	-0.993949	-0.998102	-1.000000	-0.995492	-0.995492	-0.990864
7	-0.993593	-0.993949	-0.998102	-1.000000	-0.995492	-0.995492	-0.990864	-0.987186
8	-0.993949	-0.998102	-1.000000	-0.995492	-0.995492	-0.990864	-0.987186	0.991339
9	-0.998102	-1.000000	-0.995492	-0.995492	-0.990864	-0.987186	0.991339	0.991220

Y coordinate dataset

	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)	var1(t+1)	var1(t+2)
5	0.275715	0.270937	0.263127	0.241448	0.237951	0.245760	0.240632	0.257416
6	0.270937	0.263127	0.241448	0.237951	0.245760	0.240632	0.257416	0.270004
7	0.263127	0.241448	0.237951	0.245760	0.240632	0.257416	0.270004	0.277697
8	0.241448	0.237951	0.245760	0.240632	0.257416	0.270004	0.277697	0.002506
9	0.237951	0.245760	0.240632	0.257416	0.270004	0.277697	0.002506	-0.011248

Figure 5.11: Transformed data of X and Y datasets corresponding the time series problem

Split the converted 2 sets of X and Y data into training and testing datasets so that the deep learning models can use them to train and test the model's performance as shown in figure below.

In [158]:
```
# split into train and test sets
values = reframed.values
n_train = round(values.shape[0]*0.2)
train = values[:n_train, :]
test = values[n_train:, :]
# split into input and outputs
train_X, train_y_1 = train[:, :-3], train[:, 5:]
test_X, test_y_1 = test[:, :-3], test[:, 5:]
# reshape input to be 3D [samples, timesteps, features]
train_X_1 = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X_1 = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))

print(train_X_1.shape, train_y_1.shape, test_X_1.shape, test_y_1.shape)
```

```
(2566, 1, 5) (2566, 3) (10264, 1, 5) (10264, 3)
```

In [160]:
```
# split into train and test sets
values = reframed.values
n_train = round(values.shape[0]*0.2)
train = values[:n_train, :]
test = values[n_train:, :]
# split into input and outputs
train_X, train_y_2 = train[:, :-3], train[:, 5:]
test_X, test_y_2 = test[:, :-3], test[:, 5:]
# reshape input to be 3D [samples, timesteps, features]
train_X_2 = train_X_2.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X_2 = test_X_2.reshape((test_X.shape[0], 1, test_X.shape[1]))

print(train_X_2.shape, train_y_2.shape, test_X_2.shape, test_y_2.shape)
```

```
(2566, 1, 5) (2566, 3) (10264, 1, 5) (10264, 3)
```

Figure 5.12: Step to split the training dataset and testing dataset of both X and Y datasets.

- Data preprocessing for simulated dataset

There is no need to transform the data into meter units in the simulated dataset since the data is already in a two-dimensional plane by the spatial coordinate projection of the X and Y coordinates.

```
In [489]: T1.describe(),T2.describe(),T3.describe()

Out[489]: (                 X            Y        Speed         Time
           count  3399.000000  3399.000000  3399.000000  3399.000000
           mean     63.174754   -29.067913     3.224978  1700.000000
           std     432.108042   551.068725     0.915406   981.351109
           min    -812.458685  -878.174702     0.514344     1.000000
           25%    -216.033280  -542.799372     2.636211   850.500000
           50%      55.861715   -47.513036     3.429821  1700.000000
           75%     293.852499   407.526090     3.899109  2549.500000
           max     968.735892   986.371968     4.668675  3399.000000,
                           X            Y        Speed         Time
           count  3901.000000  3901.000000  3901.000000  3901.000000
           mean    105.398738  -109.475135     3.286817  1951.000000
           std     570.692670   496.040060     0.829015  1126.266028
           min    -932.723943  -842.392840     0.658624     1.000000
           25%    -370.000251  -555.747857     2.742040   976.000000
           50%      70.305237  -135.728879     3.409057  1951.000000
           75%     560.328542   260.019988     3.876540  2926.000000
           max    1149.856802   895.302775     4.720457  3901.000000,
                           X            Y        Speed         Time
           count  3783.000000  3783.000000  3783.000000  3783.000000
           mean    -50.506524    31.696422     3.220998  1892.000000
           std     515.573246   550.668160     0.962932  1092.202362
           min    -938.832233  -867.344851     0.113298     1.000000
           25%    -504.538861  -444.334434     2.532394   946.500000
           50%      42.592752   -44.397591     3.478637  1892.000000
           75%     338.364895   517.842173     3.994498  2837.500000
           max     933.444475  1023.906636     5.069813  3783.000000)
```

Figure 5.13: Description of simulated dataset

Below is a figure that shows the plots of values of both X and Y coordinates for 3 trajectories.

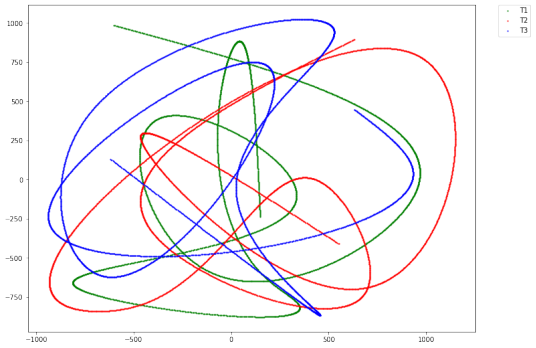


Figure 5.14: Description of Dataset.

In order to challenge the deep learning models for simulated data case, we transformed three trajectories into supervised problem and then employed concatenate function to compile three trajectories into one at the end of process.

	Feature Varables		Target Varables	
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	0.079378	-0.315985	0.079189	-0.311143
2	0.079189	-0.311143	0.079000	-0.306302
3	0.079000	-0.306302	0.078812	-0.301460
4	0.078812	-0.301460	0.078623	-0.296619
5	0.078623	-0.296619	0.078435	-0.291778
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	0.426156	-0.503541	0.422621	-0.500140
2	0.422621	-0.500140	0.419085	-0.496740
3	0.419085	-0.496740	0.415550	-0.493339
4	0.415550	-0.493339	0.412015	-0.489939
5	0.412015	-0.489939	0.408480	-0.486539
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	0.677580	0.390514	0.680372	0.387934
2	0.680372	0.387934	0.683164	0.385354
3	0.683164	0.385354	0.685956	0.382775
4	0.685956	0.382775	0.688746	0.380196
5	0.688746	0.380196	0.691536	0.377616

```
frames = [reframed, reframed_2, reframed_3]
result = pd.concat(frames)
reframed = pd.DataFrame(result)

reframed = reframed.reset_index()
reframed = reframed.drop(['index'], axis=1)
reframed.tail()
```

	var1(t-1)	var2(t-1)	var1(t)	var2(t)
11075	-0.638409	0.036072	-0.642337	0.039762
11076	-0.642337	0.039762	-0.646265	0.043451
11077	-0.646265	0.043451	-0.650194	0.047141
11078	-0.650194	0.047141	-0.654123	0.050831
11079	-0.654123	0.050831	-0.658051	0.054521

Figure 5.15: Step to transform simulated data into supervised problem and compile three trajectories into one.

The last step is to split the converted data into training and testing datasets as shown in figure below.

```
# split into train and test sets
values = reframed.values
n_train = round(reframed.shape[0]*0.2)
train = values[:n_train, :]
test = values[n_train:, :]
# split into input and outputs
train_X, train_y = train[:, :-2], train[:, 2:]
test_X, test_y = test[:, :-2], test[:, 2:]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(2216, 1, 2) (2216, 2) (8864, 1, 2) (8864, 2)
```

Figure 5.16: Step to split the training dataset and testing dataset of both X and Y datasets.

5.2 Model Training

In this section, results are shown for the two cases where predictive models were constructed and trained to yield either one or multi steps trajectories. These two scenarios are further subdivided into two distinct cases, where either a realistic dataset or simulated data is employed. Experiments with deep learning models that generate multi steps forecasting of future trajectories are also presented.

5.2.1 Single-User for Single-step and Multi-step Forecasting and Results

Table 4.1 displays all of the hyperparameters that have already been configured in each deep learning model. Figure 5.17a and Figure 5.17b show the training results for the various models, where a dataset with two coordinates of realistic dataset were used in Figure 5.17a and a simulated dataset was utilised in Figure 5.17b. While, Figures 5.18a and 5.18b demonstrate the training outcomes for the various models where a dataset with two coordinates in realistic dataset was fed in deep learning models in order to predict multi-steps of user element.

Table 5.1: Parameters used for training and producing results.

Parameters	LSTM	Stacked LSTM	Bidirectional LSTM	GRU	Stacked GRU	RNN
Epochs	600	600	600	600	600	600
Verbose	2	2	2	2	2	2
shuffle	False	False	False	False	False	False
Neural network Units	64	64, 128	150	64	32,64	64
Activation	relu	tanh	tanh	tanh	tanh	tanh
Dropout layer	0.1	0.1	0.1	0.1	0.1	0.1
BATCH_SIZE	72	72	72	72	72	72
Loss	mae	mae	mae	mae	mae	mae
Optimizer	adam	adam	adam	adam	adam	adam
Metric	accuracy	accuracy	accuracy	accuracy	accuracy	accuracy
Callback monitor	val_loss	val_loss	val_loss	val_loss	val_loss	val_loss
Callback mode	Auto	Auto	Auto	Auto	Auto	Auto
Callback patience	10	10	10	10	10	10
Restore_best_weights	True	True	True	True	True	True

Many variants of hyper parameters were explored during the step to tune deep learning models, and the ones that resulted in the lowest validation error are shown in Table 5.1. The batch size in each iteration inside an epoch; the number of hidden layers inside the models; and the number of hidden neurons in each layer were a result of tuning deep models. In this scenario, some parameter such as the learning rates were left at their default. The shape of the input layer is a reflection of the shape of the train dataset, and the shape of the output layer is a reflection of the shape of the test dataset as well. That is, when the model is fed, two outputs of one-time step or multi-time step X and Y coordinates are created to reflect the shape of the output layer. The Loss function is set to mean average error, the optimizer is Adam where Adam resulted in the lowest validation errors, and the accuracy metric is accuracy. The 600 epochs were specified for each deep learning model; however, the early stopping function will check on the monitor value of validate loss when the machine stops learning, and the training process will be halted. Because the dataset we provided is a sequence, it is not a good idea to shuffle it, so the batch size was set to 72 and shuffle was turned off.

The training diagram in Figure 5.17a shows that the validation error (MAE) in every model for the case of realistic dataset declines with the training error until about epoch 10, when the loss equals 0.05. Following that, the train error saturates, but the validation error continues to decrease until it achieves a minimum. Overfitting will result from training the Stacked LSTMs model after 30 epochs since the model does not improve and keep increasing value of loss. This result suggests that 30 epochs were sufficient to train the Stacked LSTMs model. To possibly enhance the model's accuracy, more training, different hyperparameters, or a new model architecture would be necessary.

Loss for the deep learning models fed realistic dataset in predicting one step forward

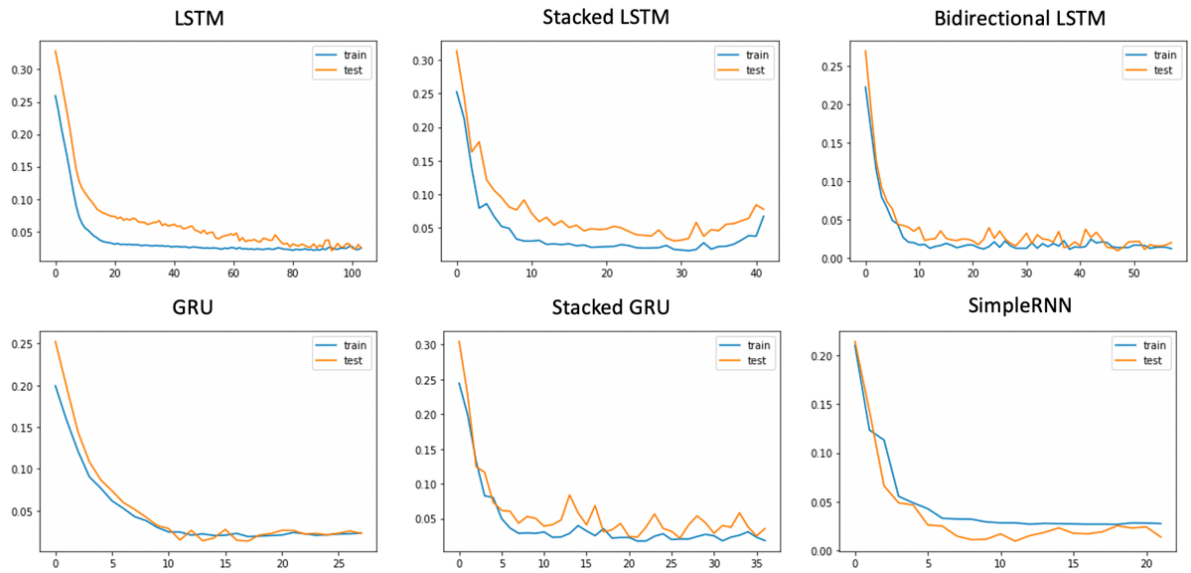


Figure 5.17a: Training (blue) and validation (orange) for forecasting the next position of a realistic dataset for various models

Loss for the deep learning models fed simulated dataset in predicting one step forward

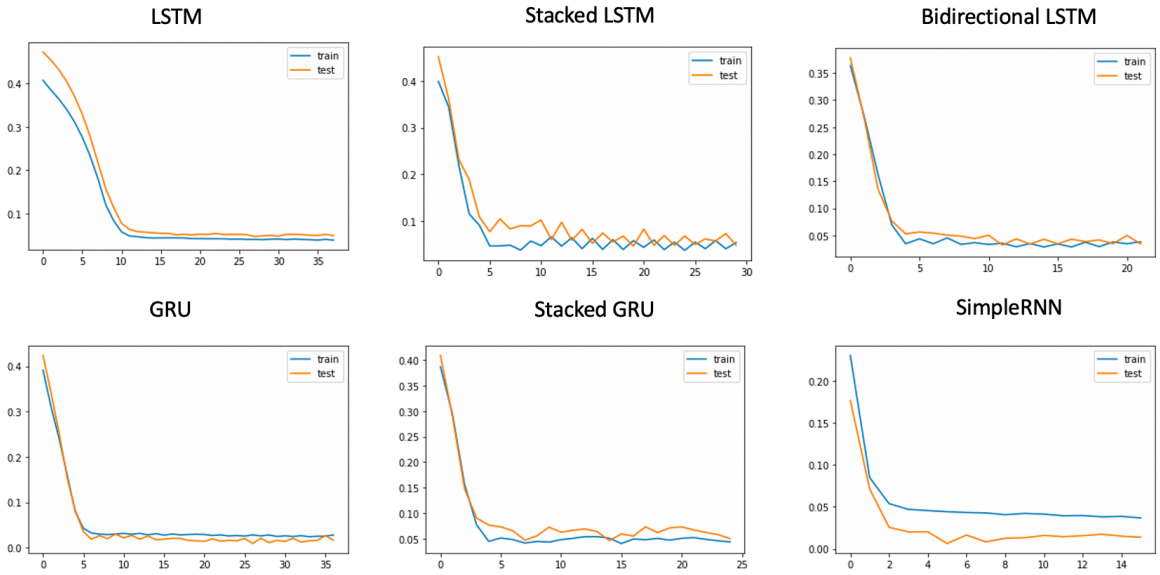


Figure 5.17b: Training (blue) and validation (orange) for forecasting the next position of a simulated dataset for various models

The training diagram in Figure 5.17b shows that the validation error in every model for the case of simulated dataset declines with the training error until about epoch 5, when the loss equals 0.05. Unlike the train error and the validation error for realistic dataset, this time those train and test error saturate at around epoch 5.

The training diagram in Figure 5.18a shows that the validation error in every model for the case of X coordinate in realistic dataset declines with the training error until about epoch 5, when the loss equals 0.03 except the Stacked LSTMs which reduces when the loss equals 0.075.

It can be seen from the training diagram in Figure 5.18b that, for Y coordinate in realistic dataset, validation error decreases with training error until around epoch 2, when it equals 0.05.

Loss for the deep learning models fed realistic X coordinate in realistic dataset for predicting multi steps forward

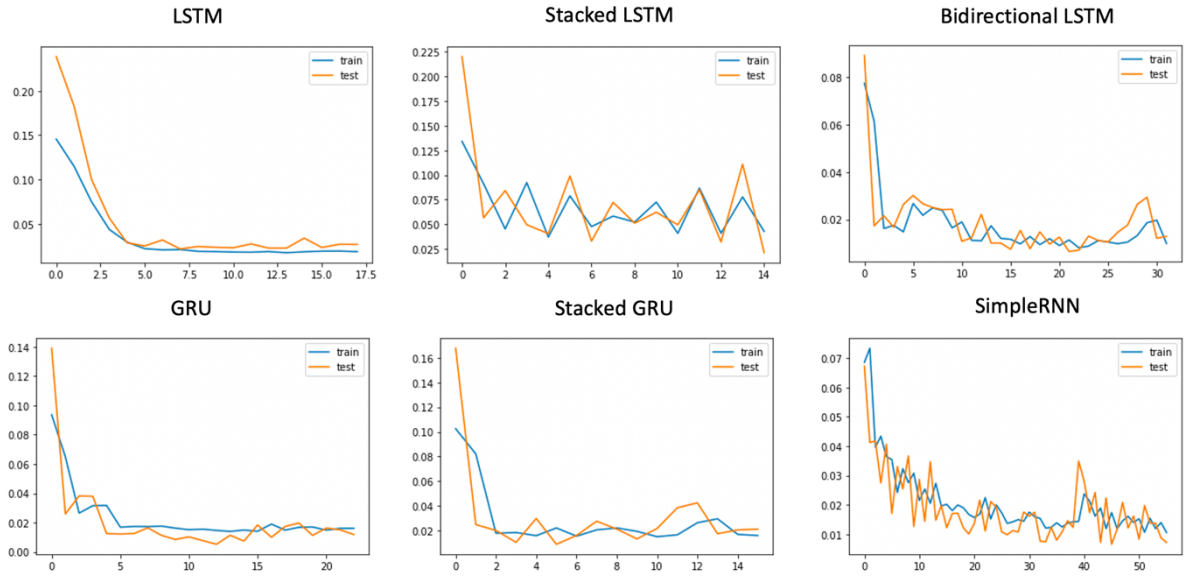


Figure 5.18a: Training (blue) and validation (orange) for forecasting the multi steps of X coordinate in a realistic dataset for various models

Loss for the deep learning models fed realistic Y coordinate in realistic dataset for predicting multi steps forward

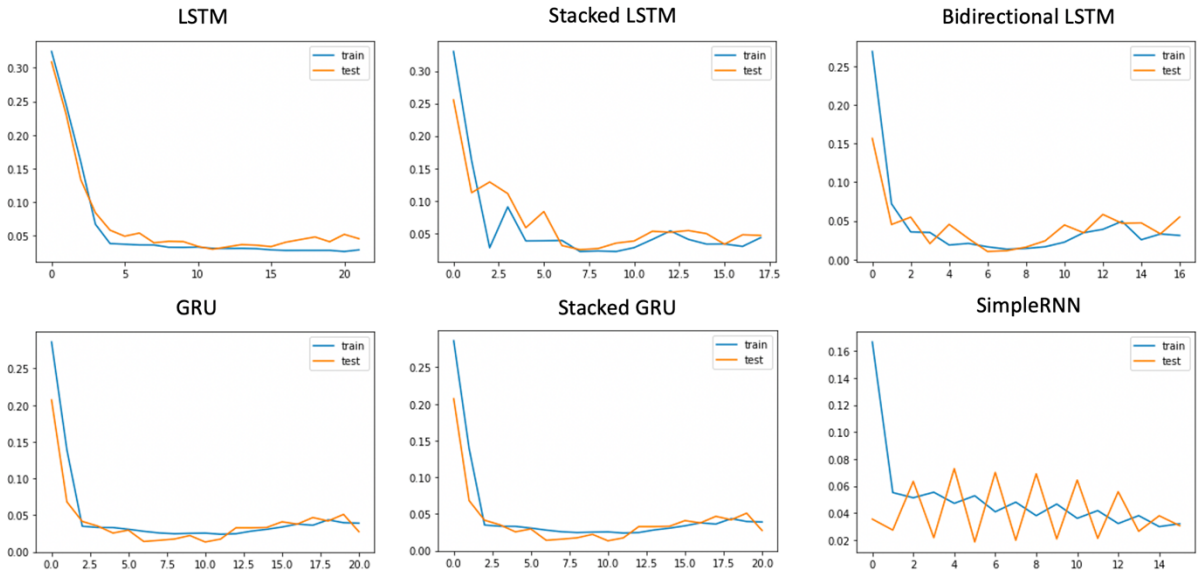


Figure 5.18b: Training (blue) and validation (orange) for forecasting the multi steps of Y coordinate in a realistic dataset for various models

5.3 Performance summary

Finally, we can observe that the accuracy metrics utilized for verifying the one step forward prediction for each model in Table 5.2 almost reach a flawless score. There is only a minor variation in accuracy of both realistic dataset and simulated dataset. The RSME error for each model performed on both datasets in Table 4.3, for models fed realistic dataset produces the average RSME of all models at 226.2341445m. While, models fed simulated dataset produces the average RSME of all models at 41.908754m. The best models for producing the lowest RMSE are RNN, GRU, and Bi-directional LSTM, in that order.

TABLE 5.2: Accuracy score of One Step Forward Prediction for Different Methods. (Unit: percent)

Algorithms	**Accuracy score for realistic dataset**	**Accuracy score for simulated dataset**
LSTM	99.5715 $\pm$ 0.1	98.7590 $\pm$ 0.1
Stacked LSTMs	99.8150 $\pm$ 0.1	97.2022 $\pm$ 0.1
Bi-directional LSTM	99.9026 $\pm$ 0.2	97.6873 $\pm$ 0.2
GRU	99.8345 $\pm$ 0.2	99.5826 $\pm$ 0.2
Stacked GRUs	99.6786 $\pm$ 0.1	96.7509 $\pm$ 0.1
RNN	99.8637 $\pm$ 0.1	99.6954 $\pm$ 0.1

TABLE 5.3: RMSE of One Step Forward Prediction for Different Methods. (Unit: m)

Algorithms	RMSE for realistic dataset (m)	RMSE for simulated dataset (m)
LSTM	232.770676	75.516244
Stacked LSTMs	356.818133	58.805422
Bi-directional LSTM	152.560618	42.532989
GRU	130.443684	11.709877
Stacked GRUs	388.537723	55.370970
RNN	96.274033	7.517022

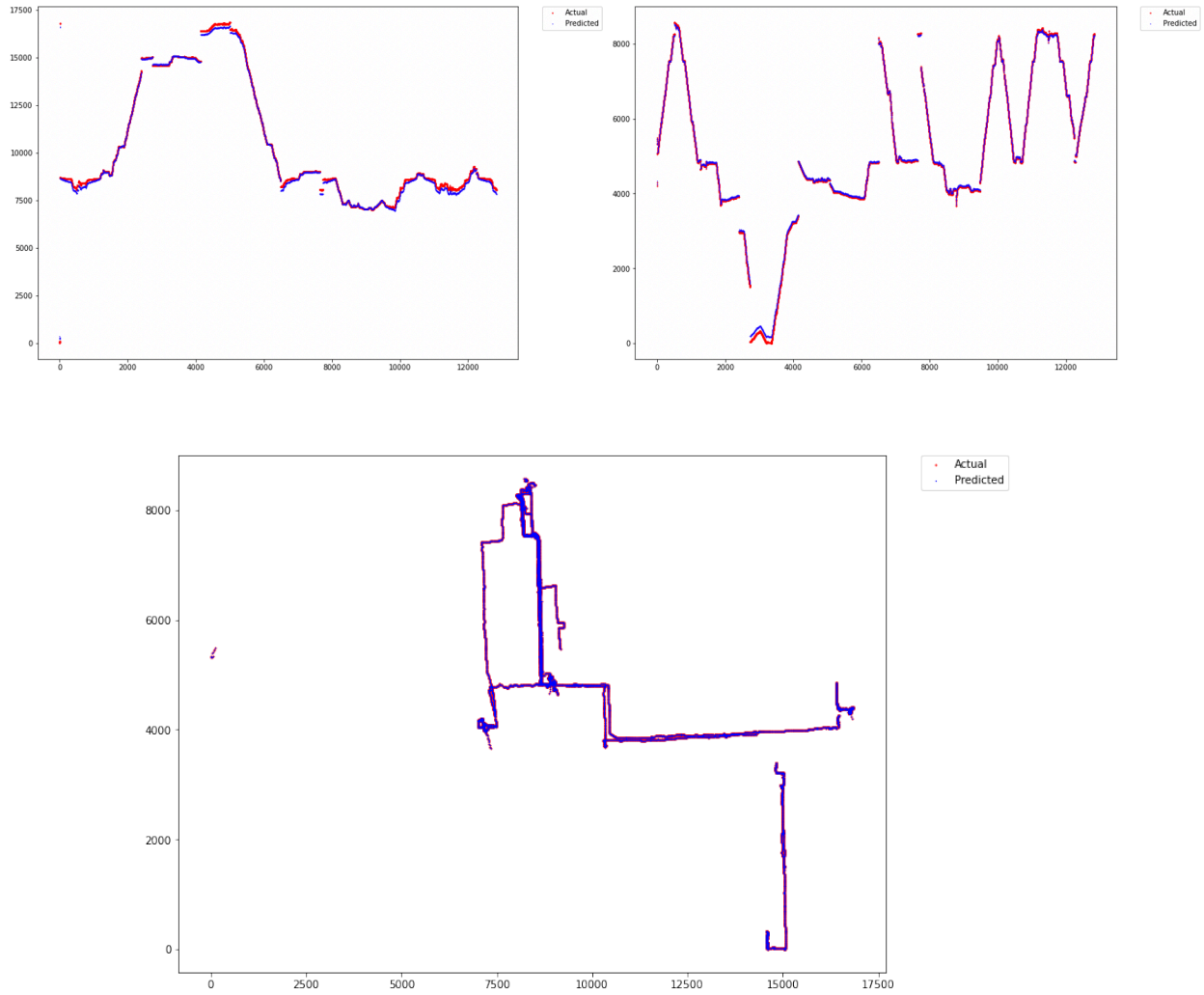


Figure 5.19: Plot of actual positions (red) and predicted position (dark blue) for realistic dataset where the top left image represents the plot of sequences of X coordinate over time and the top right image represents the plot of sequences of Y coordinate over time. The bottom image displays the plot of entire path and shows the comparison between correct path and predicted path.

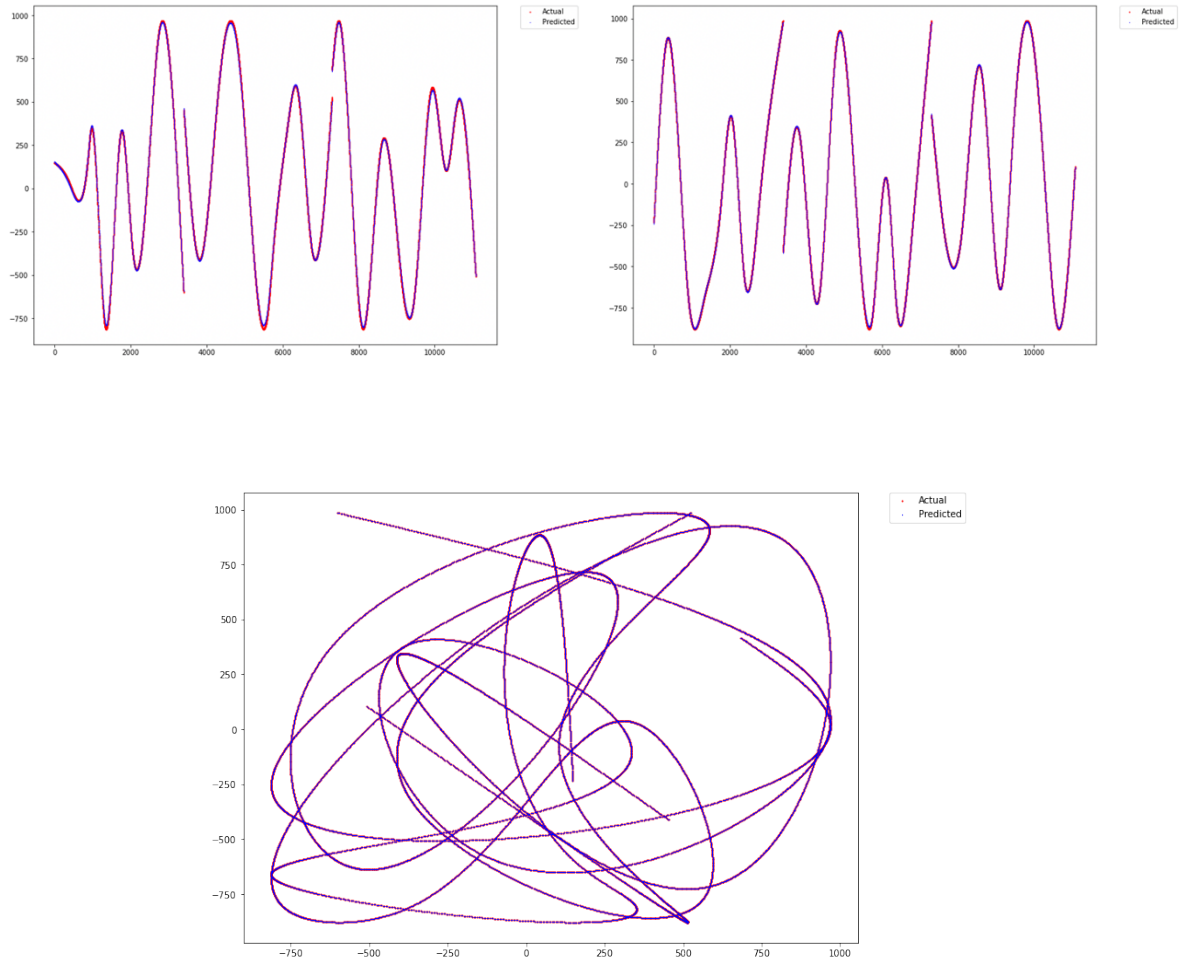


Figure 5.20: Plot of actual positions (red) and predicted position (dark blue) for simulated dataset where the top left image represents the plot of sequences of X coordinate over time and the top right image represents the plot of sequences of Y coordinate over time. The bottom image displays the plot of entire path and shows the comparison between correct path and predicted path.

Again, we can observe that the accuracy metrics utilized for verifying the multi steps forward prediction for each model in Table 5.4 that only achieve the score around 40 percent. There is only a minor variation in accuracy of both models fed X coordinate dataset and models fed Y coordinate dataset. The RSME error for each model performed on both datasets in Table 5.5 and 5.6, for models fed those dataset produces the average RSME of all models fed X coordinate dataset and all models fed Y coordinate for 1 step forward at 173.594035333m and 206.131096667m respectively. The average RSME of all models for 2 step forwards is at 166.4631335m for all models fed X coordinate dataset and 194.125899833m for all models fed Y coordinate dataset. While, the average RSME of all models for 3 step forwards is at 177.2499295m for all models fed X coordinate dataset and 214.873617m for all models fed Y coordinate dataset. The best models for producing the lowest RMSE are GRU, RNN and Bi-directional LSTM, in that order.

TABLE 5.4: Accuracy score of Multi Steps Prediction Model for Different Methods. (Unit: percent)

Algorithms	Accuracy score for model fed X dataset	Accuracy score for model fed Y dataset
LSTM	38.6984 ± 0.2	36.5355 ± 0.2
Stacked LSTMs	26.7050 ± 0.2	36.0288 ± 0.2
Bi-directional LSTM	35.5709 ± 0.2	42.1278 ± 0.2
GRU	46.2685 ± 0.5	43.4821 ± 0.5
Stacked GRUs	48.6847 ± 0.5	49.8441 ± 0.5
RNN	28.3223 ± 0.1	37.0323 ± 0.1

TABLE 5.5: RMSE of Each X Coordinate Prediction Step for Different Methods. (Unit: m)

Algorithms	1 step forward	2 step forward	3 step forward
LSTM	274.634587	309.215274	340.188057
Stacked LSTMs	391.678400	395.275280	408.288176
Bi-directional LSTM	114.050055	74.311199	78.116715
GRU	86.903621	62.504715	54.231120
Stacked GRUs	141.342496	90.865881	78.595434
RNN	32.955053	66.606452	104.080075

TABLE 5.6: RMSE of Each Y Coordinate Prediction Step for Different Methods. (Unit: m)

Algorithms	1 step forward	2 step forward	3 step forward
LSTM	227.309398	158.781124	177.823160
Stacked LSTMs	169.819913	180.382488	183.307833
Bi-directional LSTM	109.690042	101.235310	120.566983
GRU	87.622108	109.648146	176.883254
Stacked GRUs	519.188730	465.372902	471.381613
RNN	123.156389	149.335429	159.278862

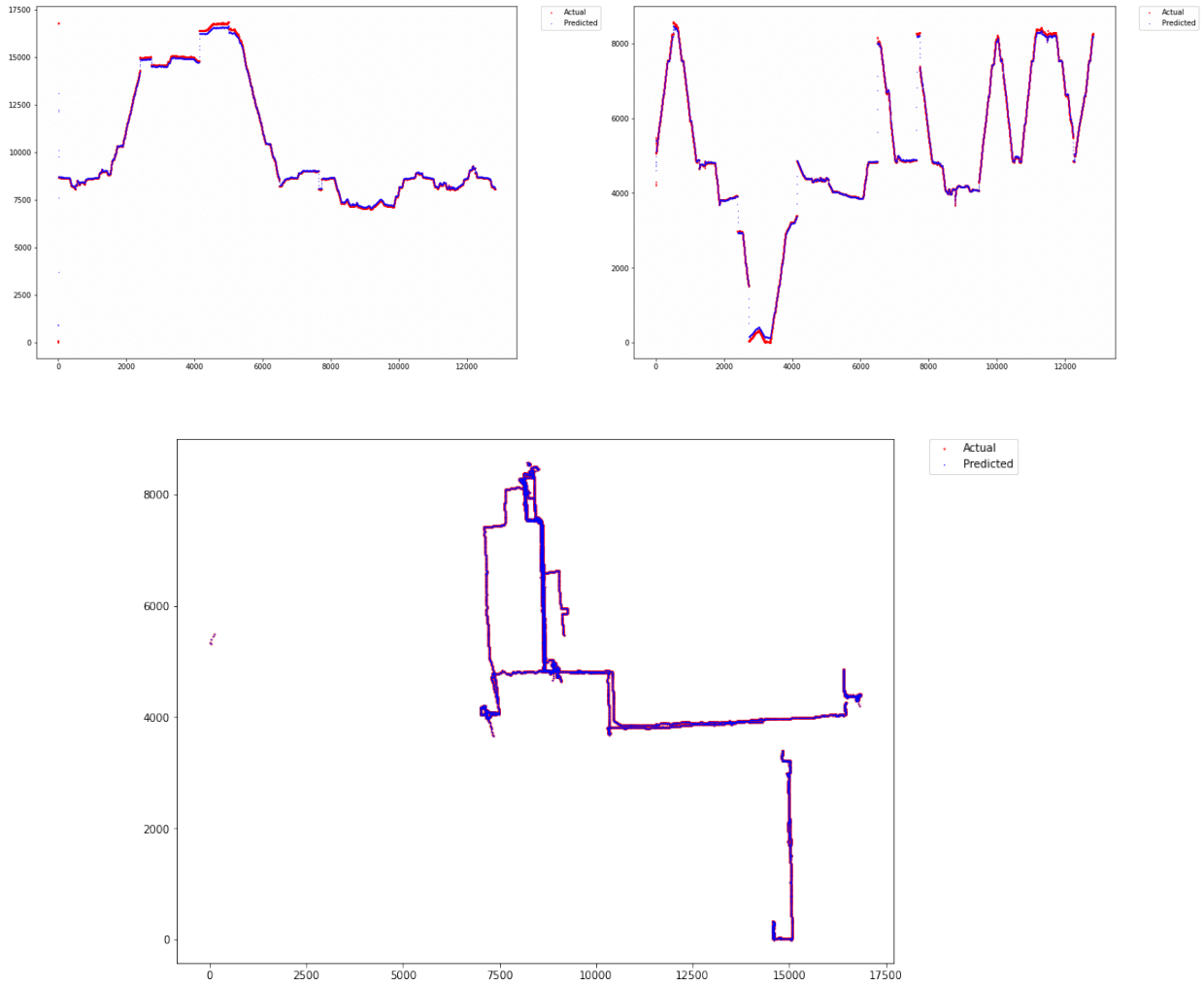


Figure 5.21: Sample plot of actual positions (red) and predicted position (dark blue) for realistic dataset with three steps forward where the top left image represents the plot of sequences of X coordinate over time and the top right image represents the plot of sequences of Y coordinate over time. The bottom image displays

6 Evaluation

A success or failure assessment is made in this section. A critical analysis of the results of numerous experiments done in Chapter 4 will be performed, as well as an assessment of overall success and how each experiment contributed to the accomplishment of objectives, in order to fulfil this goal Primary objectives and limitations are discussed.

A table outlining below is a list of requirements and status whether they were accomplished in accordance with the current service deployed.

Primary objectives

#	Success/Partially success/Fail	Objectives
i.	**Success**	To adapt the theoretical techniques of machine learning models with application to forecast the next position of user element in the future.
ii.	**Success**	To optimize, tune and train the model in order to forecast the next position of user element as accurately as possible.
iii.	**Success**	To evaluate and compare the performance of deep learning models in forecasting the possible position of user element in the future.
iv.	**Success**	To develop the machine learning model to forecast multi-step sequences of position utilizing numerous sets of historical sequences of the position as input.

Figure 6.1: Primary objectives Achievement Table

Limitations/Problems

#	Success/Partially Success/Fail	Description
i.	**Partially Success**	The method and technique used to deal with the Forecasting application are dependent on the user's expertise and can provide varying outcomes.
ii.	**Partially Success**	Because existing deep learning models cannot self-adjust their hyperparameters and neural network units, iteration tuning models may be required.
iii.	**Fail**	The present models can only forecast routes that have already been learned in deep learning models. It's incompatible with forecasting a new path that the models have never seen before.

Figure 6.2: Limitations/Problems Achievement Table

6.1 Primary Objectives

Beginning with the fulfilment of primary objectives, these were important basis as they laid the foundation for the project. To be able to apply the theoretical approaches of machine learning models with application to forecast the next position of UE in the future is critical. If the first requirement has not been achieved in any way, the experiment could not have continued since there would have been no way to utilize the practical usage of deep learning algorithms with time series problem. Hence, literature review was conducted in order to acquire ideas and strategies for adjusting machine learning with application of forecasting for time series problem and the solution chosen for this is to transform the data sequences into a supervised learning

problem corresponding time series problem in which the historical data sequences are feature variables and the present and the future data sequences are label or target variables.

The second major objective was accomplished as the deep learning models such as LSTM, stacked LSTMs, GRU, stacked GRUs and RNN were successfully in optimizing, tuning and training the models in order to be able to forecast the next position of user element. This was done based on research from a literature study and other relevant resources from many websites that involves with techniques in applying machine learning models with application for prediction. Once we had compiled all of the techniques that we had gathered from various sites for optimizing, tuning, and training deep learning models. The deep learning algorithms will then be fitted with historical data sequences as input data and current and future data sequences as output data.

The third fundamental objective was likewise accomplished since the metrics for evaluating and comparing the performance of the deep learning models were implemented in this dissertation. The accuracy and mean absolute error metrics, as well as the root square mean error metric, were employed as the evaluation technique and these measures were crucial in determining whether the algorithms were excellent in predicting position of user element in the future or not.

Lastly, the challenging feature of forecasting application from all was also achieved. This feature requires a thorough grasp of both data transformation and deep learning models along with time series theory in order to utilise the machine learning model to forecast multi-step sequences of position using several sets of previous sequences of position as input and produce multiple steps of future sequences of position. Then, once again, we had assembled all of the strategies for optimizing, tuning, and training deep learning models that we had gained experience while testing deep learning models for one step forward prediction. We then applied those strategies in this challenging application and the deep learning algorithms were trained using 5 timesteps back of historical data sequences as input and 3 timesteps forward of data sequences as output.

6.2 Limitation of this work

The following limitations are identified for this work:

1. The method and technique used to deal with the Forecasting application are dependent on the user's expertise and can provide varying outcomes. The reason for the first constraint only being partially satisfied in attempts to discover a solution is because the prediction application depended on the problem definition and the user's knowledge in a limited amount of time of study. Different researchers and developers may have distinct talents and different potential in defining scope of project and adopting the use of theories in forecasting application. Despite significant research and testing, the

robustness of any process is dependent on the critical knowledge of the individual and their own capacity to discover the most matched models for forecasting application. The limitation would be satisfied if there was more time, more testing and refining of the techniques. The reason for the second problem only being partially achieved is that

2. Because existing deep learning models cannot self-adjust their hyperparameters and neural network units, iteration tuning models may be required. These hyperparameters act as knobs which can be tweaked during the training of the model and they must be optimized for the deep learning model to produce optimal results. An iteration tuning and training models may be necessary when the machine learning predicts a new path of user element that it has never learned before. As a result, deep learning models are only able to forecast in offline mode. To solve this problem, the adaptation of hyperparameters online as the network trains is need. Self-tuning networks (STNs) will be the next level of studying, which fit a hypernetwork to approximate the optimum response function in the region of the present hyperparameters. Another approach would be to refine the algorithms, for example, by utilizing the practical application of online learning, such as reinforcement learning, which is a robust algorithm that learns how to find the best possible path in an uncertain by employing trial and error to solve the problem and obtain the rewards or penalties that follow the design policy.

3. The present models can only forecast routes that have already been learned in deep learning models. It's incompatible with forecasting a new path that the models have never seen before. In accordance with the limitation presented on the second limitation, the existing deep learning models cannot self-adjust their hyperparameters, neural network units and self-trained to fit the deep learning model with new path of user element. In the case that the new path was fed into the deep learning model as input in the condition that the machine learning never learn this path before, it is possible that the prediction made by the model may be incorrect. To overcome this difficulty, the solution provided in the second limitation also applies to this instance.

7 Conclusion and Future Work

In this dissertation, we study the usage of forecasting trajectory for single user and examine the possible techniques to deal with one step forward and multi steps forward forecasting. We present the implementation of model framework and propose the robust deep learning models such as LSTM, Stacked LSTMs, Bidirectional LSTM, GRU, Stacked GRUs and RNN that are identified as suitable algorithms. The chosen models are trained with both realistic dataset and simulated dataset to forecast the next position of the user element based on the historical position.

Experiments on a realistic dataset and simulated data show that the bidirectional LSTM and GRU and RNN are the best approach in term of accuracy and fast training, which demonstrate their outstanding generalization ability for one step prediction as well as robustness and stability for multi-step prediction.

The future work for this dissertation could be doing more research in order to improve and refine the practical usage of deep learning model with prediction applications as well as making use of practical algorithms with prediction applications that enable the model to be trained in online mode such as reinforcement learning which is one practical algorithm in artificial intelligence and employing the adaptation of hyperparameters online, for instance, self-tuning networks.

References

[1] Bahra, N.; Pierre, S. A Hybrid User Mobility Prediction Approach for Handover Management in Mobile Networks. Telecom 2021, 2, 199–212. https://doi.org/10.3390/ telecom2020013

[2] C. Yao, J. Guo and C. Yang, "Achieving high throughput with predictive resource allocation", Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP), pp. 768-772, Dec. 2016.

[3] Z. Kai, S. Tarkoma, S. Liu and H. Vo, "Urban human mobility data mining: An overview", Proc. IEEE Int. Conf. Big Data, pp. 1911-1920, Dec. 2017.

[4] B. D. Ziebart, A. L. Maas, A. K. Dey and J. A. Bagnell, "Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior", Proc. 10th Int. Conf. Ubiquitous Comput., pp. 322-331, Sep. 2008.

[5.] S. Qiao, D. Shen, X. Wang, N. Han and W. Zhu, "A self-adaptive parameter selection trajectory prediction approach via hidden Markov models", IEEE Trans. Intell. Transp. Syst., vol. 16, pp. 284-296, Feb. 2015.

[6] R. Meena and V. T. Bai, "Study on Machine learning based Social Media and Sentiment analysis for medical data applications," 2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2019, pp. 603-607, doi: 10.1109/I-SMAC47947.2019.9032580.

[7] C. Wang, L. Ma, R. Li, T. S. Durrani and H. Zhang, "Exploring Trajectory Prediction Through Machine Learning Methods," in IEEE Access, vol. 7, pp. 101441-101452, 2019, doi: 10.1109/ACCESS.2019.2929430.

[8] Z. HAN and J. WANG, "Speech Emotion Recognition Based on Deep Learning and Kernel Nonlinear PSVM," 2019 Chinese Control And Decision Conference (CCDC), 2019, pp. 1426-1430, doi: 10.1109/CCDC.2019.8832414.

[9] K. Pahwa and N. Agarwal, "Stock Market Analysis using Supervised Machine Learning," 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), 2019, pp. 197-200, doi: 10.1109/COMITCon.2019.8862225.

[10] P. Zhang, Y. Su and C. Wang, "Statistical Machine Learning Used in Integrated Anti-Spam System," 2007 International Conference on Machine Learning and Cybernetics, 2007, pp. 4055-4058, doi: 10.1109/ICMLC.2007.4370855.

[11] “Statistical learning theory,”. Accessed on: Aug. 10, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Statistical_learning_theory

[12] O. Bousquet, S. Boucheron, G. Lugosi, Introduction to statistical learn-ing theory, in: Summer School on Machine Learning, Springer, 2003, pp. 169–207.

[13] “Supervised and Unsupervised Machine Learning Algorithms,”. Accessed on: Aug. 10, 2021. [Online]. Available: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[14] “Commonly used Machine Learning Algorithms (with Python and R Codes),”. Accessed on: Aug. 10, 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

[15] “What is reinforcement learning? The complete guide,”. Accessed on: Aug. 10, 2021. [Online]. Available: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

[16] Council of Europe. “History of Artificial Intelligence,”. Accessed on: Aug. 10, 2021. [Online]. Available: https://www.coe.int/en/web/artificial-intelligence/history-of-ai

[17] A. M. TURING, I.—COMPUTING MACHINERY AND INTELLIGENCE, Mind, Volume LIX, Issue 236, October 1950, Pages 433–460, https://doi.org/10.1093/mind/LIX.236.433

[18] IBM Cloud Education. “What is Deep Learning?,” May 1, 2020. Accessed on: Aug. 10, 2021. [Online]. Available: https://www.ibm.com/cloud/learn/deep-learning

[19] Muhammad Imran. “Advantages of Neural Networks – Benefits of AI and Deep Learning,” Mar. 9, 2020. Accessed on: Aug. 10, 2021. [Online]. Available: https://www.folio3.ai/blog/advantages-of-neural-networks/

[20] Jürgen Schmidhuber. Deep learning in neural networks: An overview. Neural networks, 61:85–117, 2015.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. nature, 521(7553):436, 2015.

[22] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In Computer vision and pattern recognition (CVPR), 2012 IEEE conference on, pages 3642–3649. IEEE, 2012.

[23] SMEBOOK. “The advantages and disadvantages of deep learning,” Feb. 27, 2021. Accessed on: Aug. 10, 2021. [Online]. https://smebook.eu/knowledge-base/deep-learning/the-advantages-and-disadvantages-of-deep-learning/

[24] Recurrent Neural Network. Brilliant.org. Retrieved 11:54, September 4, 2021. Accessed on: Aug. 10, 2021. [Online]. Available: https://brilliant.org/wiki/recurrent-neural-network/

[25] M. Štěpnička, J. Peralta, P. Cortez, L. Vavříčková, G. Gutierrez, “Forecasting seasonal time series with computational intelligence: contribution of a combination of distinct methods,” in Proceedings of the 7th conference of the European Society for Fuzzy Logic and Technology (EUSFLAT-11), July, 2011. Place of publication: Atlantis Press, 2011.

[26] Chenyang Xu and Changqing Xu. “Predicting Personal Transitional Location Based on Modified-SVM”. In: Dec. 2017, pp. 340–344. doi: 10.1109/ CSCI.2017.57.

[27] I. Cohen, “Time Series-Introduction,” Jun. 5, 2019. Accessed on: Aug. 10, 2021. [Online]. https://towardsdatascience.com/time-series-introduction-7484bc25739a

[28] “Stationarity,” Accessed on: Aug. 10, 2021. [Online]. https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm

[29] Luca Pappalardo, Filippo Simini, Gianni Barlacchi, Roberto Pellungrini, “Scikit-mobility: a Python library for the analysis, generation and risk assessment of mobility data”, the Journal of Statistical Software, August. 2019.

[30] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu and J. G. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols", Proc. ACM MobiCom, vol. 114, pp. 119, Oct. 1998.

[31] J. Ariyakhajorn, P. Wannawilai and C. Sathitwiriyawong, "A comparative study of random waypoint and Gauss-Markov mobility models in the performance evaluation of MANET", Proc. Int. Symp. Commun. Inf. Technol., pp. 894-899, Oct. 2006.

[32] K. Lee, S. Hong, S. J. Kim, I. Rhee and S. Chong, "SLAW: Self-similar least-action human walk", IEEE/ACM Trans. Netw., vol. 20, no. 2, pp. 515-529, Apr. 2012.

[33] B. D. Ziebart, A. L. Maas, A. K. Dey and J. A. Bagnell, "Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior", Proc. 10th Int. Conf. Ubiquitous Comput., pp. 322-331, Sep. 2008.

[34] M. Morzy, "Mining frequent trajectories of moving objects for location prediction" in Machine Learning and Data Mining in Pattern Recognition, Berlin, Germany: Springer, pp. 667-680, 2007.

[35] S. Qiao, D. Shen, X. Wang, N. Han and W. Zhu, "A self-adaptive parameter selection trajectory prediction approach via hidden Markov models", IEEE Trans. Intell. Transp. Syst., vol. 16, pp. 284-296, Feb. 2015.

[36] Q. J. Lv, Y. Qiao, N. Ansari, J. Liu and J. Yang, "Big data driven hidden Markov model based individual mobility prediction at points of Interest", IEEE Trans. Veh. Technol., vol. 66, no. 6, pp. 5204-5216, Jun. 2017.

[37] A. B. R. Johansson, “Vehicle trajectory prediction using recurrent LSTM neural networks,” MSc Thesis, Dept. of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden, 2021. [Online]. Available: https://odr.chalmers.se/bitstream/20.500.12380/300726/1/B%C3%BCkk_Johansson_2020.pdf

[38] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces", Proc. CVPR, pp. 961-971, Jun. 2016.

[39] Y. Liu, X. Wang, G. Boudreau, A. B. Sediq and H. Abou-zeid, "Deep Learning Based Hotspot Prediction and Beam Management for Adaptive Virtual Small Cell in 5G Networks," in IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 4, no. 1, pp. 83-94, Feb. 2020, doi: 10.1109/TETCI.2019.2926769.

[40] H. R. Pamuluri, “Predicting User Mobility using Deep Learning Methods,” MSc Thesis, Dept. of Comput. Sci., Blekinge Institute of Technology SE–371 79, Karlskrona, Sweden, 2021. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:1416798/FULLTEXT01.pdf

[41] C. Yao, J. Guo and C. Yang, "Achieving high throughput with predictive resource allocation", Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP), pp. 768-772, Dec. 2016.

[42] X. Liu, J. Zhang, X. Zhang and W. Wang, "Mobility-aware coded probabilistic caching scheme for MEC-enabled small cell networks", IEEE Access, vol. 5, pp. 17824-17833, 2017.

[43] C. Wang, Z. Zhao, Q. Sun and H. Zhang, "Deep learning-based intelligent dual connectivity for mobility management in dense network", Proc. IEEE VTC Fall, pp. 1-5, Aug. 2018.

[44] N. Prihodko, “Machine Learning for Forecasting Signal Strength in Mobile Networks,” MSc Thesis, Dept. of Comput. Sci., Univ. of Linköping University,Linköping, Sweden, 2021. [Online]. Available: https://liu.diva-portal.org/smash/get/diva2:1288566/FULLTEXT01.pdf

[45] F. Altché and A. de La Fortelle, "An LSTM network for highway trajectory prediction", Proc. IEEE 20th Int. Conf. Intell. Transp. Syst. (ITSC), pp. 353-359, Oct. 2017.

[46] A. Sarkar, K. Czarnecki, M. Angus, C. Li and S. Waslander, "Trajectory prediction of traffic agents at urban intersections through learned interactions", Proc. IEEE 20th Int. Conf. Intell. Transp. Syst. (ITSC), pp. 1-8, Oct. 2017.

[47] J. M. Perkel, “Why Jupyter is data scientists’ computational notebook of choice,” Oct. 30, 2008 Accessed on: Aug. 10, 2021. [Online]. https://www.nature.com/articles/d41586-018-07196-1

[48] “IPython Notebook Quick Start Guide,” Accessed on: Aug. 10, 2021. [Online]. https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id7

[49] Y. Zheng, X. Xie and W.-Y. Ma, "GeoLife: A collaborative social networking service among user location and trajectory", IEEE Data Eng. Bull., vol. 33, no. 2, pp. 32-39, Jun. 2010.

[50] D. Mwiti, “Using a Keras Long Short-Term Memory (LSTM) Model to Predict Stock Prices,” Accessed on: Aug. 10, 2021. [Online]. https://www.kdnuggets.com/2018/11/keras-long-short-term-memory-lstm-model-predict-stock-prices.html

[51] J. Brownlee “On the Suitability of Long Short-Term Memory Networks for Time Series Forecasting,” May 26, 2017 Accessed on: Aug. 10, 2021. [Online]. https://machinelearningmastery.com/suitability-long-short-term-memory-networks-time-series-forecasting/

[52] Anita Yadav, C K Jha, Aditi Sharan, Optimizing LSTM for time series prediction in Indian stock market, Procedia Computer Science, Volume 167, 2020, Pages 2091-2100, https://www.sciencedirect.com/science/article/pii/S1877050920307237

[53] Junyoung Chung et al. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. 2014. arXiv: 1412.3555 [cs.NE].

[54] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[55] J. Brownlee “Stacked Long Short-Term Memory Networks g,” Aug 18, 2017 Accessed on: Aug. 10, 2021. [Online]. https://machinelearningmastery.com/stacked-long-short-term-memory-networks/

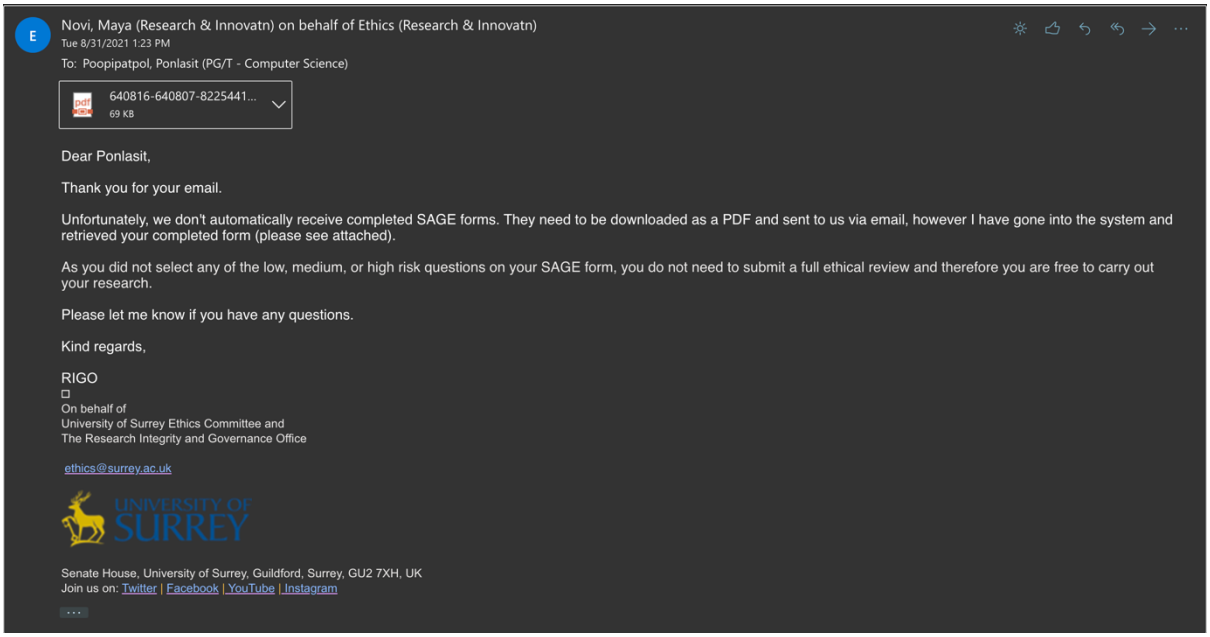
[56] J. Brownlee “How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras,” June 16, 2017 Accessed on: Aug. 10, 2021. [Online]. https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

[57] Alex Graves. Generating Sequences With Recurrent Neural Networks. 2013. arXiv: 1308.0850 [cs.NE].

[58] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: Neural computation 9 (Dec. 1997), pp. 1735–80. doi: 10.1162/neco.1997. 9.8.1735.

Appendix

This dissertation has already undergone an ethical review. The screenshot email below displays verification of approval from the ethics committee, indicating permission to proceed with the research, ensuring that the project's research, implementation, and conclusions could begin in an ethical manner. Moreover, this dissertation does not need us to do other types of research, such as surveys, which would have required more information from human participants.



Novi, Maya (Research & Innovatn) on behalf of Ethics (Research & Innovatn)
Tue 8/31/2021 1:23 PM
To: Poopipatpol, Ponlasit (PG/T - Computer Science)

640816-640807-8225441...
69 KB

Dear Ponlasit,

Thank you for your email.

Unfortunately, we don't automatically receive completed SAGE forms. They need to be downloaded as a PDF and sent to us via email, however I have gone into the system and retrieved your completed form (please see attached).

As you did not select any of the low, medium, or high risk questions on your SAGE form, you do not need to submit a full ethical review and therefore you are free to carry out your research.

Please let me know if you have any questions.

Kind regards,

RIGO
On behalf of
University of Surrey Ethics Committee and
The Research Integrity and Governance Office

ethics@surrey.ac.uk

UNIVERSITY OF SURREY

Senate House, University of Surrey, Guildford, Surrey, GU2 7XH, UK
Join us on: Twitter | Facebook | YouTube | Instagram