1. **ABSTRACT**

Human mobility prediction is of great importance in a wide range of modern applications in different fields such as personalized recommendation systems, the fifth-generation (5G) mobile communication systems, and so on. Generally, the prediction goal varies from different application scenarios. For the applications of 5G network including resource allocation and mobility management, it is essential to predict the positions of mobile users in the near future from dozens of seconds to a few minutes so as to make preparation in advance, which is actually a trajectory prediction problem.

With the particular focus on multi-user multi-step trajectory prediction, we first design a basic deep learning-based prediction framework, where the long short-term memory (LSTM) network is directly applied as the most critical component to learn user-specific mobility pattern from the user's historical trajectories and predict his/her movement trends in the future. Motivated by the related findings after testifying and analyzing this basic framework on a model-based dataset, we extend it to a region-oriented prediction scheme.

We propose an LSTM-based single-user prediction framework and evaluate its performance on a model-based dataset. Experimental results demonstrate the capability of LSTM to predict user’s mobility based on pre-learning of the user’s mobility patterns. We also highlight some challenges (e.g., poor generalization ability, annoying error-accumulation effect) of this user-speciﬁc prediction scheme. To cope with these challenges, we further extend the user speciﬁc prediction scheme to a region-oriented prediction scheme and put forward a multi-user multi-step trajectory prediction framework based on the Seq2Seq learning.

Finally, we show empirically that the proposed multi-user multi-step trajectory prediction framework inclusive of the LSTM model and ELM algorithm can effectively mitigate the error-accumulation effect and improve the generalization ability on a realistic dataset.

1. **INTRODUCTION**

Increasing pervasive usage of smart-phones and location-based services around the world has contributed to vast and rapid growth in mobility data. The large size of mobility data provides new opportunities for discovering the characteristics of human mobility patterns and making mobility predictions. Practically, human mobility prediction is of great importance in a wide range of modern applications, ranging from personalized recommendation systems to intelligent transportation, urban planning, and mobility management in the fifth-generation mobile communication system. Generally, the prediction goal varies from different application scenarios.

For the case of mobile communications, it is essential to predict the positions of mobile users in the near future from dozens of seconds to a few minutes so as to prepare for mobility management and resource allocation. It is actually a trajectory prediction problem where the trajectory refers to a time series of positions with a fixed sampling time interval between each other.

Here, we are describing concept to predict next location of single or multiple users by training trajectories i.e., latitude and longitude of their previous locations using RNN (Recurrent Neural Networks) advance version called LSTM (Long-Term Short-Term Memory) and Seq2Seq (sequence to sequences) algorithms. Predicting location of users plays an important role for 5G Internet networks as network service providers need to allocate nearest resources (cloud servers who take users mobile heavy computation task and process that request and send result back to mobile, if nearest cloud allocate to user then response will be faster and this nearest allocation can be done if users next locations can be predicted) to users to process their mobile request data. We have considered the use of deep learning methods such as Extreme Learning machine which have a promising result in the decrement of errors and which can be proved to be one of the most optimal algorithms to provide us with the best of results.

Beneﬁting from the latest advance in deep learning, this paper makes a detailed exploration of the trajectory prediction problem from both the single-user perspective and multiuser perspective. We propose an LSTM-based single-user prediction framework and evaluate its performance on a model-based dataset. Experimental results demonstrate the capability of LSTM to predict user’s mobility based on pre-learning of the user’s mobility patterns. We also highlight some challenges (e.g., poor generalization ability, annoying error-accumulation effect) of this user speciﬁc prediction scheme. To cope with these challenges, we further extend the user-speciﬁc prediction scheme to a region-oriented prediction scheme and put forward a multi-user multi-step trajectory prediction framework based on the Seq2Seq learning. Finally, we show empirically that the proposed multiuser multi-step trajectory prediction framework can effectively mitigate the error-accumulation effect and improve the generalization ability on a realistic dataset.

There have been some theoretical mobility models proposed to mimic the movements of mobile users and simulate their mobility patterns using parametric methods synthetically, such as Random Walk mobility model, Gauss-Markov mobility model, Levy-Walk mobility model, and so on. Although these models are relatively simple, they can hardly describe the movement of different users in a complex and volatile real environment, making it unreasonable to apply them in practice. Besides, a number of previous efforts have attempted to model user mobility based on real-life movement trajectories. Early methods related to mobility prediction mainly focus on discovering frequent trajectories and then performing trajectory matching to predict the location of a moving object. However, these methods are computational cost and suffer from the data sparsity problem. Another widely used mobility prediction methods fall into the scope of Markov-based models. The authors in propose a hidden Markov model-based trajectory prediction algorithm to discover transition rules from one location to another. Lv et al. further combine the HMM model with user’s living habits for an individual to achieve effective location prediction. In addition, other conventional machine learning techniques such as K-nearest neighbor (KNN) and decision tree have also been applied for location recognition and prediction in. However, these methods need the locations to be discrete, thus not applicable to trajectories composed of continuous coordinates with small sampling time intervals. Within the framework of deep learning, the work in applies LSTM to trajectory prediction for vehicles on highway. However, the proposed method is speciﬁcally designed for the highway scenario and requires complex external features, including position and velocity of surrounding vehicles, which restricts its general applicability. Alahi el al. proposes a social LSTM network for pedestrian trajectory prediction. However, it can only predict human trajectories through static-images under a speciﬁc small range scene such as hotels and intersections. Feng et al. propose a Deep Move model which combines the GRU network with the attention mechanism to predict future discrete locations from long-range and sparse trajectories. However, its prediction accuracy can only reach cellular network scenarios since it is difﬁcult to capture the trend of user movements in each cell from trajectories composed of discrete cells. Trajectory prediction has a wide range of applications in networks, such as radio resource pre-allocation, caching decision at the wireless edge, mobility management, and etc. For example, in order to mitigate the negative impact of frequent hand overs in dense networks, our previous work in proposes an intelligent dual connectivity mechanism for mobility management based on trajectory prediction, which improves the quality of service of mobile users in the handover process while guaranteeing the network energy efﬁciency. Moreover, driven by the stringent safety requirement of autonomous driving and advanced driver assistance systems, it is critical to understand the intentions of surrounding vehicles through trajectory prediction. Therefore, trajectory prediction is a problem worth well careful studying.

Different users typically have distinct mobility patterns, making the mobility prediction problem to be user-speciﬁc. Therefore, in order to make mobility predictions for a user, the most critical step is to establish a speciﬁc mobility model which fully represents the user’s mobile pattern from his/her historical trajectories. The prediction process involves three major steps. First, the given trajectory is processed by a fully connected (FC) input layer with 128 neurons so that each two-dimensional coordinate is mapped to a 128-dimensional feature tensor. Then, the processed sequence is sent to the main part of the mobility model, a deep recurrent neural network formed by three stacked LSTM layers each with 128 neurons. Each LSTM layer takes the output of the previous layer as input and feeds its output to the next layer. Finally, an FC output layer with 2 neurons maps the output of the last LSTM layer at each time-step to a two-dimensional coordinate as the predicted location of the next time-step, and thereby we get the prediction sequence. The training goal is to minimize the distance error between the predicted location and the actual location. Thus, we choose the Mean Square Error (MSE) as the loss function to update the network parameters. Ultimately, the user’s mobility pattern is saved in the mobility model as network parameters and the prediction of future trajectory can be completed based on the trained mobility model.

GeoLife is alocation-based social-networking service, which enables users to share life experiences and build connections among each other using human location history. Dr. Yu Zheng started this project in 2007 with his team. GeoLife enables user to share travel experience using GPS trajectories. By mining multiple users’ location histories, GeoLife can discover the top most interesting locations, classical travel sequences and travel experts in a given geospatial region, hence enable a generic travel recommendation. By understanding individual location history, GeoLife can measure the similarity between users and perform personalized friend & location recommendation.

By uploading your GPS data and associated multimedia content like photos to the website of GeoLife, you can interact with your trajectory like playing a video. First, you can enjoy and memorize your past experiences on a map. Second, you can share it with your friends. Thus, your friends can know where you have been, see what you saw and understand the whole journey within a few seconds. It is more intuitive and convenient than writing and reading a blog.

By mining multiple users’ location histories, GeoLife can automatically discover the top most interesting locations and classical travel sequences in a given geographical region. The information can enable generic travel recommendation, which helps users understand an unfamiliar city within a short period and plan a trip with minimal effort.

GeoLife can recommend you a group of users in terms of the similarity between your location histories and theirs. As people’s location histories imply to some extent their tastes and preferences, these users, called potential friends, might share similar interests with you. With this friend list, you can conveniently deliver invitations to these persons in the community and hence sponsor, with minimal effort, a social activity such as hiking, cycling, or travelling. As they share similar interest with you, they are more likely to accept your invitation. Further, from these potential friends’ past experiences, you are more likely to discover some places that might match your tastes while have not been found by yourself. It is a personalized location recommendation.

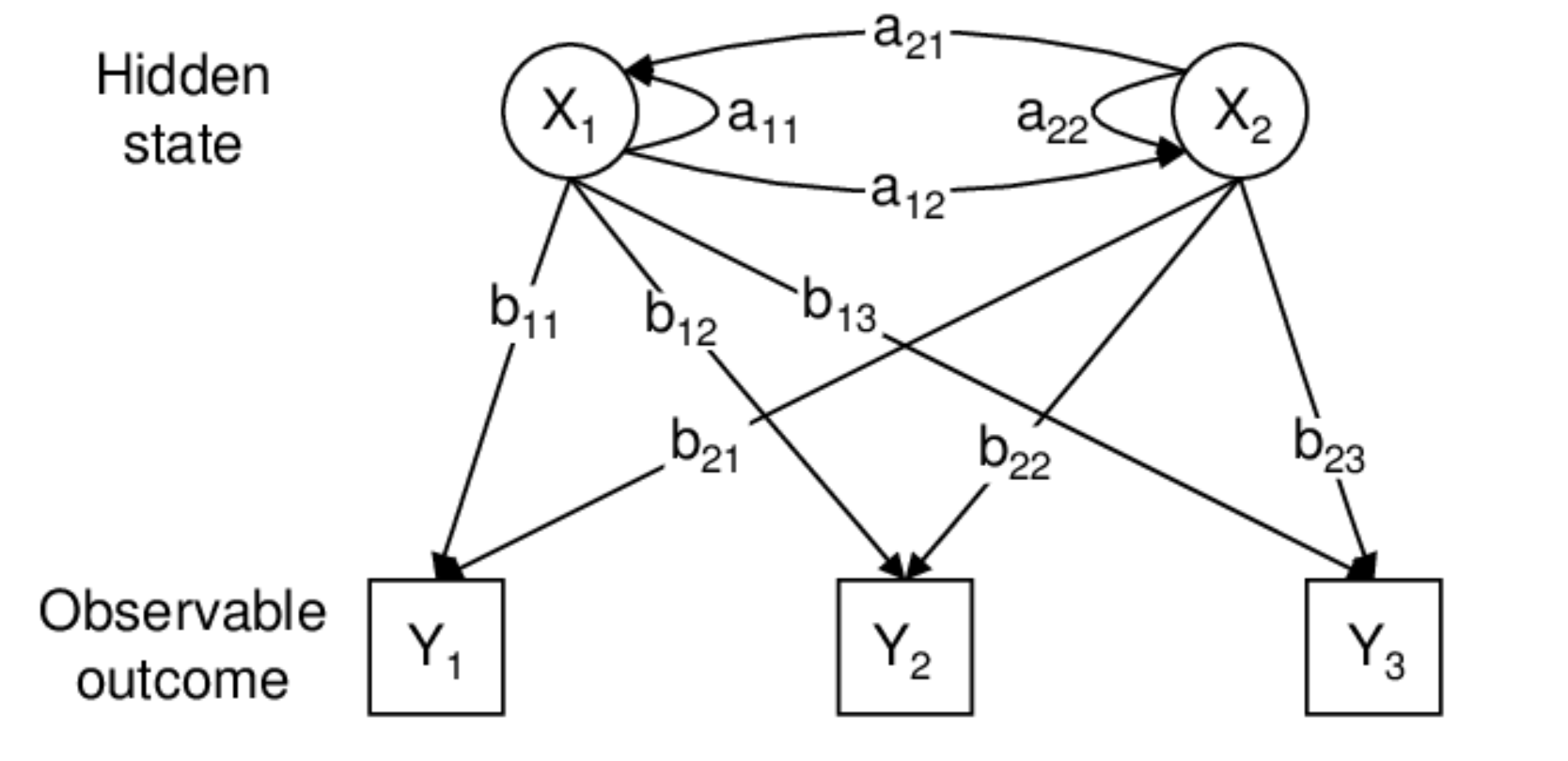
1. **LITERATURE REVIEW**

Trajectory prediction is a very demanding process for many fields of work, these include travel companies, geographic companies, department of police etc. our work is based of the paper “Exploring Trajectory Predictions using Machine Learning Methods”, published by Chujie Wang, Lin Ma, Rongpeng Li, Tariq S. Durrani, and Honggang Zhang. The authors of the base paper have utilized simpler methods such as Linear regression and Support Vector Regression. Although researchers have proposed many mobility prediction methods, such as frequent patterns mining, Markov-based models and other machine learning methods, most of these methods are dedicated to discrete location prediction, which is actually a multi-classiﬁcation problem, and not suitable for predicting trajectories with ﬁxed sampling time intervals.

The initial validated algorithms for trajectory data are linear regression, support vector regression, LSTM and GRU. Linear Regression is a conventional machine learning algorithm to discover linear relationships among data for regression problems. Support Vector Regression (SVR) is another conventional machine learning algorithm which can cope with non-linear problem based on kernel method. LSTM is one of the recurrent neural networks with gate control mechanism and has shown its superiority in encoding long-term dependencies. GRU is a simplified version of LSTM with only reset gate and update gate, which has less computational complexity.

**Linear Regression** is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression. The regression line is the best fit line for our model.

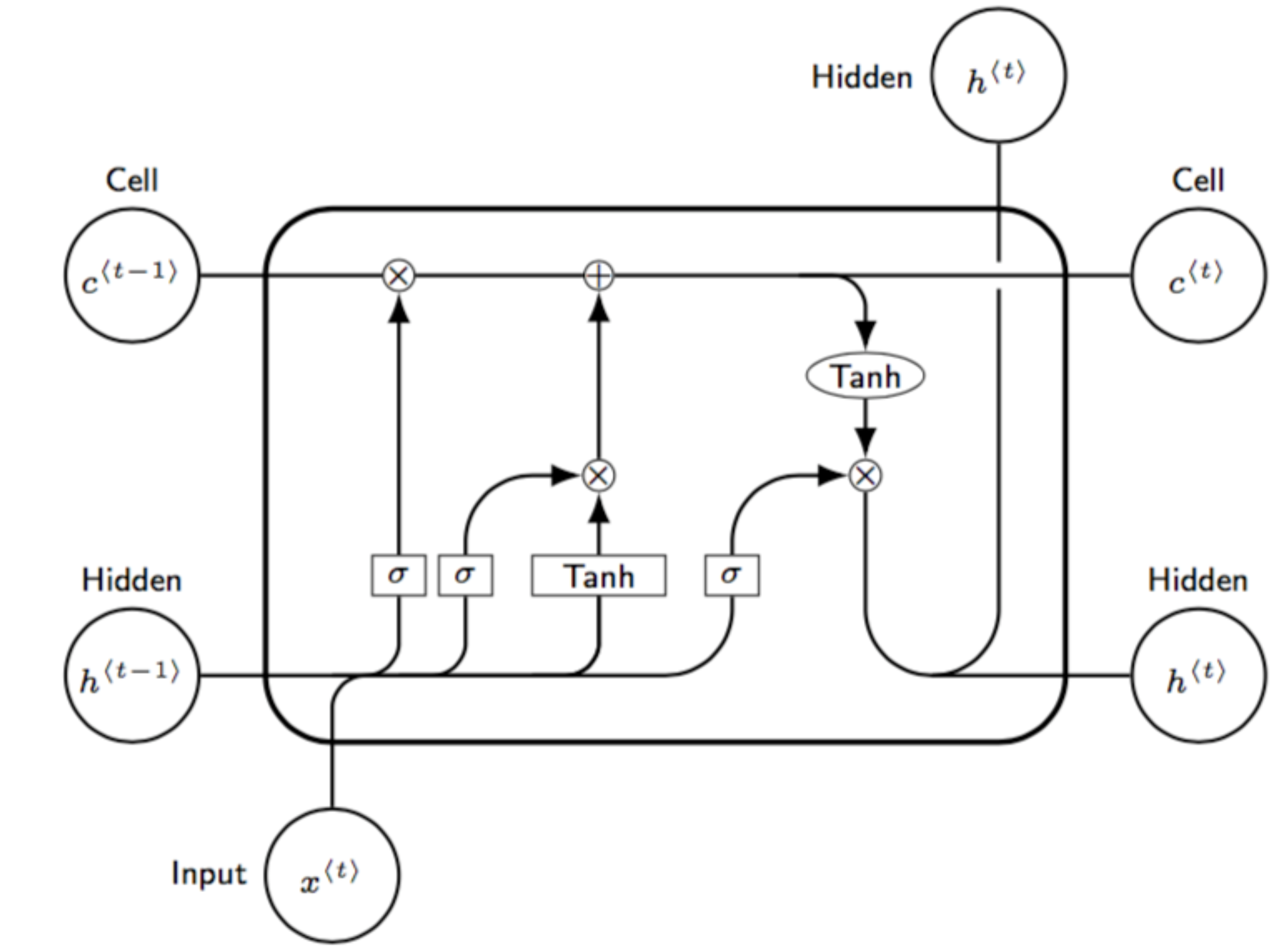
Support Vector Regression (SVR) uses the same principle as SVM, but for regression problems. The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample. Consider these two red lines as the decision boundary and the green line as the hyperplane**. Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line.** Our best fit line is the hyperplane that has a maximum number of points.



**Fig 3.1** Hidden Markov Model

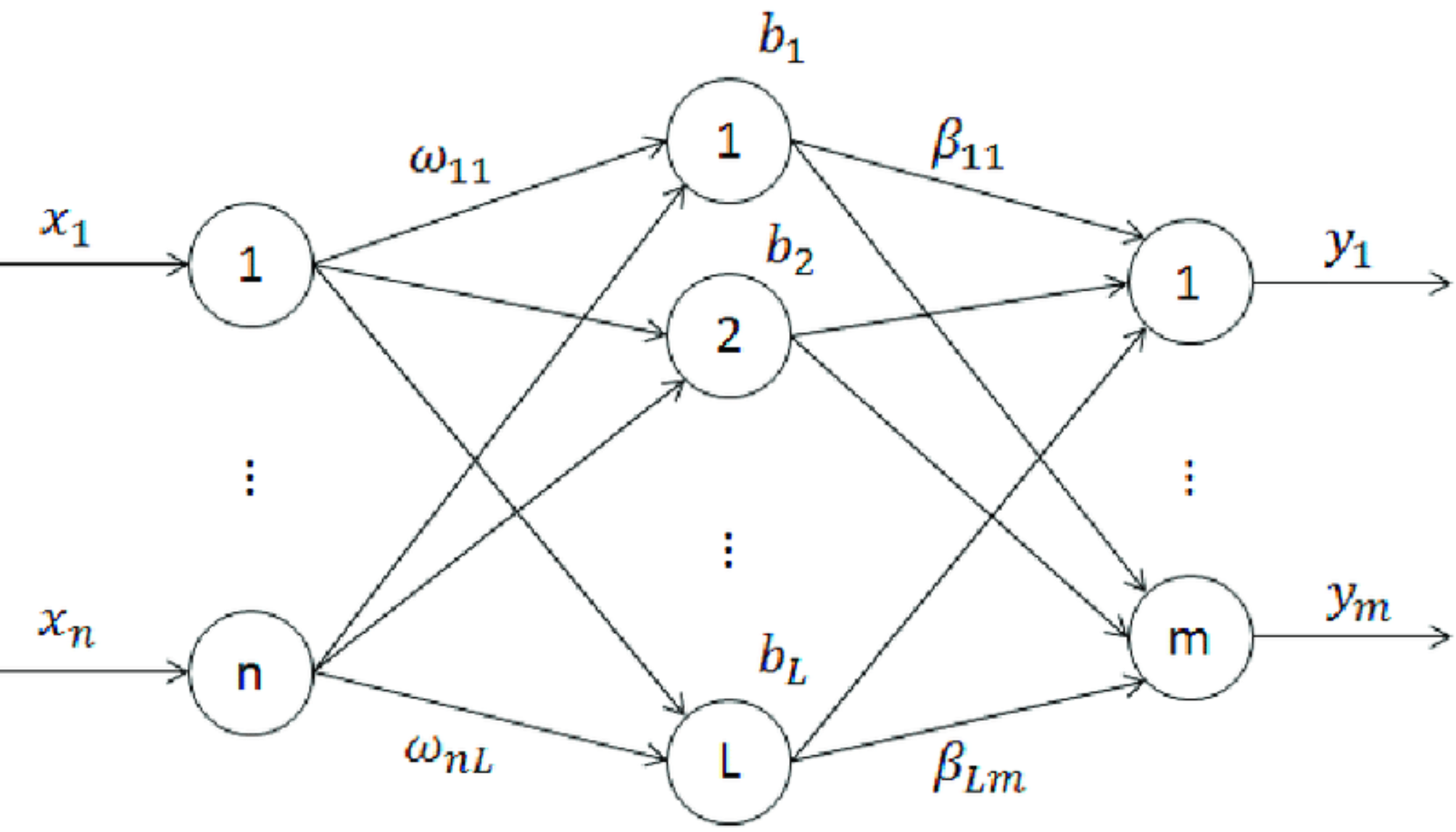
With the advances in deep learning, more optimal methods have come into light which provide better results and cover up discrepancies from the other simpler algorithms. Within the framework of deep learning, the Recurrent Neural Network (RNN) has proved its superiority in various time series problems not only in natural language processing ﬁeld (i.e. machine translation, speech recognition) but also some other ﬁelds (i.e. trafﬁc prediction, precipitation prediction). Therefore, as the improved versions of typical RNN, Long-Term Short-Term Memory (LSTM) and Extreme Learning Machine (ELM) which are promising algorithms for the trajectory prediction problem. However, the proposed method is speciﬁcally designed for the highway scenario and requires complex external features, including position and velocity of surrounding vehicles, which restricts its general applicability.

GRU (Gated Recurrent Unit) aims to solve the **vanishing gradient problem**which comes with a standard recurrent neural network. GRU can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce equally excellent results. If you are not familiar with Recurrent Neural Networks. To solve the vanishing gradient problem of a standard RNN, GRU uses, so-called**, update gate and reset gate.** Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction**. The update gate helps the model to determine how much of the past information (from previous time steps) needs to be passed along to the future.**That is really powerful because the model can decide to copy all the information from the past and eliminate the risk of vanishing gradient problem. We will see the usage of the update gate later on. Reset **gate is used from the model to decide how much of the past information to forget.** The difference comes in the weights and the gate’s usage. GRUs are able to store and filter the information using their update and reset gates. That eliminates the vanishing gradient problem since the model is not washing out the new input every single time but keeps the relevant information and passes it down to the next time steps of the network. **If carefully trained, they can perform extremely well even in complex scenarios.**



**Fig 3.3** Structure of LSTM of RNN

Within the framework of deep learning, the work in applies LSTM to trajectory prediction for vehicles on highway. However, the proposed method is speciﬁcally designed for the highway scenario and requires complex external features, including position and velocity of surrounding vehicles, which restricts its general applicability. Alahi el al. proposed a social LSTM network for pedestrian trajectory prediction. However, it can only predict human trajectories through static images under a speciﬁc small range scene such as hotels and intersections. Feng et al. proposed a Deep Move model which combines the GRU network with the attention mechanism to predict future discrete locations from long-range and sparse trajectories. However, its prediction accuracy can only reach 59.3% in cellular network scenarios since it is difﬁcult to capture the trend of user movements in each cell from trajectories composed of discrete cells. We establish a Seq2Seq framework based on the LSTM encoder-decoder architecture to capture the temporal association within the trajectory like speed or direction. All trajectories in the speciﬁc area are utilized for the network to acquire the shared short-term mobility patterns caused by geographical constraints. Multi-user multi-step prediction promises to bring lots of signiﬁcant merits. Firstly, it allows for more practical nearreal-time resource pre-allocation. But it has to deal with the annoying error-accumulation effect. Secondly, the generalization ability of the prediction model across users also makes it feasible to quickly perform trajectory prediction for any user. Thirdly, the computation overhead of training a model for each user separately can be signiﬁcantly reduced. Therefore, we consider the real-world user movement scenario and propose a multi-user multi-step trajectory prediction framework.



**Fig3.2** Network Structure of Extreme Learning Machine

Extreme learning machine (ELM) is a new learning algorithm for the single hidden layer feed-forward neural networks. They are feed-forward neural networks for classification, regression, clustering, sparse approximation, compression and feature learning with a single layer or multiple layers of hidden nodes, where the parameters of hidden nodes (not just the weights connecting inputs to hidden nodes) need not be tuned. These hidden nodes can be randomly assigned and never updated (i.e. they are random projection but with nonlinear transforms), or can be inherited from their ancestors without being changed. In most cases, the output weights of hidden nodes are usually learned in a single step, which essentially amounts to learning a linear model. Compared with the conventional neural network learning algorithm it overcomes the slow training speed and over-fitting problems and ELM is based on empirical risk minimization.

1. **PROBLEM IDENTIFICATION AND OBJECTIVES**

**4.1 PROBLEM IDENTIFICATION**

Although researchers have proposed many mobility prediction methods, such as frequent patterns mining, Markov-based models and other machine learning methods, most of these methods are dedicated to discrete location prediction, which is actually a multi-classiﬁcation problem, and not suitable for predicting trajectories with ﬁxed sampling time intervals. The reasons are as follows. On one hand, for trajectories composed of discrete location indexes, locations may keep same for several consecutive time-steps when the sampling time interval is small, while locations may have a mutation between two adjacent time-steps when the sampling time interval is large. Therefore, they can hardly reﬂect user movement trends effectively. On the other hand, for trajectories composed of continuous location co-ordinates, it is hard to specify the discretization granularity of coordinates. Generally, high discretization granularity beneﬁts to reﬂect user movement trends. However, the prediction accuracy may decrease with increasing number of candidate locations under high discretization granularity.

**4.2 OBJECTIVE**

In this project, we are describing concept to predict next location of single or multiple users by training trajectories (users previous location movement latitude and longitude) of their previous locations using RNN (Recurrent Neural Networks) advance version called LSTM (Long-Term Short-Term Memory) and ELM (extreme learning machine) algorithms. Predicting location of users plays an important role for 5G Internet networks as network service providers need to allocate nearest resources (cloud servers who take users mobile heavy computation task and process that request and send result back to mobile, if nearest cloud allocate to user then response will be faster and this nearest allocation can be done if users next locations can be predicted) to users to process their mobile request data.

**4.3 BENEFITS OF LSTM AND ELM**

1. ELM is an efficient learning algorithm for the single hidden layer feed forward neural networks. Compared with the other conventional neural network algorithm it has the advantage of over-fitting problems and slow training speed.
2. The ELM algorithm completes the whole process at once and generates a unique optimal solution without the necessity of iterations. So, it has the advantages of easy parameter selection and fast learning speed.
3. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration.
4. LSTM’s have a nature of remembering information for a long period of time.

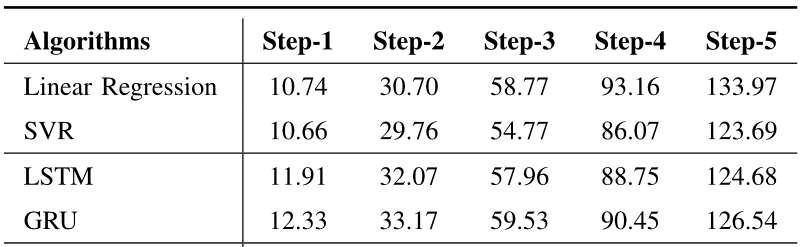


Table comparing the predictions made using different algorithms.

The above table gives a comparison of prediction by different algorithms taking instance of a data from the dataset. It depicts that the LSTM gives the closest results than the other algorithms.

### 4.4 SCOPE

Our current work does not consider the semantic context in the trajectory like the point of interests because of the limitation of data. For future work, we plan to combine our algorithm with some semantic information to improve the prediction performance.

* + 1. **SOFTWARE REQUIREMENTS**

**HOSTING ENVIRONMENT**

* + - * Operating System: Windows 7 or later, Mac OS 10.9 or later or modern Linux
      * Support Software: Visual Studio Code, v7 or higher, Node Js v8 or higher, Oracle 10g Express Edition, Anaconda
      * Browser: Google Chrome v70 or higher.

**DEPLOYING ENVIRONMENT**

* + - * Operating System: Windows XP or later, Mac OS 7.0 or later or modern Linux, Anaconda, Jupyter Notebook
      * Recommended Browser: Google Chrome (Across All Platforms)
    1. **HARDWARE REQUIREMENTS**
       - RAM: Recommended 2 GB free
       - Storage: Recommended 10 GB free
       - Processor: Intel Pentium 4 (1.5 GHz) or higher
    2. **DATASET**

In order to evaluate the performance of the mobility prediction framework, we adopt two types of datasets (i.e., a model-based dataset and a realistic dataset) for the reasons as follows. Given the strong randomness of user mobility, it is necessary to testify the performance of a proposed algorithm in a realistic environment. But considering that a realistic dataset is often collected in a user-voluntary manner, the dataset usually consists of user mobility trajectories lasting short duration and possessing irregular starting and ending time. Hence, we generate a model-based dataset from well-known models to assist in ﬁnding some intuitive guidance.

We utilize a large real-life GPS trajectory dataset from the Geolife project of Microsoft Research Asia. The dataset was collected by 182 users, containing 18,670 trajectories with various sampling rates. Each trajectory is represented by a series of time stamp points with latitude and longitude coordinates recorded by GPS-functioned phones. As an essential work for a large and messy raw dataset, we take the following preprocessing steps.

1. **SYSTEM METHODOLOGY**

**5.1 INTRODUCTION**

The design phase of software development deals with transforming the customer requirements into a form implementable using a programming language. The software design process can be divided into the following three levels of phases of design:

● System Design

● Architectural Design

● Detailed Design

This chapter provides the design phase of the Application. To design the project, we use the UML diagrams. The Unified Modelling Language (UML) and Design diagrams are a general- purpose, developmental, modelling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.

**5.2 DESIGN DIAGRAM:**

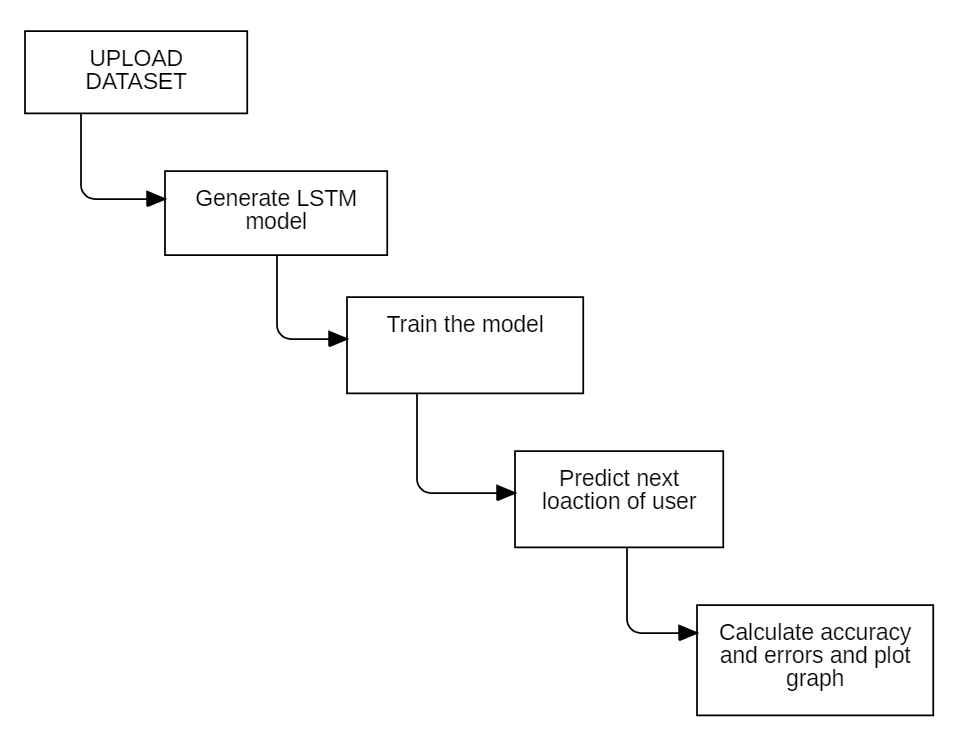


Fig 5.2 Design Diagram

The above Design diagram represents the workflow of the project at various stages. The first step of the project is to upload the dataset. Once the dataset is uploaded, we generate the LSTM model. The LSTM model generally consists of multiple layers. Here, we use the input layer, output layer and the dense layer.

After we create the LSTM model, we train the model in accordance to our dataset provided by giving the train and test sets. The dataset in a csv file is passed on to the model and the list prepared in the beginning which consists of the values of dataset in a list format. All these resources are used in a processed way to train the model. And a certain number of epochs are run while training the model. The metrics for the epoch are also displayed accordingly. The next function is to predict the next location user using LSTM model created. The function requests for certain values of the user like longitude, latitude and user id. By providing those values to the model we predict the next location of the user. The model passes the data through the network created and predicts the next values. Once the values are predicted, we calculate the accuracy and the MSE error of the model.

The next model is invoked simultaneously after the completion of the LSTM model predictions. The prompt asks for the id of the user. Once the id is entered, the model displays all the full path of the user with the prediction for the next step. This makes it easy for comparison and for calculating MSE error. The MSE error is also displayed.

Once the calculations and predictions are completed the model creates a graphical representation of the comparison of MSE error in between the two models to present which model is better than the other. In this context, ELM model proves most effective in all the cases.

1. **OVERVIEW OF TECHNOLOGIES**

**6.1 ENVIRONMENT AND LIBRARIES**

Anaconda Command Prompt is a library for prompting input on the command line for Python 3.3+. It is pure Python code with no dependencies. Anaconda command prompt is just like command prompt, but it makes sure that you are able to use anaconda and conda commands from the prompt, without having to change directories or your path. When you start Anaconda command prompt, you'll notice that it adds/("prepends") a bunch of locations to your PATH. These locations contain commands and scripts that you can run. So as long as you're in the Anaconda command prompt, you know you can use these commands. During the installation of Anaconda there is a choice to add these to the PATH by default, and if checked you can also use these commands on the regular command prompt. But the anaconda prompt will always work.

**6.1.1 PANDAS**

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

**6.1.2 NUMPY**

NumPy is a library for the Python programming language, adding support for large, multidimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

**6.1.3 MATPLOTLIB**

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

**6.1.4 SKLEARN**

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

**6.1.5 MATH**

This module provides access to the mathematical functions defined by the C standard. These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don’t is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

**6.2 LSTM**

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they’ll have a hard time carrying information from earlier time steps to later ones. So, if you are trying to process a paragraph of text to do predictions, RNN’s may leave out important information from the beginning. During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weight. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn’t contribute too much learning.

LSTM ’s was created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information.

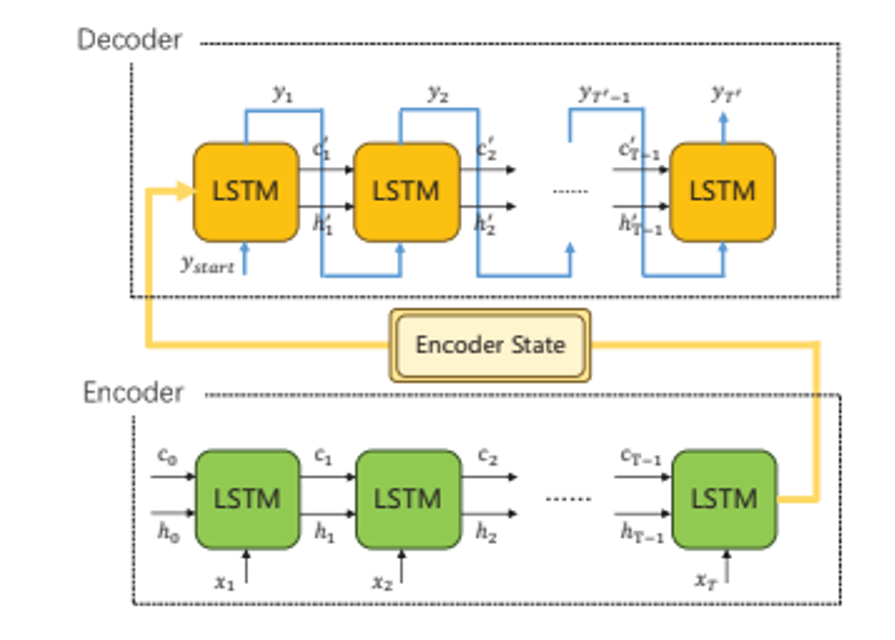


Fig. 6.2 Different Components and workflow of LSTM

An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM’s cells.

The core concept of LSTM’s is the cell state, and its various gates. The cell state act as a transport highway that transfers relative information all the way down the sequence chain. You can think of it as the “memory” of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. As the cell state goes on its journey, information get’s added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

**6.3 Extreme Learning Machine (ELM)**

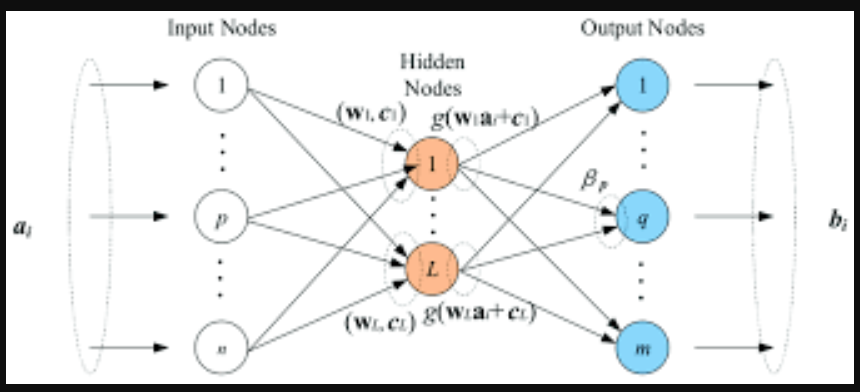


Fig 6.3. The general pictorial representation of Extreme Learning Machine.

Extreme Learning Machine (ELM) is a novel method for pattern classification as well as function approximation. This method is essentially a single feedforward neural network; its structure consists of a single layer of hidden nodes, where the weights between inputs and hidden nodes are randomly assigned and remain constant during training and predicting phases. On the contrary, the weights that connect hidden nodes to outputs can be trained very fast. Experimental studies in the literature showed that ELMs can produce acceptable predictive performance and their computational cost is much lower than networks trained by the back-propagation algorithm.

1. **IMPLEMENTATION**
   1. **IMPLEMENTATION IN ANACONDA PROMPT**

Anaconda Individual Edition contains conda and Anaconda Navigator, as well as Python and hundreds of scientific packages. When you install Anaconda, you install all these. Conda works on your command line interface such as Anaconda Prompt on Windows and terminal on macOS and Linux. Navigator is a desktop graphical user interface that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands.

**7.2 ALGORITHM**

LSTM:

1. Concatenate previous hidden state and present input ->combine

2. Forget layer <- combine

3. Create Candidate layer by utilizing combine

4. Input Layer <- combine

5. Compute forget layer, input layer and candidate layer.

6. Calculate cell state using previous state and the derived vectors.

7. Compute output

EXTREME LEARNING MACHINE

1. Randomly assign the input weights ‘w’ and biases ‘b’.
2. Calculate hidden layer output matrix ‘H’
3. Calculate the output weights matrix.
4. Compute outputs accordingly.

**7.3 FLOWCHART**

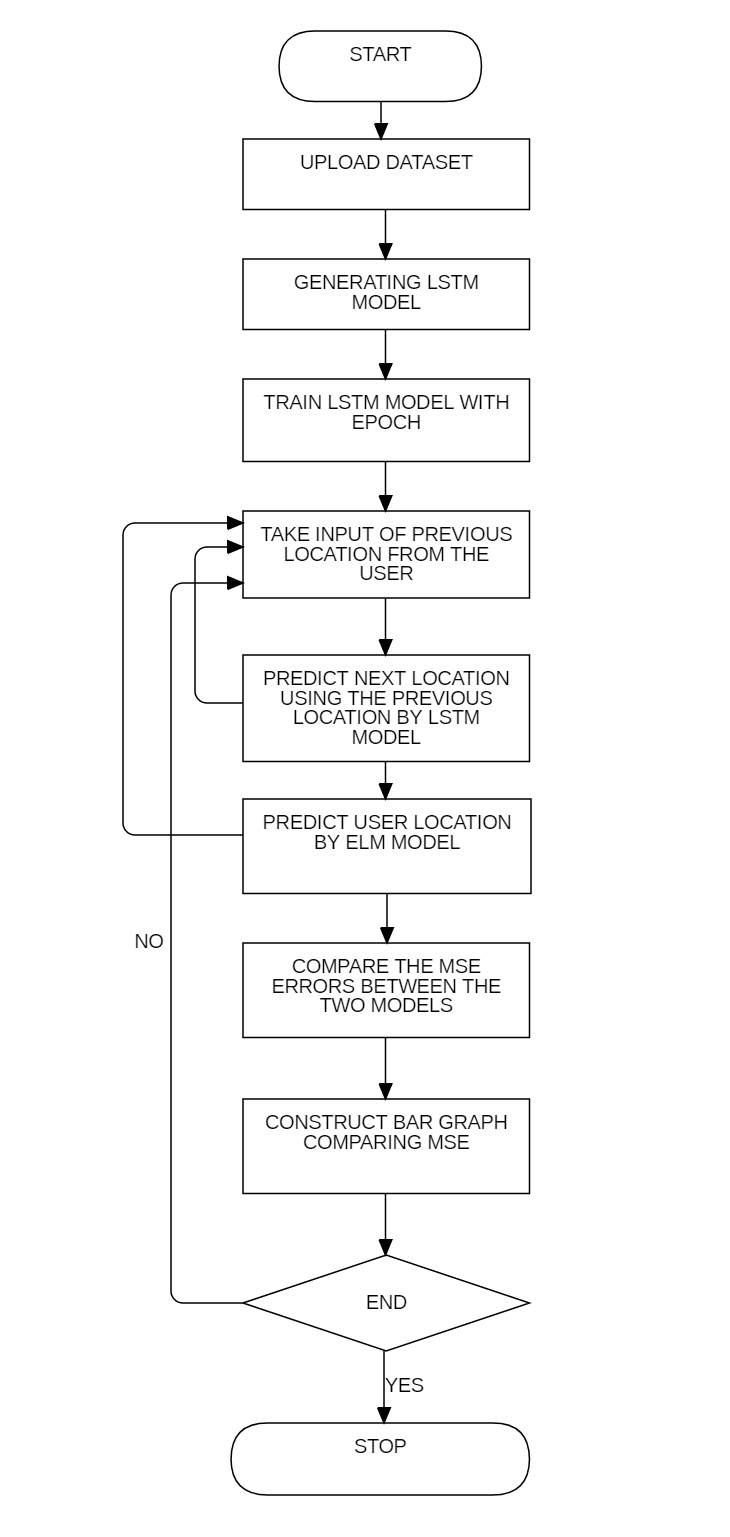


Fig 7.1 Flow chart

**7.4 CODING**

**7.4.1 LSTM**

def lstmModel():

global model, encoder\_model, decoder\_model

encoder\_inputs = Input(shape=(None, 9))

encoder = LSTM(512, return\_state=True)

encoder\_outputs, state\_h, state\_c = encoder(encoder\_inputs)

encoder\_states = [state\_h, state\_c]

decoder\_inputs = Input(shape=(None, 9))

decoder\_lstm = LSTM(512, return\_sequences=True, return\_state=True)

decoder\_outputs, \_, \_ = decoder\_lstm(decoder\_inputs, initial\_state=encoder\_states)

decoder\_dense = Dense(9, activation='softmax')

decoder\_outputs = decoder\_dense(decoder\_outputs)

model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)

encoder\_model = Model(encoder\_inputs, encoder\_states)

decoder\_state\_input\_h = Input(shape=(512,))

decoder\_state\_input\_c = Input(shape=(512,))

decoder\_states\_inputs = [decoder\_state\_input\_h, decoder\_state\_input\_c]

decoder\_outputs, state\_h, state\_c = decoder\_lstm(decoder\_inputs, initial\_state=decoder\_states\_inputs)

decoder\_states = [state\_h, state\_c]

decoder\_outputs = decoder\_dense(decoder\_outputs)

decoder\_model = Model([decoder\_inputs] + decoder\_states\_inputs, [decoder\_outputs] + decoder\_states)

print("LSTM Model Generated\n")

* + 1. **Extreme Learning Machine (ELM):**

global elm\_error

filename1=open("C:/Users/Krish/Desktop/maj\_proj/data1.csv","r")

user = input("Enter User ID")

print("\n")

train = pd.read\_csv(filename1)

X = train.values[:, 0:4]

y = train.values[:, 0]

trainX = []

trainY = []

trainY1 = []

for i in range(len(X)):

usr = X[i][0]

x\_loc = X[i][2]

y\_loc = X[i][3]

if str(usr) == user:

trainY.append(x\_loc)

trainY1.append(y\_loc)

trainX.append([x\_loc,y\_loc])

trainX = np.asarray(trainX)

trainY = np.asarray(trainY)

trainY1 = np.asarray(trainY1)

print(trainX)

svm\_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)

srhl\_tanh = MLPRandomLayer(n\_hidden=200, activation\_func='tanh')

cls = ELMRegressor(regressor=svm\_rbf)

cls.fit(trainX, trainY)

y\_pred = cls.predict(trainX)

srhl\_tanh = MLPRandomLayer(n\_hidden=200, activation\_func='tanh')

cls = ELMRegressor(regressor=svm\_rbf)

cls.fit(trainX, trainY1)

y\_pred1 = cls.predict(trainX)

err = []

for i in range(len(y\_pred)):

err.append([y\_pred[i],y\_pred1[i]])

err = np.asarray(err)

elm\_error = mean\_squared\_error(trainX, err)

length = len(y\_pred) - 1

print("\nELM Extension Next Predicted Sequence is :\n")

print("Latitude : "+str(y\_pred[length])+"\n")

print("Longitude : "+str(y\_pred1[length])+"\n")

print("ELM MSE Error : "+str(elm\_error))

**7.4.3 Training using LSTM model:**

def trainLSTM():

global dataset

global model

global lstm\_error,gru\_error

filename1=open("C:/Users/Krish/Desktop/maj\_proj/data1.csv","r")

train = pd.read\_csv(filename1)

size = len(train)

dataset = np.zeros((size, 9, 9))

m = 0;

n = 0

p = 0

for i in range(len(train)) :

person = int(train.iloc[i, 0])

position = int(train.iloc[i, 1])

latitude = float(train.iloc[i, 2])

longitude = float(train.iloc[i, 3])

n = 0

for j in range(len(train)):

person1 = int(train.iloc[j, 0])

position1 = int(train.iloc[j, 1])

latitude1 = float(train.iloc[j, 2])

longitude1 = float(train.iloc[j, 3])

if person == person1:

dataset[m][position1-1][n] = latitude1

n = n + 1

dataset[m][position1-1][n] = longitude1

n = n + 1

dataset[m][position1-1][n] = person

n = n + 1

if n >= 9:

n = 0

m = m + 1

print(dataset.shape)

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics = ['accuracy'])

print(model.summary())

model.fit([dataset,dataset], dataset, epochs=10)

scores = model.evaluate([dataset,dataset], dataset, verbose=2)

accuracy = scores[1]\*100

lstm\_error = 100.0 – accuracy

print("LSTM MSE Error : "+str(lstm\_error)+"\n")

model.save("draft\_traj\_model.h5py")

**7.2 TESTING**

**7.2.1 INTRODUCTION**

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements contrary to the actual requirements.

**WHITE BOX TESTING**

It is also called as Glass Box, Clear Box, and Structural Testing. White Box Testing is based on the application& internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level.

The testing is performed in order to check the data types that are used in modules and their performance testing when data type is changed with in the code. And further testing on the internal code can see in unit testing.

**BLACK BOX TESTING**

It is also called as Behavioral/Specification-Based/Input-Output Testing. Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure.

Thus, as the part of black box testing. The testing is done by passing the inputs about the users whose data has been used to train the model. This type of testing is generally done by passing the inputs manually or by passing a data frame to the model to test it’s working.

**GREY BOX TESTING**

The grey box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. This helps to create better test cases in this process.

As a use case for grey box testing, we have taken the data of users whose trajectory is not related to the data we have used for the model training. So, the model follows the trained neural network to predict the next sequence for the tested user.

**STATIC TESTING**

It is also known as Verification in Software Testing. Verification is a static method of checking documents and files. Verification is the process, to ensure that whether we are building the product right i.e., to verify the requirements which we have and to verify whether we are developing the product accordingly or not. Activities involved here are Inspections, Reviews, and walkthroughs.

In this testing we tried to review the resource of data we had and the material which can be referred to build this project. But looking at the map of our project we tried to incorporate all these into the desired phases of project map. And the main idea was to clear about building a model to predict next sequence out of the available information or the data provided for training. We followed the steps in a way to sort out the result we were expecting.

**DYNAMIC TESTING**

It is also known as Validation in Software Testing. Validation is a dynamic process of testing the real product. Validation is the process, whether we are building the right product i.e., to validate the product which we have developed is right or not. Activities involved in this are testing the software application.

As, the dynamic testing suggests we have followed certain testing procedures discussed ahead in order to test the software solution that is developed.

**UNIT TESTING**

Unit Testing is done to check whether the individual modules of the source code are working properly. I.e. testing each unit of the application separately by the developer in the developer’s environment.

The modules in the code are closely coupled. So, we had to follow the sequential procedure, the procedure which the flow of project follows. The modules are tested based on expected performance for the inputs. We tried to look at the individual’s modules if they were able to generate the output which is to be used by other modules to process.

**INTEGRATION TESTING**

Integration Testing is the process of testing the connectivity or data transfer between a couple of units tested modules. It is AKA I&amp; T Testing or String Testing.

We tested the type and compatibility issues during communication modules. If the modules we’re accepting the outputs which is generated by other modules as input to process.

**SYSTEM TESTING**

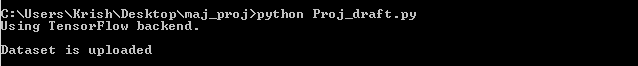
It is a black box testing. Testing the fully integrated application this is also called as the end to end scenario testing. To ensure that the software works in all intended target systems.

Verify thorough testing of every input in the application to check for desired outputs. The project has all of its modules in one code file. Thus, the testing was about running the code and checking if the desired execution is resulted or not.

1. **RESULTS AND DISCUSSION**

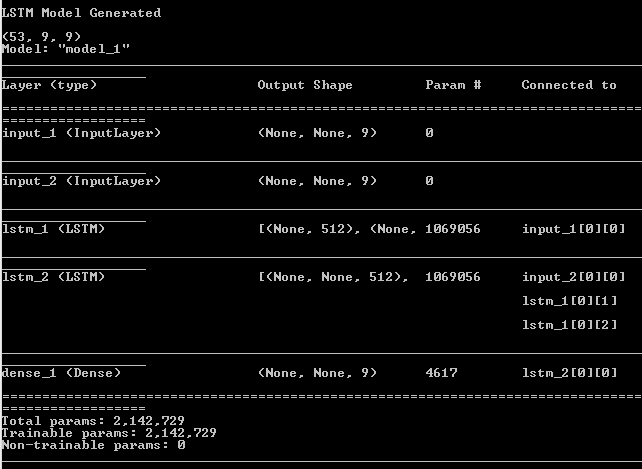
The multi user model includes data from multiple users. Thus, the prepared dataset include data from multiple user folders of the original geolife project dataset. The project code is developed in the python .py file. Therefore, the code is executed in the command prompt. The functions use TensorFlow modules to create deep neural network model for the multi user purpose.

During execution the main function invokes all the function calls in it. The upload function accesses the dataset file present in the specified location. Then it opens up the data present in dataset file under read access. The dataset is loaded into a variable and a list is prepared for further use.



**Fig 8.1** Running code file and dataset upload

Now the function lstm model is invoked in order to generate a LSTM (Long Short-Term Memory) model. This model is a deep neural network model that is designed to guide the process to the users next location. Thus, the model consists of some input, output and dense layers.



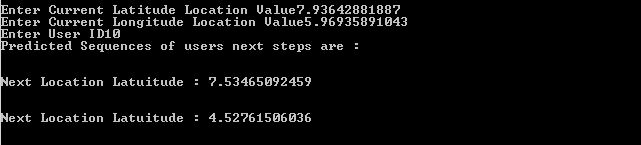
**Fig 8.2** LSTM model generation and the model attributes

The model generation is complete thus we train the model with the dataset that we have prepared which fits suitable for the project. The model is made up with encoders and decoders the idea behind the LSTM algorithm. The dataset in a csv file is passed on to the model and the list prepared in the beginning which consists of the values of dataset in a list format. All these resources are used in a processed way to train the model. And a certain number of epochs are run while training the model. The metrics for the epoch are also displayed accordingly. And certain metrics like accuracy and MSE values are calculated.



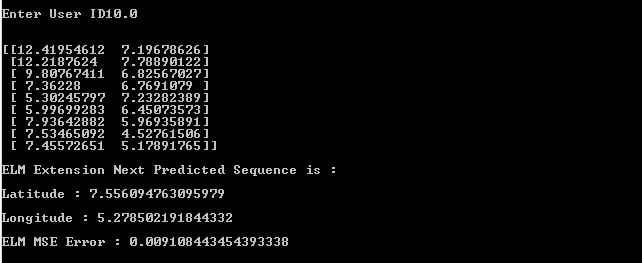
**Fig 8.3** Train the LSTM model with the epochs for better results

The next function is to predict the next location user using LSTM model created. The function requests for certain values of the user like longitude, latitude and user id. By providing those values to the model we predict the next location of the user. The model passes the data through the network created and predicts the next values.

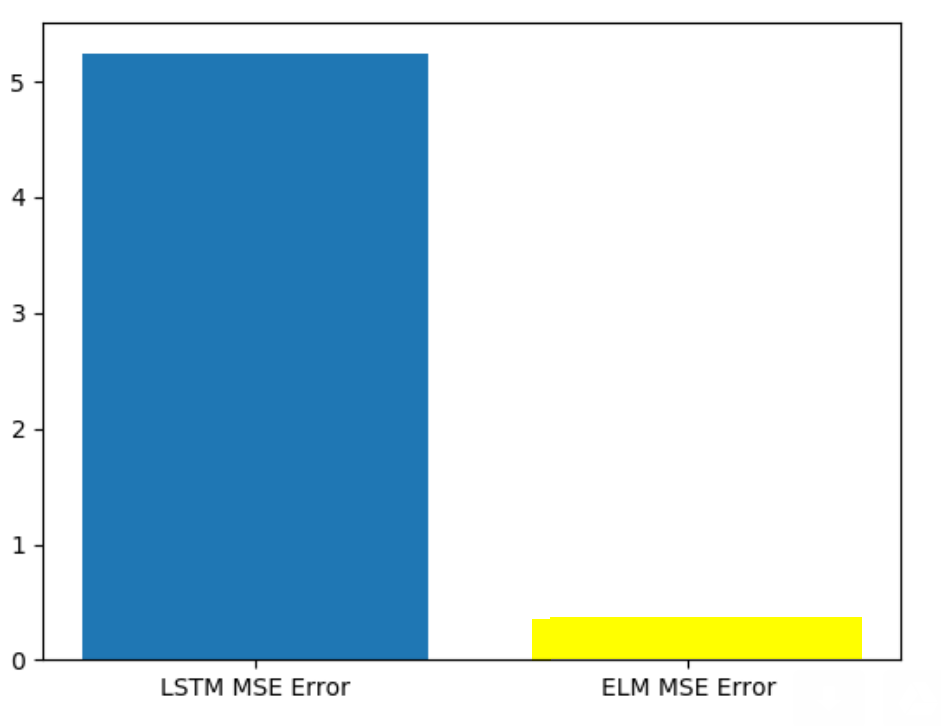


**Fig 8.4** Predicting user next location

The function we have invoked works on creating a model for EML regression algorithm. And this model helps us in predicting the next location of a particular user after all the user location points are trained. The model prepares a new location values which point to values that are potentially the next location after users last location. The function takes user id as input and training data is shown to know the latitude and longitude values. And displays finally calculated new location values.



**Fig 8.5** EML regression model and new location prediction



**Fig 8.6** Graphical representation of comparison

In above graph we can see with LSTM more prediction error is there on comparison with ELM. In above graph x-axis contains algorithm name and y-axis contains error rate. This graph proves that Extreme learning machine has an advantage over the LSTM.

1. **CONCLUSION AND FUTURE SCOPE**

Here, we investigate the significance of trajectory prediction and explore feasible approaches from both the single-user perspective and multi-user perspective. For single-user trajectory prediction, we propose a basic LSTM framework and experimental results on a model-based mobility dataset illustrate the superiority of LSTM to make predictions based on pre-learning of user-specific mobility patterns. For multi-user multi-step prediction, we further propose a region-oriented prediction scheme and put forward an LSTM-based Seq2Seq framework. Experiments on a realistic dataset show that the proposed framework outperforms the other competing approaches, which demonstrate its outstanding generalization ability for multi-user prediction as well as robustness and stability for multi-step prediction. Our current work does not consider the semantic context in the trajectory like the point of interests because of the limitation of data. For future work, we plan to combine our algorithm with some semantic information to improve the prediction performance.

Though, the aforementioned experimental results have shown the superiority of basic LSTM framework for learning user-speciﬁc mobility pattern, there still exist several issues to address for practical applications. First is the poor generalization ability of the proposed user speciﬁc mobility model. Usually, it is necessary to predict the trajectories of multi-users simultaneously in practical applications. Therefore, we need to train speciﬁc prediction models for each user of interest, which is not a sensible approach. On one hand, it incurs large computation overhead. On the other hand, training such a model usually needs a lot of historical data of the user, leading to cold start problem for users with insufﬁcient training data. Second is the error-accumulation effect for multi-step prediction. When the position measurements are suspended, the prediction is rapidly unable to follow the actual evolution of trajectory accurately, resulting in negative impacts on practical applications

1. **REFERENCES**
2. **CONFERENCE PROCEEDINGS**

* Z. Kai, S. Tarkoma, S. Liu, and H. Vo, “Urban *human mobility data mining: An overview*,” in IEEE International Conference on Big Data, **2017**.
* C. Yao, J. Guo, and C. Yang, “*Achieving high throughput with predictive resource allocation*,” in 2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP). IEEE, **2016, pp. 768–772.**
* B. D. Ziebart, A. L. Maas, A. K. Dey, and J. A. Bagnell, “Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior,” in Proceedings of the 10th International Conference on Ubiquitous Computing. ACM, **2008, pp. 322–331**.
* M. Morzy, “*Mining frequent trajectories of moving objects for location prediction*,” in Machine Learning and Data Mining in Pattern Recognition. Berlin, Heidelberg: Springer Berlin Heidelberg, **2007, pp. 667– 680**.
* A. Graves and N. Jaitly, “*Towards end-to-end speech recognition with recurrent neural networks,*” in International Conference on Machine Learning, **2014, pp. 1764–1772**.
* C. Zhang and P. Patras, “*Long-term mobile trafﬁc forecasting using deep spatio-temporal neural networks,*” in Proc. MOBIHOC 2018, Los Angeles, USA, **Jun. 2018**.
* X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO, “*Convolutional LSTM network: A machine learning approach for precipitation now casting,*” in Advances in Neural Information Processing Systems 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc**., 2015, pp. 802–810**.

1. **JOURNAL**

* S. Qiao, D. Shen, X. Wang, N. Han, and W. Zhu, “*A self-adaptive param- eter selection trajectory prediction approach via hidden markov models,*” IEEE Transactions on Intelligent Transportation Systems**, vol. 16, no. 1, pp. 284–296, Feb 2015**.
* Q. 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 Transactions on Vehicular Technology, **vol. 66, no. 6, pp. 5204– 5216, 2017**.
* S. B. Cho, “*Exploiting machine learning techniques for location recognition and prediction with smartphone logs,*” Neurocomputing, **vol. 176, pp. 98–106, 2016**.
* A. J. Smola and B. Sch¨ olkopf, *“A tutorial on support vector regression,*” Statistics and Computing, **vol. 14, no. 3, pp. 199–222, 2004**.
* A. A. Adebiyi, A. O. Adewumi, and C. K. Ayo, “*Comparison of ARIMA and artiﬁcial neural networks models for stock price prediction,”* Journal of Applied Mathematics**, vol. 2014, 2014.**
* H. Z. Moayedi and M. Masnadi-Shirazi, “*ARIMA model for network trafﬁc prediction and anomaly detection,*” in 2008 International Symposium on Information Technology**, vol. 4**. IEEE, **2008, pp. 1–6.**
* H. Yu and X. Zhu, “*Recurrent neural network based rule sequence model for statistical machine translation,*” in Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing(**Volume 2: Short Papers), vol. 2, 2015, pp. 132–138**.

1. **BOOKS**

* J. Neter, W. Wasserman, and M. H. Kutner, “*Applied linear regression models,”* **1989***.*

**APPENDIX A: UML DIAGRAMS**

**1. USE CASE DIAGRAM**:

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use case in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

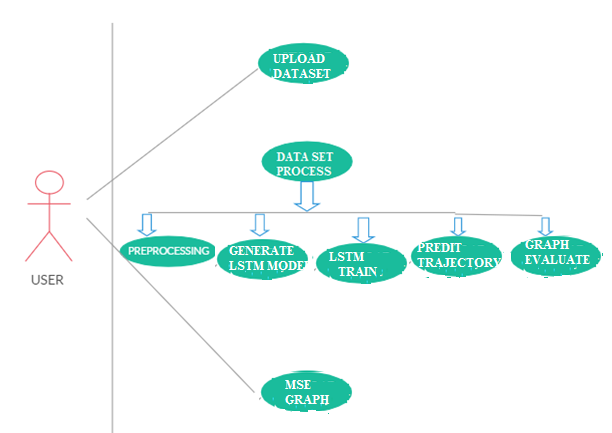
****

Fig. 9.1 Use Case Diagram

**2. CLASS DIAGRAM**

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and the constraints imposed on the system. The class diagrams are widely used in the modeling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

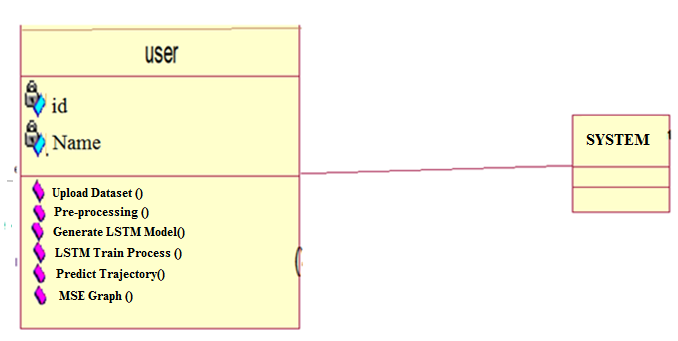
****

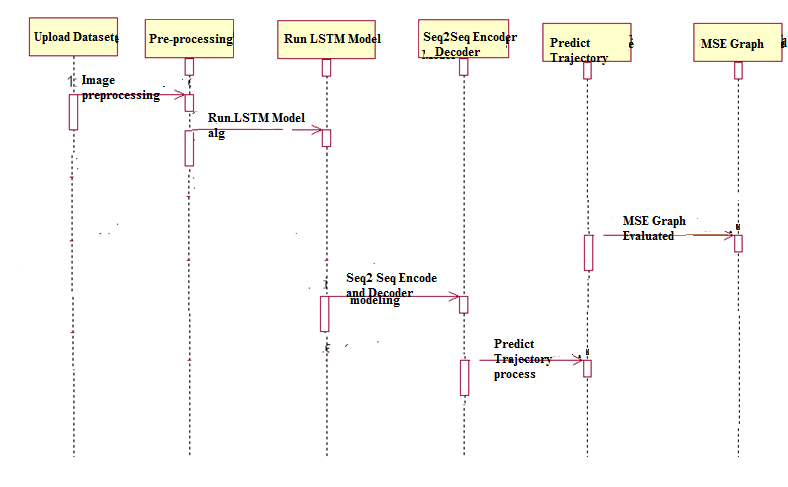
Fig. 9.2 Class Diagram

**3. SEQUENCE DIAGRAM**

A sequence diagram is an interaction diagram. From the name, the diagram deals with some sequences, which are the sequence of messages flowing from one object to another.

Interaction among the components of a system is very important from implementation and execution perspective. Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.

The below diagram represents the sequence of flow of actions in the system.

****

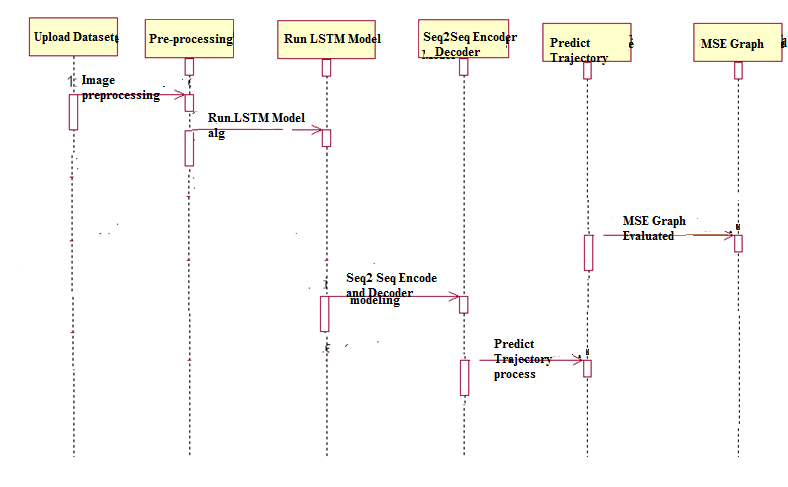
****

Fig. 9.3 Sequence Diagram

**4. ACTIVITY DIAGRAM:**

Activity diagram describes the flow of control in a system. It consists of activities and links. The flow can be sequential, concurrent, or branched.

Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.

Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed.

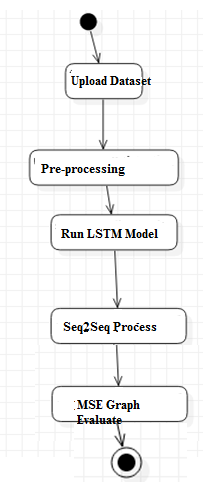


Fig. 9.4 Activity Diagram