



Master of Science in Information Systems

**Generating Synthetic Mobility Trajectories  
By Applying Machine Learning**

**By  
Yannick Patschke**

*Supervised by Prof. Benoît Garbinato  
and M.Sc Vaibhav Kulkarni*

January 2018



---

## Abstract

The purpose of this master thesis is to capture accurately human mobility behavior, in order to generate realistic synthetic mobility trajectories. Since many years an increasing number of applications have appeared about mobility data. However, these applications are struggling to exist mainly due to the need for large datasets and privacy restrictions of user data. Many mathematical models exist to describe or generate trajectories for different purposes, such as birds migration, predators behavior, and human mobility. They used a (random) stochastic process to create trajectories, but none of them clearly succeeded in capturing a realistic human behavior. The main issues with the trajectories created with these models are their huge boundaries or too random behavior. We choose to use machine learning models to extract information on real human trajectories, in order to obtain better results. Unlike the Markov processes, we focused on the idea that having some memory of past positions can be used to predict the following positions of a person. Indeed as movements of people go clearly often into one specific direction, it is interesting to know what were their last steps in order to better predict or generate their trajectories. Moreover, we simplified the problem by using areas of interest where people stay more time instead of geographical coordinates. The project was divided into three parts: prediction, optimization, generation.

The first part consists in comparing the performances to predict the next places of people of different machine learnings models, such as the Linear Regression, the Logistic Regression, the Neural Network and the Recurrent Neural Network. Among all models used, the Recurrent Neural Network (RNN) can capture long-term time dependency and seems to perform better even with basic parameters. The predicted trajectories with this model obtain better Mean, Median, Correlation, and P-value than all other models. !!!!!!!!!!!!!!!!

Thereafter the second part tried to fine-tune the Recurrent Neural Network by searching for the optimal hyperparameters. The maximum performances were reached by a RNN with 5 timesteps and 5 hidden layers with 50 neurons each. The predicted trajectories with this model obtain better Mean, Median and almost always better on other metrics, such as Standard Deviation, Maximum, Minimum, Correlation, and P-value.

The final part focused on generating trajectories. For that, we created and compared three different generators; Random Generator, Best Probability Generator and Random Choice Probability Generator. The Random Generator generate trajectories randomly based on a Markov process. The Best Probability Generator uses the RNN model to generate trajectories based on the predicted next position with the highest probability. The Random Choice Probability Generator interprets the output of the RNN as the probability of each possible next positions to generate trajectories. This last generator performs better to predict realistic human trajectories. Indeed its generated trajectories compared to the real one, are better than the first two generators on many measures, such as Mean, Median, Maximum, Minimum, Unique, Standard Deviation and Amplitude. Although none of these generators are yet prepared to be used in final applications, we show that generating synthetic traffic by learning the human mobility behavior through RNN is possible.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Motivation</b>	<b>7</b>
<b>3</b>	<b>Background</b>	<b>8</b>
3.1	Machine Learning . . . . .	8
3.1.1	Linear Regression . . . . .	9
3.1.2	Logistic Regression . . . . .	10
3.1.3	Neural Network . . . . .	10
3.1.4	Recurrent Neural Network . . . . .	12
3.1.5	Long Short-term Memory . . . . .	13
3.2	Tensorflow . . . . .	13
3.3	Cesium . . . . .	14
3.4	Dataset . . . . .	14
<b>4</b>	<b>Related Work</b>	<b>16</b>
<b>5</b>	<b>Methodology</b>	<b>18</b>
5.1	Prediction . . . . .	18
5.2	Optimization . . . . .	18
5.3	Generation . . . . .	18
<b>6</b>	<b>Prediction</b>	<b>19</b>
6.1	Data Pre-processing . . . . .	19
6.2	Prediction Performance . . . . .	19
6.2.1	Linear Regression . . . . .	20
6.2.2	Logistic Regression . . . . .	21
6.2.3	Neural Network . . . . .	21

6.2.4	Recurrent Neural Network - Long Short-term Memory . . . . .	22
6.3	Overview and Comparison . . . . .	23
<b>7</b>	<b>Optimization</b>	<b>24</b>
7.1	Hyperparameter Selection . . . . .	24
7.1.1	Hidden layer(s) . . . . .	25
7.1.2	Neurons . . . . .	26
7.1.3	Timesteps . . . . .	27
7.2	Hyperparameter Selection Values . . . . .	29
<b>8</b>	<b>Generation</b>	<b>30</b>
8.1	Random Generator . . . . .	30
8.2	Best Probability Generator . . . . .	31
8.3	Random Choice Probability Generator . . . . .	31
8.4	Performance and Comparison . . . . .	31
<b>9</b>	<b>Discussion</b>	<b>35</b>
<b>10</b>	<b>Conclusion</b>	<b>36</b>
<b>11</b>	<b>References</b>	<b>37</b>
	<b>List of Figures</b>	<b>39</b>
	<b>List of Tables</b>	<b>41</b>
<b>A</b>		
	<b>Software, libraries and tools used</b>	<b>42</b>

# 1 Introduction

Nowadays the mobility of people has clearly never been so high with so much means of locomotion. It is interesting to have the possibility to predict or even generate trajectories of people. There are several useful applications where predicting and generating trajectories can be integrated, such as traffic management, urban planning, consumer profiling. For now, in practice, it is still difficult to generate trajectories of people due to the complex behaviors that people exhibit. Generating trajectories of projectiles has already been done since many years with ballistic and even more complex trajectories such as fluid dynamics can be done with a known accuracy. However, generating trajectories for living entities becomes difficult. For instance, with insects, ants colonies are still studied to understand their behavior within a colony and to create better algorithms to solve the traveling salesman problem. So capturing real human behaviors and generating their trajectories is a challenge.

The approach chosen to generate human trajectories is to focus on the idea that information on past positions is a crucial characteristic. The trajectories of persons are mainly influenced by their starting locations and even more by their destination. Moreover, people nearly always try to avoid a long detour and go more or less straight to their destinations. It's why capturing human mobility behaviors will be possible only if we can integrate some part of their movement. We choose to divide the project into three parts: Prediction, Optimization, and Generation. The first part consisting in finding a model that performs well for predicting a person's next positions. For that, we compared four different models: Linear Regression model, Logistic Regression model, Neural Network (NN) model and Recurrent Neural Network - Long Short-term Memory (RNN-LSTM) model. The two first models are simple models, while the others two are more complex. These models were put in competition to predict the next positions of people based on their last position or until 5 of their last positions. Among all these models the RNN-LSTM model seems clearly more appropriate for the task [12]. Indeed its characteristics allow it to perform great where long-term dependencies are important. Moreover, the others models need to be adapted to capture long-term dependencies. The RNN-LSTM, even with very basic parameters, obtains better results than other models. Moreover, it can be improved by fine-tuning its parameter, which is clearly more complicated for the Linear and Logistic Regression models. The RNN-LSTM was kept for the next phase: Optimization. For this task, we search for the best hyperparameters to optimize the performances of the model. The tests are made on three parameters: number of hidden layers, number of neurons and number of timesteps. According to our test, the best number of hidden layers is 5 with 50 neurons each and 5 timesteps. Indeed the trajectories predicted seem clearly better. Moreover, some measures extracted from them and compared with real trajectories confirms it: the Mean, Median, Standard Deviation, Correlation and P-value are on average clearly in favor of these parameters. During the last phase, we define a way to generate different successive positions. We created three different generators; Random Generator, Best Probability Generator and Random Choice Probability Generator. The first one needs nothing to begin to generate trajectories. The two others need to have five last positions before beginning to generate trajectories based on the RNN-LSTM model trained before. The three results were compared; the Random Generator is not able to be used in the context of generating realistic human mobility. The other two perform well with few generate steps, but the Random Choice Probability Generator is better when generating long trajectories. Moreover, many measures confirm it on generated trajectories, such as Mean, Median, Unique, Standard Deviation and Skew.

Looking at all the results some obvious points can be found. For instance, during the prediction, we see that different machine learning models can be adapted to our project. However finding rapidly the best one is

preferable. Moreover, the thought about the importance of long-term dependencies is crucial in the context of mobility. It is why RNN models are predisposed to the task. The generators created are not yet usable in real applications. Nevertheless, we have proved that generating synthetic mobility trajectories by learning the human mobility behavior through machine learning is possible.

## 2 Motivation

The movements keep increasing in every field; human, material and information. Indeed the number of shiftings, transitions, and movements in our society has never been so high and continues to increase. For more than a century, we have not stopped seeing advances in many fields; mechanics, medical, computer science and many other sciences. Every time, these advances are reserved for a small number of people, but rapidly popularized and accessible to all. We can think of the first cars rapidly popularized with the *Ford T* by the brilliant *Henry Ford*. Especially since globalization, all new things have become even more rapidly accessible to the general public. So a lot of things that were one century ago reserved for rich or a particular sector, for instance, travel, are now common and widely widespread. So our technology and also our way of life, nowadays have particularly influenced movements in our society. They have really increased and promoted the rise of movements of people, and materials. It's really difficult to imagine the amount of data that we could collect just about daily movements of goods in a city, the number of letter ships or people moving to work. Now that some technologies like the GPS and connected device are so widely used and accessible, getting all these information about movements is possible at a technological level. All this different data about mobility can be useful in some ways. Thinking only about human mobility, we can already find a lot of fields where this data could be useful [12]:

**Traffic management:** We could analyze the movement of people to better know the most affected areas. We could think of a way to improve the fluidity or prevent traffic jam.

**Urban planning:** With the behavior of movement of people, we could predict the best areas to construct. We could increase existing area level for instance where construct a new mall to be accessible to as many people as possible.

**Consumer profiling:** We could try to learn behaviors of consumers to better understand their needs or find a better way to reach them.

The main problem is the need of a lot of data to obtain results which could make sense. It's not sufficient just to go and ask 100 people to give us their mobility data. As said before at a technological level we could obtain nearly all this data, but in practice, we are stopped by the amount of data. It's really difficult for somebody, a city, or an organization to obtain enough data in a field like mobility. Indeed, you have to ask people directly, which means to really ask a lot of people.

The second main problem will result from the free will of people. Not a lot of people will agree to give freely their mobility data. So there will always be the barrier of privacy. Rare are those who accept to give all their movements in life. Thinking of paying people to give us their traffic data is just not imaginable. If it was the case, more people would maybe participate, but the city or the organizations would need really a lot of funds. Fortunately, we live in a free country based on democracy, so trying to force people to give us the data or even find a way to collect their data without their agreement it also not possible. So based on all the above reasons, generating synthetic geospatial trajectories could be an interesting thing. Indeed it would allow us to need really a lot less data, which would mean also a lot fewer problems about collecting them.



## 3 Background

There are already some helpful knowledge and tools really useful for our project. This section shows some of them with their respective overview and their explanations. For instance, it focuses on a machine learning overview and some of its specific models related to our work. This section speaks also about some really interesting and relevant library existing or even the dataset used.

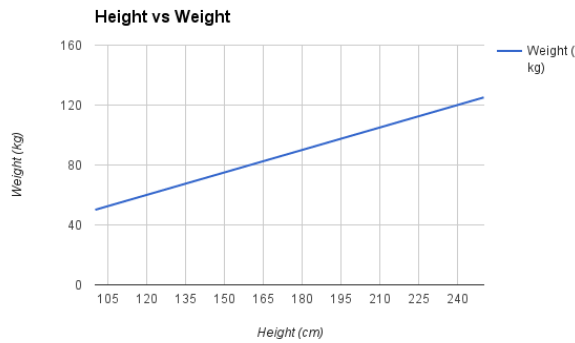
### 3.1 Machine Learning

Nowadays Machine learning is really widespread even if most people don't really know its utility and its possibilities. Machine learning is basically to teach the machine to act in situations that differ from what we have trained it. It's a process that first begins with a training phase where the machine trains with examples. Then the next phase is an evaluating process. Indeed we give the machine new but similar data and we look at how it processes it. Which means the machine tries to work on new data based on what it has learned previously with its trained data. Without going into details, nowadays we distinguish two major types of machine learning; supervised and unsupervised.

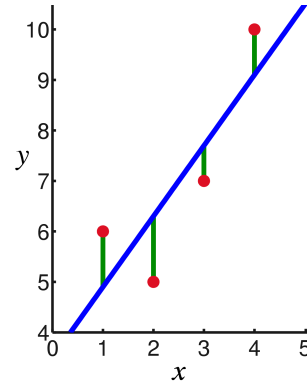
**Supervised machine learning:** The method allows us to train the machine on example data and its results. Basically, during supervised machine learning, we show the machine examples and their correct results several times. Until the machine has learned some patterns about the data and understood that if we give it an input  $A$  the correct answer will be  $B$ . For instance, we can make a connection with how a human kid learns to read. The kid will try to read some simple books several times until he correctly reads and articulates all words of these books. When he has correctly read all these books, we can assume that he understands how to read different words. Therefore we can assume that if we give him another book, he will manage to read this new book even if it's the first time. For the machine, it's similar; each time the machine sees the data example, it will try to remember what it has done good or bad. Then it will try to make better the next time. So we can say that supervised machine learning, mainly uses to understand the way to go from a point  $A$  (input data) to a point  $B$  (output data, the correct answer).

**Unsupervised machine learning:** This method works differently; this time the machine will just train on examples without feedback of what is true or false. It means without correct answers from the example data. Imagine again a child who is given a pile of marble to tidy up separately, but with no instruction about criteria to separate them. Maybe the child will separate them by color, size, weights or others, maybe mixed criteria. He will try to find a way; the machine does the same. The machine will, for example, try to identify some attributes about the data that are similar to some examples. It will try to find a way to group them together. Here it was an example of clustering. We can see that unsupervised machine learning works more to find some structure in the data given.

In our society, we can find traces of machine learning nearly everywhere and many times if we are attentive. For instance in our emails with the spam classification, with some social network with facial detection on pictures, with speech recognition for example from Siri of Apple, and so many other examples can be found. Moreover, machine learning will surely continue to extend in our society. Indeed these last decades have allowed really to see the power of computer increase. Now it allows processing a large amount of data faster than before. It makes it more interesting to use machine learning; indeed machine learning needs often to



**Figure 1: Sample of Linear Regression - Height vs Weight**  
Source: [3]



**Figure 2: Distance to square of a Linear Regression [21]**

work with a large amount of train data to be efficient. These last years have been really good for this field. We have seen more and more people using machine learning, not only in big organizations but also private people.

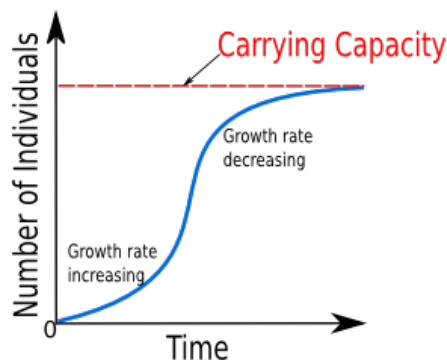
Indeed everybody doesn't need to have an expensive computer to make machine learning, but most of the personal computers can now handle machine learning in a great way. Moreover, a lot of courses have been created on the subject and a lot of services help us on machine learning, for instance: Tensorflow, Keras, Caffe, Torch, PyTorch, MXNet and others [20]. So now we can find easily what we need to work with machine learning. Some models are basic and well known while others are more complex. Now let's speak of some models that seem obvious, interesting or even promising: the Linear Regression Model, the Logistic Regression Model, the Neural Network Model, the Recurrent Neural Network and its special case with Long Short-term Memory.

### 3.1.1 Linear Regression

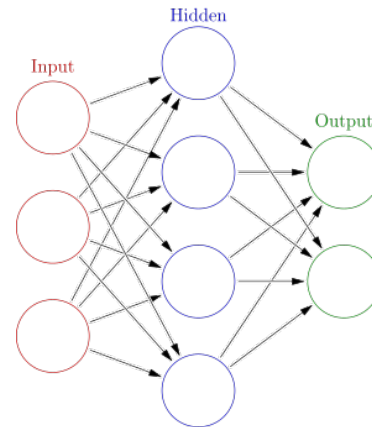
The linear regression model is clearly the most known model in statistics and machine learning. It's generally the first model that people begin to test in order to learn basic stuff in machine learning. So a linear regression model is basically a model which assumes a linear relationship (*See figure: 1*) between the input variables and the single output variables. The output can be calculated from a linear combination of the input variables. So basically the representation mathematic for a simple model is  $y = B0 + B1 \times x$ . Where  $x$  represents the input,  $y$  the output,  $B0$  and  $B1$  coefficients.

So using a linear regression model means estimating the values of the coefficients used in the equation with the data that is available. We use usually the Ordinary Least Squares procedure to seek to minimize the sum of the squared residuals. This means that given a regression line through the data we calculate the distance from each data point to the regression line (*See figure: 2*). Then we square distances, and sum all of them. This sum is what we try to minimize. In machine learning, this quantity is often called cost function.

So we optimize the values of the coefficients by iteratively minimizing the error of the model on our training data. This operation is called Gradient Descent and works by starting normally with random or zero values



**Figure 3: Logistic Population Growth [17]**



**Figure 4: Sample of a Neural Network (with 3 features, 4 neurons on one hidden layer and 2 results possible) [22]**

for each coefficient. The sum of the squared errors is calculated for each pair of input and output values. The coefficients are updated in the direction towards minimizing the errors. A learning rate is also used as a scale factor which determines the update level of the coefficients. The process is repeated until a minimum sum squared error seems to be obtained. So at that moment, we can say that the model has converged. Then basically we use the model with the coefficient updated and test it on test data and train data to see how it performs. In order to evaluate its performance, we can compare training accuracy and test accuracy of the model or even the cost function of the training part and the one from the test part. *(See [3] for more details about the model)*

### 3.1.2 Logistic Regression

The Logistic Regression model is similar to the previous Linear Regression Model. It is also one of the most famous models. It is based mainly on the logistic function, also called the sigmoid function  $1/(1 + e^{-value})$ . This function was named by *Pierre Franois Verhulst*, who first used it to represent the population growth, where the first stage of growth is exponential, then the growth slows, and finally stops *(See figure: 3)*.

So basically the function in our case can be represented as this  $y = e^{(b_0 + b_1 * x)} / (1 + e^{(b_0 + b_1 * x)})$ . Where  $y$  is the predicted output,  $b_0$  is the bias and  $b_1$  is the coefficient for the input value. So each feature of our input data has a coefficient that we need to determine with the train step as explained before for the linear regression model. So we try to optimize the cost function at each iteration and update the coefficients values. *(See [4] for more details about the model)*

### 3.1.3 Neural Network

The Neural network model is quite different. It's a more complex model and really an interesting one. The basic idea behind a Neural Network (NN) is to simulate interconnected brain cells inside a computer. So we can get the machine to learn things, recognize patterns, and make decisions in a humanlike way. The

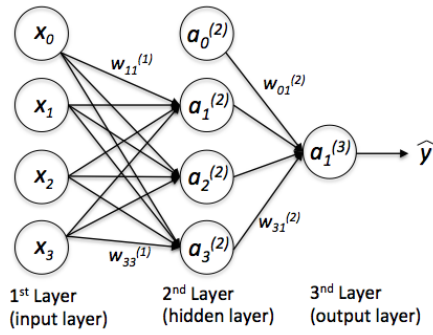


Figure 5: A Neural Network with weights [14]

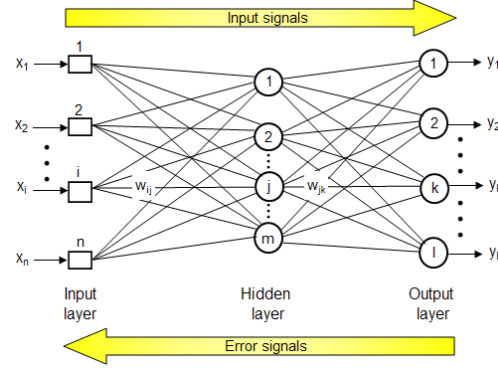


Figure 6: A Neural Network with feed forward and back propagation [5]

amazing thing about a neural network is that you don't have to program it to learn explicitly: it can learn by itself, like a brain! The Neural network can be composed of many cells that simulate the brain. The Neural Network can be separated into 3 different parts (*See figure: 4*):

1. Input layer: It contains input cells which are regrouped in one side of the network. These cells represent information from the outside world that the network will attempt to assimilate. It's basically all features and information that we want to learn about.
2. Output layer: It contains output cells which are on the opposite side of the network compared to those from the input layer. These cells can represent how the Neural Network responds to the information it has learned. We can see them as the result provided by the network after the data has gone throughout all of it.
3. Hidden layer(s): It is in the middle of the network. It can be composed of one or more layers which can each of them have a various number of cells usually called neurons. So these layer(s) and neuron(s) form the majority of the artificial brain. They actually influence directly the size and complexity of the network. For instance, a Neural Network with a large number of hidden layers and neurons compared to one with only one hidden layer and one neuron could, in theory, learn more complex things. It is as if we were comparing a human brain with the one of a mouse. Sadly in practice with machine learning, we can have some surprises. A small network could sometimes be better on complex data and in contrary, a complex network can be good for simple things. Indeed other parameters can influence the performance of the network on the data.

The connections between one cell and another are represented by a number called a weight (*See figure: 5*). Weights represent the influence that one unit has on another. So more the weights are high more the selected cell influences the cell linked. The Neural Network as the two last models learns iteratively. We give it the training data a certain number of times and it will each time go through the entire network. Each time data goes through the network, two steps occur. The first one is a basic method called feedforward. On this step, each unit receives inputs from the units on the left on which they are linked, and the inputs are multiplied by the weights of the connections in which their inputs go through. Every cell adds all the inputs it receives, computes some score and calls the next units it's connected to in order to give its results. The second step is where the backpropagation process occurs. It is a feedback process, in order to compare the

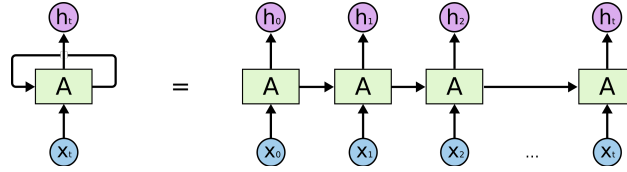


Figure 7: A Recurrent Neural Network [13]

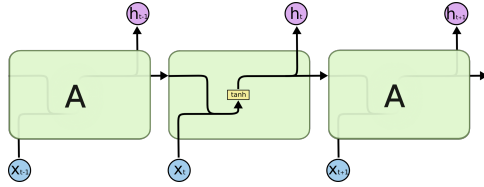


Figure 8: Simple RNN [13]

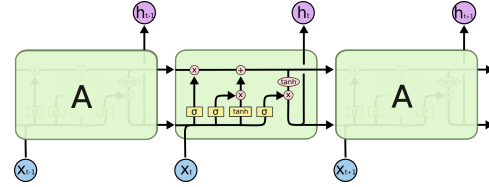


Figure 9: RNN-LSTM [13]

output produced with the real output: The NN uses the difference between them to modify the weights of the connections between the cells in the NN in order to be nearest to the good results the next time. (See [23] for more details about the model)

To summarize, during a simple cycle, the data goes through the network by passing by the corresponding cells and links. Then the model computes scores based on the information received and weight from the connections of its cells. Thereafter the model compares results that are given and the real results in order to correct itself (its weights). The purpose is to try to perform better the next time that we give it data. So this process is repeated a lot of times until it seems no longer able to improve.

### 3.1.4 Recurrent Neural Network

The model Recurrent Neural Network (RNN) works like an extension of a simple NN. RNN is similar to the NN while learning, except that it can learn and capture long-term dependencies also when we need to have some persistence of some past information. Indeed when we need to work with previous events, previous results found, previous data to predict the next one, it's where a Recurrent Neural Network becomes really useful. For instance, if we set a simple Neural Network and try to teach it to identify the nouns in a sentence by giving it only a word as input, it won't be able to perform well. Indeed a lot of information about the word given is in its context. So the only way to really capture and determine this information is by looking at its context, its sentences, which means words next to it. Therefore we need to look at the entire sequences or at least a part of it. Only then an RNN makes sense. We can see an RNN as multiple copies of the same Neural Network. These multiple copies of networks are linked one by one and each of them passing a message to its successor. So we have a persistence of some past events or some past results. (See [10] for more details about the model) So for instance basically in a context of sequences of numbers we give as input to the Recurrent Neural Network some of the past numbers. It tries to predict the next number. It does it by trying to learn the behavior of the sequences based on the input given and results it found before.

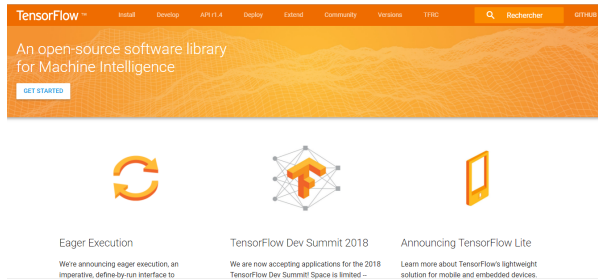


Figure 10: Tensorflow website [19]

```
# split data and get all input, output and number of example uses in the train
# here we cannot shuffle because we want to work with ordered past events
train_input, train_output, test_input, test_output, num_examples = sp.split(input, output, train_size, False)

# create tensor for my input and output ~ [batch-size, timestep, numberOfFeatures] and [batchsize, numberOfClasses]
X = tf.placeholder(tf.float32, [None, timestep, num_input])
Y = tf.placeholder(tf.float32, [None, num_classes])

# create multiple LSTMCells of same size (neurons) for each hidden layers that we want
rnn_layers = [tf.nn.rnn_cell.LSTMCell(n_neurons) for _ in range(n_hidden)]

# create a RNN cell composed sequentially of a number of RNNCells
multi_rnn_cell = tf.nn.rnn_cell.MultiRNNCell(rnn_layers)

# 'outputs' is a tensor of shape [batch_size, timestep, numberOfClasses]
# 'state' is a N-tuple where N is the number of LSTMCells containing a tf.contrib.rnn.LSTMStateTuple for each cell
outputs, state = tf.nn.dynamic_rnn(cell=multi_rnn_cell, inputs=X, dtype=tf.float32)
```

Figure 11: Some python code with Tensorflow on a multi-layers RNN-LSTM

### 3.1.5 Long Short-term Memory

The Recurrent Neural Network Long Short-Term Memory (RNN-LSTM) is a special kind of RNN, which is capable of learning really the long-term dependencies. The RNN-LSTM does have the ability to remove, select and add pertinent information from past events. So it can really retain important data from previous inputs and use that information to modify its current output. In contrary to a simple RNN which has less control over what kind of data from past events need to be retained an RNN-LSTM introduces a few more control steps. These control steps can be viewed as gates. These gates really control the access to the cell state and select which part of the data or previous data will be thrown away or kept from the cell state. So an RNN-LSTM has clearly a good selective memory even on long-term. It's also why the model is the main model of RNN used in practice. (*See [13] for more details about the model*)

## 3.2 Tensorflow

Building and using some models for machine learning from the scratch is totally possible. Indeed we only need some mathematics knowledge about it and some skills in programming to do it, but a lot of libraries exist already to optimize it as Torch or MXNet [20]. So we didn't really need to lose time by creating these models from nothing. Tensorflow [19] is one of these numerous libraries for machine learning and the one WE used for our project. This library was developed by Google. First, it was used for internal use in Google, but then it has been released under an open source license. So now it is a free and open source software library. Tensorflow allows us to work with numerical computation using data flow graphs. Without going into details Tensorflow allows working with a large number of models of machine learning or mathematics functions. Moreover, it is portable, the graph (so what we want to do) can be executed immediately or also saved to be used later or reuse. Further, it is flexible and can be run on multiple platforms: CPUs, GPUs, mobile and others. It can also be used with a different level of details. For example, on one side we can use the Keras part of Tensorflow like using machine learning with really fewer needs and knowledge about how works really machine learning, so less control but rapidly useable with pretty good results. On the other side, we can also set and configure all the steps of a model and clearly go into more details about a model. So Tensorflow is really optimal for different purposes whether they are complex or rapid and simple, or even whether the type of users with broad knowledge of machine learning or not.

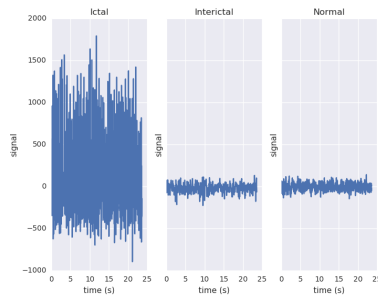


Figure 12: Plots [6] from EEG time series dataset [1]

```
from cesium import featurize
features_to_use = ["amplitude",
                  "percent_beyond_1_std",
                  "maximum",
                  "max_slope",
                  "median",
                  "median_absolute_deviation",
                  "percent_close_to_median",
                  "minimum",
                  "skew",
                  "std",
                  "weighted_average"]
fset_cesium = featurize.featurize_time_series(times=eeg["times"],
                                              values=eeg["measurements"],
                                              errors=None,
                                              features_to_use=features_to_use)
print(fset_cesium.head())
```

Figure 13: Python code [6] with Cesium to extract some selected features from a time series

### 3.3 Cesium

Cesium [6] is also a useful library, but in contrary to Tensorflow it doesn't work to properly run Machine Learning models. Indeed Cesium is useful in another specific field: time series. Indeed it can extract some really useful information about these kinds of data. It is optimal to extract different features about time series as mean, average, standard deviation but also clearly more complex one as skewness or amplitude of the dataset and a lot many other ones. Furthermore, Cesium can also be used to prepare a model of machine learning based on some of these features or even generate predictions for new data. Concerning its use Cesium works with Python which is a programming language. Moreover, this library is also an open source library. So basically we just need to import the library into Python script. Then through its methods for our case simply select some useful features to extract from the relevant data. So Cesium can be really helpful for some little analysis of time series.

### 3.4 Dataset

For the project, we have access to a dataset from Nokia. This dataset is interesting in priority because it has the position in the time of some users in their everyday life. A lot of other information was collected in this dataset and are also accessible for us, as for instance message sends, calls, battery level and so one. Moreover, this dataset is interesting especially to me with data about positions of our country and near our region. This dataset contains 150 users that received smartphones to use and keep all time from 2009 to 2011. Smartphones have collected all data possible, as SMS, phone call, network and many more data. It was always done preserving the confidentiality of the users.

The data that concerns us in priority was geolocation positions of the users in time. So basically the longitude, latitude and time per user. With this data, we can begin to use them and work on our subject. So after having looked at the datasets, WE needed to select a way to begin. Starting directly with positions (longitude, latitude) and time with machine learning is not necessarily the best way; it can be really complex and has bad results. Especially, if like me, this is the first time that we work on machine learning. So WE have first simplified the data, by taking only what we call points of interest. These are only some areas on a map where users stay more than a certain time. So the data on which WE work now is simply movements between these points of interest (POIs). So all this data are put in a sequence. We have then a sequence of indexes, where each index represents one of these zones where users stay more than a certain time. For

	A	B	C	D	E	F	G
1	0,"1, 2, 1, 2, 1, 2, 3, 1, 2, 1, 2, 1, 4, 1, 2, 3, 1, 2, 1, 5, 1, 2, 1, 2, 1, 2, 4, 3, 1, 2, 1, 4, 1, 9, 10, 11, 2, 7, 1						
2	1,"3, 4, 5, 7, 4, 8, 4, 9, 5, 4, 3, 11, 12, 4, 5, 4, 12, 13, 12, 11, 3, 4, 10, 5, 14, 5, 4, 9, 5, 15, 4, 16, 17, 4,						
3	2,"1, 2, 1, 3, 1, 2, 1, 6, 1, 7, 1, 2, 8, 2, 1, 9, 10, 9, 1, 11, 2, 12, 1, 2, 1, 2, 1, 2, 1, 2, 13, 14, 1, 2, 1, 2, 15,						
4	3,"1, 2, 1, 3, 1, 2, 1, 4, 5, 6, 2, 1, 2, 4, 2, 1, 2, 1, 2, 4, 2, 7, 1, 8, 1, 2, 3, 1, 4, 5, 2, 1, 2, 4, 2, 4, 1, 2,						
5	4,"2, 1, 2, 1, 3, 1, 3, 1, 3, 1, 4, 5, 1, 3, 7, 8, 1, 8, 1, 8, 3, 1, 3, 1, 10, 3, 1, 12, 3, 14, 15, 3, 1, 8, 1, 3, 1, 8,						
6	5,"1, 2, 1, 2, 1, 3, 1, 6, 1, 7, 1, 2, 4, 1, 2, 1, 2, 1, 2, 8, 2, 1, 2, 9, 1, 10, 1, 2, 1, 2, 1, 2, 1, 9, 1, 2, 8, 2, 9,						
7	6,"1, 2, 3, 4, 2, 5, 2, 6, 2, 7, 2, 8, 2, 3, 2, 10, 2, 3, 11, 3, 2, 3, 2, 3, 11, 3, 2, 3, 11, 3, 2, 3, 2, 12, 13, 12,						
8	7,"1, 2, 1, 2, 1, 2, 1, 6, 2, 1, 7, 8, 9, 1, 2, 10, 11, 2, 1, 2, 1, 2, 1, 13, 14, 1, 2, 10, 11, 2, 16, 1, 2, 17, 2, 1,						
9	8,"1, 2, 3, 1, 2, 4, 5, 6, 7, 9, 10, 1, 9, 1, 11, 1, 11, 12, 1, 2, 1, 11, 1, 11, 4, 13, 14, 8, 15, 1, 11, 16, 11, 1,						
10	9,"1, 2, 3, 2, 1, 3, 2, 1, 4, 5, 6, 4, 5, 4, 1, 2, 1, 8, 9, 8, 2, 10, 11, 1, 2, 3, 2, 1, 12, 3, 2, 1, 13, 4, 1, 2, 1, 5,						
11	10,"1, 2, 1, 3, 4, 3, 6, 1, 6, 7, 9, 7, 3, 1, 7, 10, 3, 1, 3, 6, 1, 12, 13, 3, 1, 2, 1, 14, 3, 6, 1, 3, 15, 3, 1, 6, 3,						
12	11,"1, 2, 3, 4, 3, 5, 6, 5, 3, 7, 3, 5, 3, 5, 8, 9, 3, 5, 10, 3, 11, 1, 12, 14, 3, 5, 16, 5, 17, 3, 5, 18, 5, 3, 5, 3,						
13	12,"1, 2, 3, 2, 5, 1, 2, 6, 1, 7, 1, 8, 1, 9, 10, 1, 2, 11, 1, 2, 1, 2, 1, 2, 1, 7, 1, 12, 1, 2, 1, 2, 1, 2,						

**Figure 14: Dataset where each row represents the movements of one user between POIs**

instance, the following sequence [0 1 0 2 3 .... ] will represent the movement of one user. First, he is in the area 0 then goes to 1. Next, he returns to 0, then goes to 2, and so one. In the further work, we can imagine maybe to reuse the first data (longitude, latitude, and time, and also maybe more feature if we think that can help our case) and try to perform better and have also better results.



## 4 Related Work

In the context of machine learning, cases and studies are very wide and numerous. Other people have already worked on a similar or at least a project with some concept in common with ours. So trying to discover and work all by ourselves without according regards to the works of others would be like wasting time. Indeed looking at models and researches of others made us win a lot of time to understand similar problems and how to handle them. Moreover, people have also worked on the generation of mobility trajectories, but not necessarily about human trajectories. Indeed Technitis Georgios and Weibel Robert [18] have worked on a generation of trajectories. Their work was to create an algorithm to generate random trajectories between two points for bird migrations. Their models work fine, their trajectories seem good, but still are mainly based on random. Indeed such models based on random give some interesting results when using some constraints and boundaries. For instance, some of the Random walk models give particularly good results for generalized patterns. Indeed the Levy walk model can be used in the case of ocean predators when they can't find food [8]. These predators abandon their previous pattern to follow a Levy walk model with a mix of long trajectories and short ones. Another work related to random which caught all our attention was the work of Bin Jiang and al [9]. They worked also on the subject of generation of mobility trajectories, but in this case about human trajectories. Their purpose was to show that the network of city influences clearly the behavior. Indeed they generated the mobility of a large number of random walkers which are forced to follow street network. Then they compared the results with real walkers and it seems that random walkers and real one have common behavior. So it seems that random can be applied in some specific cases of mobility field.

Another category of models, not necessarily involving random as the main principle to rely on, is mobility models based on some historical traces. For instance, Ghosh et al. [7] have worked also on finding a way to model human movement behavior, but without connection to random walk model. Indeed they did it by analyzing GPS traces in order to find some correlation between users, places or even a mix of both. Their model is able to categorize the different users based on their traces. It gives us also summaries of the trajectories per category of users. In a similar approach, the work of Kim et al. [11] is also interesting. Indeed they have also worked on extracting a mobility model based on GPS traces. They found a mobility model where the speed and pause time of the user's movements follow a log-normal distribution so that we can find some relationship between some users. The two above studies and works allow to find some common characteristics on users traces, but neither of them resolves the challenge of modeling human mobility for behaviors that are not part of the majority.

In the above models and techniques, the trajectories of the models can in someways stop suddenly or even turn very sharply. Indeed they can fail in really capturing the true movement behaviors of specific people, animal or even objects and can clearly have a mismatch with some particular type of behavior. Jean-Daniel et al. try to avoid a part of this problem of sudden change of trajectories proposed by the common model. They use an Enhanced Gauss-Markov (EGM) mobility model [2]. Indeed they try to limit some sudden stops and same cycles within specific regions. So their model ensures nearly only smooth trajectories. Among all these works we see that it is possible to cluster some behaviors based on their similitudes and even to obtain some smooth trajectories. Some facts occur with these models; they are based on stochastic process and have not clearly captured human realistic mobility characteristics. Moreover, they don't use some relation to past locations as we want to do. So all this consolidates the thought that finding a global model that can work on common behaviors and uncommon ones is quite challenging and yet not found. Furthermore,

it pushes us, even more, to try to go further, to really try to capture the behavior of people and generate trajectories. We tried to do it without a goal-oriented to prove some facts about society or human behavior as some of these works have done, and mostly without setting restrictive limits on locations.

## 5 Methodology

Every project needs to have a plan to follow or to try to follow. So to achieve the project of creating generating synthetic mobility trajectories, we divided it into three big phases: Prediction, Optimization, and Generation.

### 5.1 Prediction

The first and important part of the project is to find a model that predicts the trajectories of users based on their previous position(s). For this, we need to check and find a model that performs well and better than a classic model as logistic, linear model or random model. We need to try, execute and compare different models. See how they perform on the dataset available, check where and maybe why one model performs better than others. We need also to try to find their limits. When we are confident in a model and its performances, the next step can begin.

### 5.2 Optimization

After having selected a model on which work, the next step is now to begin to optimize and deepen the model selected; begin to switch and compare parameters. We need to find an adequate learning rate, a number of epochs during the training (number of time that our train data will pass in our model), number of neurons or the number of hidden layers for a neural network, and so one. We need also to check the best way to give it data and try to go to the limit of the model with the dataset available. So a big part of testing, switching some little things, comparing them, and trying to reach conclusions and some results to finally find the best parameters for a model on our data.

### 5.3 Generation

The last part of the project is about generating data based on the model of predictions selected and optimized. So for example by giving a program a starting position and letting the model give the next positions based on how it has predicted it. It is a phase of generating and evaluating the data generated to see if it confirms what we expect and if it seems near the human behavior observed in our datasets. Finally, this phase gives some conclusions about the different methods find for generating synthetic mobility trajectories.

## 6 Prediction

This first part of the project was to create different models of machine learning in order to predict the next positions of users based on their last position or some of their last positions. Then we compared these different models to finally be able to argue in favor of one of them for the next steps of the project. For all that we first began with some data pre-processing to be able to be used in a good way with all our future models.

### 6.1 Data Pre-processing

As described in the section *Dataset*, we have worked on a simplification of the problem with sequences of points of interest (that we called POIs). So the first thing was to extract all transitions from users in shape of sequences of POIs. So having all data as `[0 1 0 2 3 .... ]`. Then to rework them to have an easier and better shape to work on machine learning, so according to some readings, we choose to represent all positions by a vector binary which represents all POIs possible. We obtain a vector of a shape (1,238) because we identify 238 different areas where users stay more than a certain time. So a position can be represented by a vector of 238 binaries where only one of them will be equal to 1 and will be the index of the position listed.

### 6.2 Prediction Performance

So now let's speak about the results of the models used. The models were set up to try to predict the next positions of a user based on their last position or some of their past positions. First, all models were set with the same simple parameters. So we set the learning rate, the batch size, the number of epochs, the training size, the input, the output, the number of timesteps, the number of hidden layers, and the number of neurons. In our case the parameters are defined and first set as following:

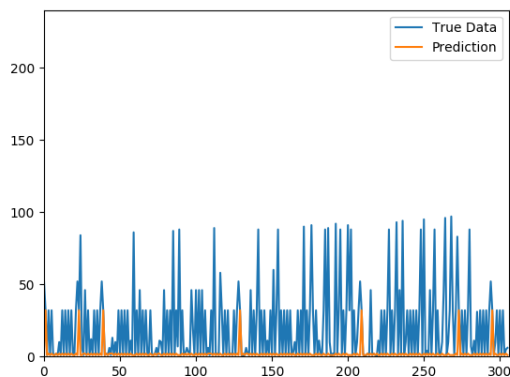
**Learning rate** = 0.0001: It can be represented as the speed that the model tries to correct its prediction between training cycles. On one hand, higher the learning rate is, faster the model will change and adapt its prediction, but it will also maybe really put the model in error by going too fast. On the other hand, smaller is the learning rate, slower the model will converge near its best value, but the model will be less likely to miss its apprenticeship and be in error.

**Batch size** = 20: It corresponds to the number of input data given to the machine at the same time during training the sessions. So if we set it to 20 and we have 100 input data of training, we will have 5 small cycles of training (one for each batch) in order to complete one entire cycle of training (named epoch).

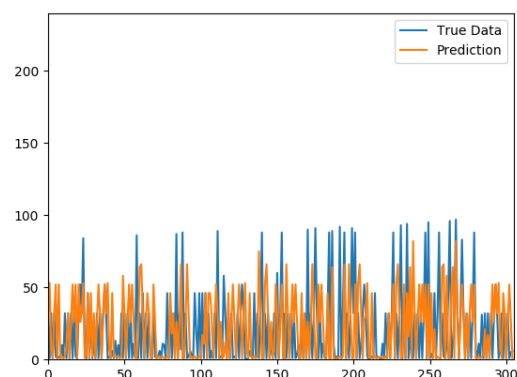
**Number of epochs** = 1000: One epoch represents a complete cycle of training. So when the machine has done one epoch, it has received one time all training data on which it has learned. So indeed the number of epochs corresponds to the number of time all training data will pass through the training.

**Training size** = 0.7: It is the percentage  $p$  of all data that will be used for training. So  $1 - p$  will be the percentage of data that will be used for testing.

**Input** : It's a vector of 238 binaries which represents all the index of POIs of users. So for one current position, only one of these binaries is equal to 1, which corresponds to the index of the position.



**Figure 15: Linear Regression model: True movement vs Prediction with 1 timestep and parameters from table 1**



**Figure 16: Linear Regression model: True movement vs Prediction with 5 timestep and from table 1**

User	Learning rate	Batch size	Epoch	Training size
24	0.0001	20	1000	0.7

**Table 1: Settings of simple models during prediction**

**Output** : It's also a vector but instead of 238 binaries, it is of 238 probabilities. So it corresponds to the probabilities of each of this 238 POIs to be the next position. Obviously, it doesn't give directly the next position, but we can then take the maximum probability obtained from these different POIs and say that the model predicts this one as next position.

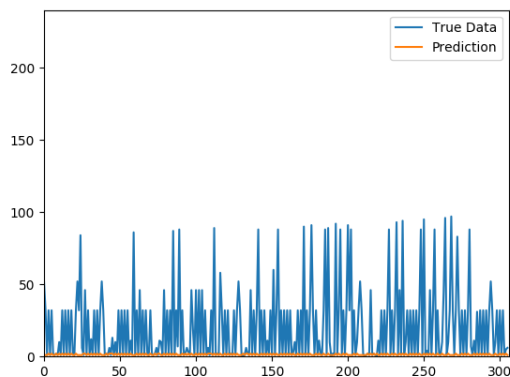
**Timesteps** 1 until 5: It corresponds to the number of previous positions that will be taken into account to predict the next position. So it represents the history of the user movements that will be used to predict his next position. For instance, a timestep set to 2 corresponds to our model to predict the next positions based on the two last positions.

**Hidden layers** = 1: It is the number of hidden layers, so it can represent the depth of the brain, its size or its complexity.

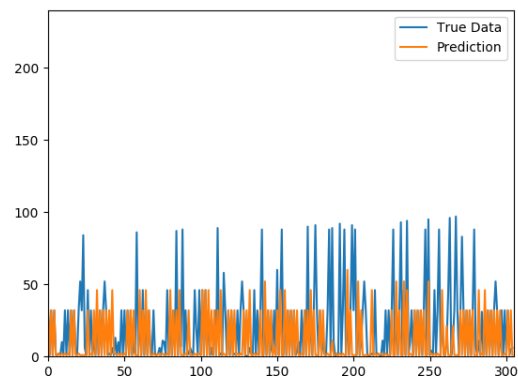
**Neurons** = 20: It is the number of neurons per hidden layers. Where each neuron tries to compute a score based on the weights of their relations between them and other cells. So more neurons mean more things trying to be captured by the model on the training data (but doesn't mean that it will capture useful things).

### 6.2.1 Linear Regression

The Linear model was really a basic model. So we didn't expect it to work really well. The first result confirmed our thoughts. It was with only one timestep, so by trying to predict the next position with only the last position of the user. It seems that the movement predicted was clearly not good. It didn't really fit with the true trajectories linked (*See figure: 15*). So then we tried with more timesteps until 5. It seemed



**Figure 17: Logistic Regression model: True movement vs Prediction with 1 timestep and parameters from table 1**



**Figure 18: Logistic Regression model: True movement vs Prediction with 5 timestep and parameters from table 1**

User	Learning rate	Batch size	Epoch	Training size	Hidden layer	Neuron
24	0.0001	20	1000	0.7	1	20

**Table 2: Settings of complex models during prediction**

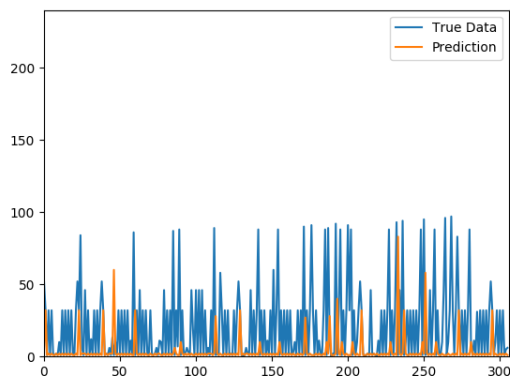
quickly to improve the performance of the predicted positions, even if we used a linear model (*See figure: 16*). These first graphs and results seemed finally not so bad, but to perform better and improve this model, it would be very hard. Indeed this model is a very simple one, without much parameters to change or adapt. This model shows us that even now we can see that some timesteps can really improve results, but it can't really improve the capturing of a human movements behavior. So the need for a model that can capture more information and perform better is required.

### 6.2.2 Logistic Regression

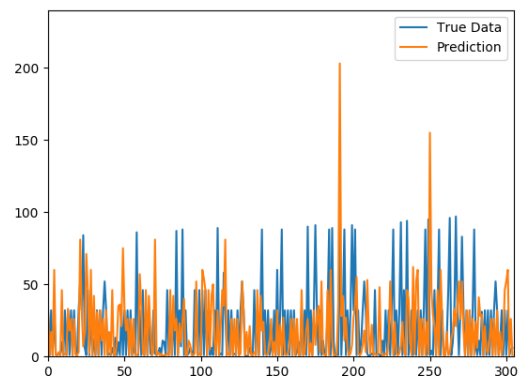
The Logistic model was also quite a basic model. So like with the linear model we didn't expect much of it and the results with only one timestep also proved it to us (*See figure: 17*). The same effect occurs as with the linear model. Indeed the model shows us that having more timesteps is already better (*See figure: 18*), but really work with it will be hard. Its results seem too much mechanical and without the human touch and as with the Linear Regression model it will be hard to really improve it, because of its lack of parameters and flexibility. So it seems that we need a model more complex, flexible and that we can really optimize.

### 6.2.3 Neural Network

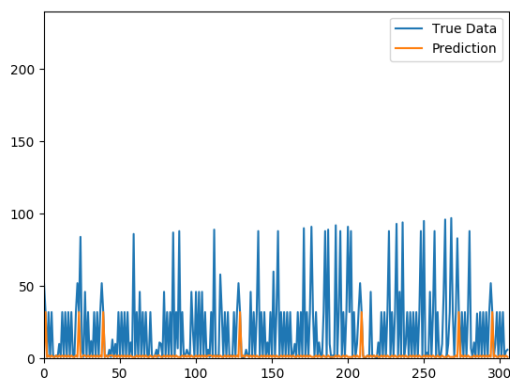
First, the neural network was also disappointing and upset us with its first results with only one timestep (*See figure: 19*). It seems also not to be really accurate as the basic models (linear and logistic) for prediction. Then after a second look to really check with more timesteps (*See figure: 20*), we see that one more time the predictions seems really more near a human behavior. We can see that even with some basic settings the



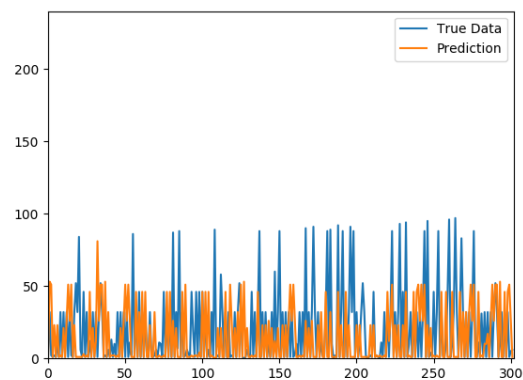
**Figure 19: Neural Network model: True movement vs Prediction with 1 timestep and parameters from table 2**



**Figure 20: Neural Network model: True movement vs Prediction with 5 timestep and parameters from table 2**



**Figure 21: RNN-LSTM model: True movement vs Prediction with 1 timestep and set of parameters 3 from table 2**



**Figure 22: RNN-LSTM model: True movement vs Prediction with 5 timestep and set of parameters 4 from table 2**

model is already trying to catch a more realistic behavior of movement than a simple model as the Linear and the Logistic.

#### 6.2.4 Recurrent Neural Network - Long Short-term Memory

For the RNN-LSTM model, nearly the same effect occurs when we look at trajectory predicted by the model (*See figures: 21 and 22*). After having seen the real behavior and compared it with one of the basic models, we can only comfort our first thought; so a more complex model will help us to have better results. Indeed we will be able to improve the model because it has now only very simple parameters (1 hidden layers and 20 neurons only). Even more in the next phases; we tried to consolidate even more the fact that having some memory of past information could really help us to have better results, even if we saw it already a little bit.

### 6.3 Overview and Comparison

Indeed in all models used, it seems that they capture some information and behavior from the data given to them, but only the Neural Network and the Recurrent Neural Network seem to give us something near the reality more often than basic models. It will be possible to perform better by trying to change some of the parameters and find maybe a better way for all these models before choosing one, but already with some tricky parameters, we can see that results lean in favor of a more complex model than the basic ones. Moreover as said on the basic models they have fewer parameters to really improve results. Indeed with parameters of the Logistic Regression model and Linear Regression model we can only try to change some basic stuff for their training as the number of epochs, the learning rates, so without really improving or really changing them; it will be as trying to improve a model that is already near its full capacity, so very complicated. In contrary to the NN and RNN-LSTM which can really be modified by changing their complexity by changing their number of neurons or even their number of hidden layers, so basically their size and power. It's why we choose not to spend more time with a basic model and try next to optimize the Recurrent Neural Network: our RNN-LSTM.

The Neural Network model seemed also really promising, but it is just because we tried to simulate the potential of the RNN-LSTM as with the basic models. Especially we tried to simulate timesteps for having some memory of past events. So trying to continue with the NN was just to simulate the work of the LSTM. It was not really useful, and even more, the way the model received data to simulate some timesteps was only time-consuming during training sessions. Indeed the NN and the two basic models are not able to keep a memory of past information or results. To simulate this memory we simply increased their input data given, so finally, they had really larger inputs when working with more timesteps. Indeed when the RNN-LSTM has always a simple input of 238 binaries, the other models had an input of 238 binaries multiplied by the number of timesteps, where each slice of 238 binaries represents one of the last positions. So with this practice, it simulates timesteps of the RNN-LSTM and by the way, also has inputs clearly larger. So the time that these models take to process is also higher. So it's also why we choose to clearly continue with only the RNN-LSTM.



## 7 Optimization

The next big phase was to try to find the best or at least good parameters for our models, parameters that could give me interesting results. So set and test some parameters then try to change them. For this, we tried to select some different range of values for our parameters and tried to compare their results. For instance, we choose to set some parameters as 5 hidden layers, save the results, and then try to increase this number of layers. If the new result was better than the last one, in the next step we continued to increase it, but on the contrary, if the results were worse, we decreased the parameters. And then the same process was done for some of the other parameters.

It was clear that this task could take a lot of time if knowing that there are 150 data's users to train, so as many models to train. Having time and computer power limited, we choose to only train and make results for a part of our data. We choose to take only 25 users with more data, so in our case more movements between POIs. We choose this number of users because we assumed that users with more data, with more transitions between POIs, would be better than those with few transitions. Indeed one fact was that some users had really few data, so few transitions between POIs. So using a big model of machine learning on them as LSTM would have been absurd knowing that this type of machine learning needs normally a lot of data to be correctly trained. The second fact was that users with more movements would have also normally more transitions between different POIs. So it would represent globally better our users and their areas visited.

We used our models with different parameters on these users. Thereafter we obtained also different results for each of our 25 users selected. So we needed to find a way to say if the model performed good or bad or at least find out if with these parameters it performed better than with other parameters. So we choose two simple ways to evaluate the results for each set of parameters:

1. Manually by simply looking at the movements predicted for each user and comparing them with some real one.
2. Statistically by selecting some measures that seemed interesting. So these measures were extracted on the movement predicted and the real one, then compared with each other. So we choose the parameter that seemed to have measures on average nearest to those from true data.

These two processes are not really purely scientific and are also based on our own opinion, but it was easier and a fast way to choose between parameters.

### 7.1 Hyperparameter Selection

Concerning our parameters in comparison with the ones in the section *Prediction*, we just increased the batch size and the number of epochs, because now we wanted to train the machine with only users that had more transition, so more data. Moreover, we used more complex model with higher parameters, so it would need more training cycles to capture the behavior. Furthermore having more training cycles (epochs) would drastically increase the time spent on training, so we choose to increase the batch size that would speed up the process without having really negative effects on the results. It would maybe even be better with a higher

Learning rate	Batch size	Epoch	Training size
0.0001	100	5000	0.7

**Table 3: Basic settings during optimization**

batch size. Indeed according to some forum and discussion read; higher the batch is, better is the accuracy of the estimate of the gradient. Having only a batch size of 20 as before was clearly only to process on users with really few transition, but now we restricted our job to only users with more transitions. So the batch size could be higher easily. So we set first some basic parameters (*See table 3*). Then as said some measures were taken into consideration to compare the results of our different settings. These measures were extracted from the predicted path of our models. So the path consists of each point predicted by the model based on the last real position or some of the last real positions depending on the number of timesteps. Moreover, most of these measures were extracted from the library *Cesium* as spoken in the section *Background*.

**Mean:** Mean of observed values.

**Median:** Median of observed values.

**Maximum:** Maximum observed value.

**Minimum:** Minimum observed value.

**Unique:** Total number of unique observed values. [15]

**Std:** Standard deviation of observed values.

**Skew:** Skewness of a dataset. Approximately 0 for Gaussian data.

**Amplitude:** Half the difference between the maximum and minimum magnitude.

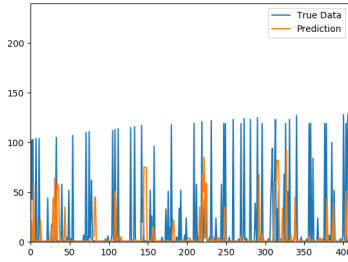
**Max slope:** Compute the largest rate of change in the observed data.

**Correlation:** The Pearson correlation coefficient measures the linear relationship between two datasets. [16]

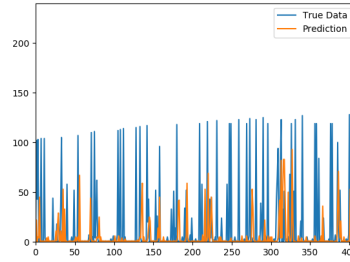
**P-value:** The p-value indicates the probability of an uncorrelated system between two datasets. [16]

### 7.1.1 Hidden layer(s)

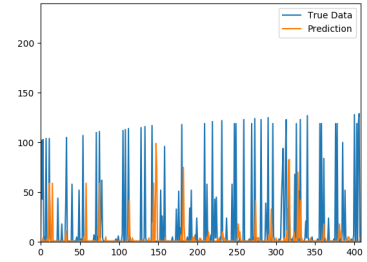
First, the tests were to find the best number of hidden layers so we tried different numbers: 1,3,5,7,10 and 20. The result was pretty obvious that 5 hidden layers were clearly better than less or more layers if the model have 50 neurons per hidden layers and timesteps of 5. We can see some features extracted from generating path predicted (point by point) of our different results and the real mean based on true movement from our users in the following table. We see clearly that 5 hidden layers perform nearly always a little bit better than the others. Especially when we look to the *Mean* and *Median* which is with *Correlation* and *P-value* the measures that we look in priority. Only for the *Correlation* and *Skew* the model with 5 hidden layers was not the best, but so nearly from the best results (*See table: 4*).



**Figure 23:** True movement vs Prediction (1 hidden layer)



**Figure 24:** True movement vs Prediction (5 hidden layers)



**Figure 25:** True movement vs Prediction (10 hidden layers)

	1 HL	3 HL	5 HL	7HL	10 HL	20 HL	Real
<b>Mean</b>	15.003	16.361	17.843	17.175	14.21	5.426	22.012
<b>Median</b>	5.86	8.32	8.48	7.64	8.0	2.82	9.52
<b>Maximum</b>	106.56	108.72	110.2	106.76	104.76	76.24	133.12
<b>Minimum</b>	0.2	0.16	0.16	0.2	0.2	0.32	0.12
<b>Unique</b>	27.68	33.64	36.2	34.8	25.72	9.36	47.08
<b>Std</b>	22.158	23.982	25.169	24.334	21.54	10.134	33.279
<b>Skew</b>	2.842	2.598	2.489	2.478	3.189	5.753	2.226
<b>Amplitude</b>	53.18	54.28	55.02	53.28	52.28	37.96	66.5
<b>Max slope</b>	104.24	106.36	107.6	104.44	102.2	74.12	130.48
<b>Correlation</b>	0.082	0.06	0.056	0.044	0.023	0.013	1.0
<b>P-value</b>	0.328	0.335	0.315	0.435	0.457	0.384	0.0

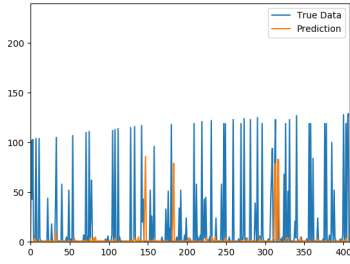
**Table 4:** Some measures from different number of hidden layers and true movement

Then for instance with the following graphs that compare movement generated point by points, it seems that 5 hidden layers are better than a unique hidden layer. It is also clear that the model with 5 hidden layers tries to capture more transitions and clearly more different positions (*See figures: 23 and 24*).

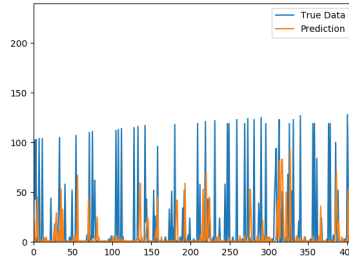
Then we can also see that more hidden layers decrease a lot the results (*See figure: 25*). It seems that the model with too many hidden layers begins to only predict the POIs with more occurrences. We can presume different effects. Maybe there are too few training cycles or even that the model is too complex for this data as we try to calculate a simple addition with a powerful computer while a simple calculator can do the job. However, all these results seem to be in favor of 5 hidden layers. So we kept these parameters for the following.

### 7.1.2 Neurons

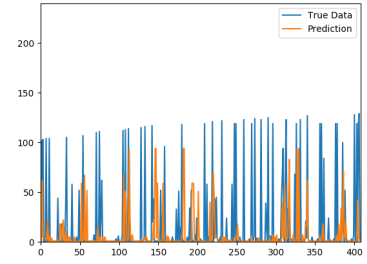
So then the next tests were to find a good number of neurons per layer. It was set to 50 during the last tests, so then we tried to increase, decrease and compare them: 10,20,50,70,100 and 150 neurons per layers. So we found that the best number of neurons was approximately 50 with 5 hidden layers and 5 timesteps.



**Figure 26: True movement vs Prediction with 10 neurons**



**Figure 27: True movement vs Prediction with 50 neurons**



**Figure 28: True movement vs Prediction with 100 neurons**

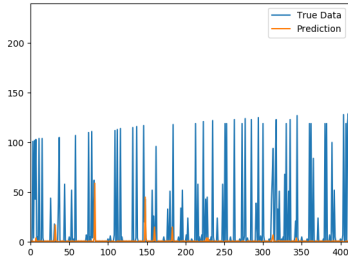
	10 N	20 N	50 N	70 N	100 N	150 N	Real
<b>Mean</b>	10.151	14.435	17.843	17.417	17.655	17.398	22.012
<b>Median</b>	5.28	7.88	8.48	4.92	5.86	8.3	9.52
<b>Maximum</b>	91.08	103.56	110.2	110.28	111.6	121.72	133.12
<b>Minimum</b>	0.28	0.24	0.16	0.16	0.2	0.16	0.12
<b>Unique</b>	11.08	21.44	36.2	38.28	36.0	36.6	47.08
<b>Std</b>	15.604	21.742	25.169	25.22	25.492	25.142	33.279
<b>Skew</b>	5.071	3.198	2.489	2.378	2.456	3.175	2.226
<b>Amplitude</b>	45.4	51.66	55.02	55.06	55.7	60.78	66.5
<b>Max slope</b>	88.84	100.2	107.6	106.2	109.96	119.44	130.48
<b>Correlation</b>	0.026	0.039	0.056	0.047	0.041	0.031	1.0
<b>P-value</b>	0.448	0.454	0.315	0.46	0.45	0.417	0.0

**Table 5: Some measures from different number of neurons and true movement**

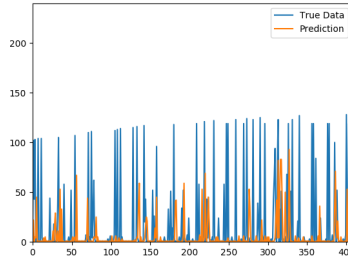
We can see it here that too few neurons per hidden layer will clearly decrease results (*See table: 5*). In the same way, too many neurons will not really improve results and will just increase drastically the training time. Based on the measures of this table we can assume that 50 neurons are the best choice. Indeed the model with 50 neurons per layers obtains the best results for 5 of these measures; Mean, Median, Minimum, Correlation and P-value. Moreover, this model never has a measure with really bad results comparing to other models. When it performs less well than another model, it is always really near to the best value obtained by the best model. Furthermore looking at the movement predicted compared to a real one, it confirms clearly that taking fewer neurons than 50 is a bad choice. In the same way, it confirms that too many neurons will not really improve results and as said will just increase the training time. So we set neurons per hidden layers to 50 for the following.

### 7.1.3 Timesteps

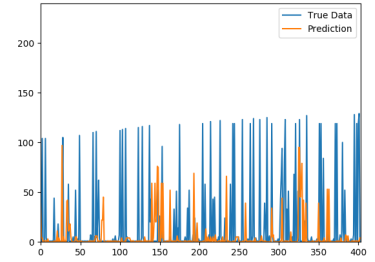
Then the last set of tests was to find the best timesteps, so the best memory of last positions to take into account for predictions. So we tried different timesteps with our previous best parameters: 1,3,5,7 and 10 timesteps. It resulted that 5 timesteps are better with our measures. Indeed for this particular user, the model with more timesteps seems good, but when we look at our measures we see clearly the superiority of



**Figure 29:** True movement vs Prediction with 1 timestep



**Figure 30:** True movement vs Prediction with 5 timesteps



**Figure 31:** True movement vs Prediction with 10 timestep

	1 T	3 T	5 T	7 T	10 T	Real
Mean	4.068	13.565	17.843	17.47	17.692	22.012
Median	1.96	3.38	8.48	5.64	5.36	9.52
Maximum	62.2	107.12	110.2	108.48	110.68	133.12
Minimum	0.28	0.16	0.16	0.16	0.2	0.12
Unique	6.44	27.72	36.2	38.36	36.32	47.08
Std	7.008	22.41	25.169	25.47	25.896	33.279
Skew	6.444	3.097	2.489	2.318	2.296	2.226
Amplitude	30.96	53.48	55.02	54.16	55.24	66.5
Max slope	60.68	105.16	107.6	106.88	108.2	130.48
Correlation	0.034	0.079	0.056	0.01	0.017	1.0
P-value	0.342	0.282	0.315	0.565	0.494	0.0

**Table 6:** Some measures from different number of timesteps and true movement

having 5 timesteps (*See table: 6*).

These measures show that increasing the number of timesteps will first rapidly increase nearly all measures until 5 timesteps. After that, if we continue to increase the number of timesteps we can see nearly no more improving. Moreover, it can result by even become worse for some measures as the median compared with the real one. So here it clearly is better to choose the model with 5 timesteps. Indeed this model performs clearly better with the median than any other models. Moreover, all its other measures are always near the best value and never clearly beaten by another model. Furthermore looking at the movement predicted compared to a real one, it confirms clearly that taking fewer timesteps than 5 is a really bad choice. It is less obvious that too many timesteps are bad with the same type of graph, but we can still conclude that more timesteps don't really improve the results. So it's why we set the timesteps to 5 for the following of the project.

Learning rate	Batch size	Epoch	Training size	Hidden layer	Neuron	Timestep
0.0001	100	5000	0.7	5	50	5

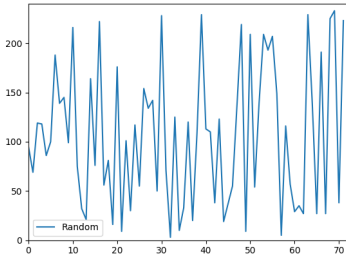
Table 7: Best parameters found

## 7.2 Hyperparameter Selection Values

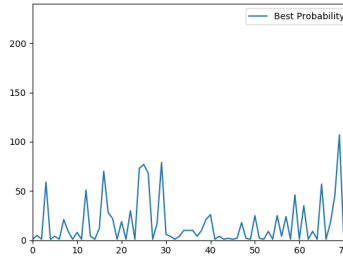
All these results show us that it can be really tricky to find the best parameters. Moreover, that trying all combinations of parameters can't be done without a lot of time spent on training on the machine. So we need to simplify the problem. We also saw that some parameters obtain better results on some measures than our final selected parameters but on average worse. Indeed machine learning as RNN can be really complicated and the results are sometimes surprising. It's really hard to know exactly why some parameters work better on some measures than other parameters. With our tests, we can still conclude some points:

1. Some hidden layers can capture better long-term dependencies. Indeed 5 hidden layers perform well, especially to approach the real mean and median.
2. Some neurons also perform better than too few. Indeed near 50 neurons, it seems to obtain the best results, especially for the median, mean and P-value.
3. Some timesteps will really try to capture and understand the movement of users. It will know some previous positions to really predict the next position based on the previous path. It clearly increases the results, especially about the median and the mean. Indeed a model with 5 timesteps seems again to perform better on the median and the mean.
4. Too much-hidden layers, neurons or timesteps will clearly increase a lot the time spent on training without really helping to capture better the behavior of user movements and can even obtain worse results
5. Not enough hidden layers or neurons will result in having really similar results than a simple model as linear or logistic. Indeed it will be as having a model which could capture only too few information on the data.
6. To few timesteps will never capture long-term dependencies. It will result in a simple Markov chain during the prediction, so with only the current state (position) that will influence the prediction. So it will not, in our case, try to understand the movement of users, but only predict the most likely next position based on occurrences of positions.

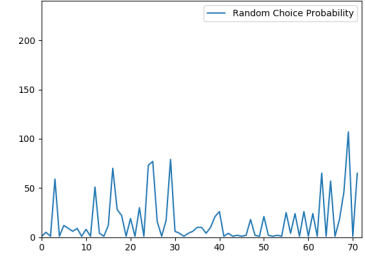
We found that the parameters from table 7 seem to respond well to prediction. So finally kept them for the last phase: Generation.



**Figure 32: Trajectory generated with Random Generator**



**Figure 33: Trajectory generated with Best Probability Generator**



**Figure 34: Trajectory generated with Random Choice Probability Generator**

## 8 Generation

Now to generate the trajectories, it had no need to run a lot of different models. A global model was trained with all our data. A new model based on the previous best parameters found but trained with all the users. The model can give us a vector of 238 probabilities for the next position based on the last 5 positions (timesteps 5). The sum of all 238 probabilities equals 1. So we have two ways to generate the trajectories based on the model:

1. By giving the model every time the last 5 positions and select the position predicted that has the highest probability in our output vector of 238 probabilities.
2. By giving the model every time the last 5 positions and select randomly between the 238 positions based on their probabilities in the output (the vector of 238). So a position with a higher probability will have more chance to be generated for the next position than a position with a low probability.

The first method is presumed better when we try to generate few steps. Indeed it will generate the most likely next positions in term of probability, but it is also more susceptible to go in a wrong cycle by predicting always the same movement. The second method can be better to go out of the same cycle of movements because even some POIs that are uncommon will be sometimes (rarely) predicted, so generated. This behavior seems more logical for generate large steps. Indeed a user goes sometimes out of his recurrent path.

### 8.1 Random Generator

In order to confirm our first thought that random model cannot capture a real human behavior, we used as a first generator of trajectories one only based on random. This generator is only to show us that if we only rely on random it does not work and moreover that our other generators give clearly better results. So it just generates the next positions randomly among the point of interests (*See figure: 32*).

## 8.2 Best Probability Generator

The second generator only based on the best probability predicted. So it really relies on our global model of prediction with the parameters found in the last section. The global RNN-LSTM model gives us probabilities for the next position. So the Best Probability Generator simply always takes the POIs with the highest probabilities for its next positions. Indeed the generator only needs that we give it five first positions and then it can begin to generate a trajectory for these positions. We can clearly see that the path that it generates seems to have a recurrent behavior or at least some cycles (*See figure: 33*). Moreover, we can notice that this generator will nearly always give the same results for the same input given. Indeed it doesn't use rely on a random process.

## 8.3 Random Choice Probability Generator

The third generator is based on probabilities predicted and random to choose between them. This generator is also based on the global RNN-LSTM model of predictions, but instead of just taking the best probabilities it is also based a little bit on random. It combines probability and random choice. So it will use the RNN-LSTM model to have the probabilities of all POIs for the next positions and it will take the next position randomly between all the probabilities of the POIs. So a POIs with a higher probability will more likely be chosen. Doing like this the method allows the path generated not to go in a repeated cycle of same movements. Moreover, it is not totally random because it is based on the probabilities given by the RNN-LSTM model. So it's just a way to try to simulate the trajectories to go sometimes in another way than the most recurrent.

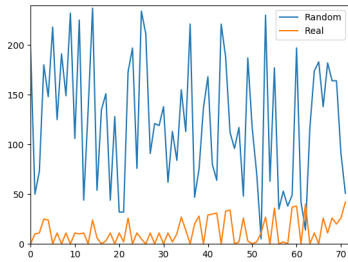
We can think of a man going every day to work; he will nearly every day go by the same path, but maybe sometimes will choose or be forced to take another path. Maybe because of too much traffic jam, maybe because he needs to look for somebody or even just simply because he is bored by this same path or he wants to try a new path. Even if, as said, this generator uses random, we see also that it is firstly based on the RNN-LSTM predictions; so it uses a bit of random based on behavior captures from users (*See figure: 34*).

## 8.4 Performance and Comparison

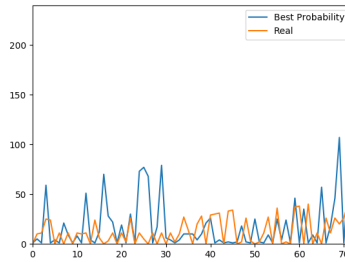
In order to really compare performances, we plot the generated path and a true path to see how they differed. So, for instance, the Random Generator obtains really bad performances. Indeed we can clearly see that, when we speak about the Random Walk Model, a generator based principally on random cannot capture a real or similar behavior than a human one. It's really obvious that for all spoken applications of trajectories, we can really not take a generator like this one (*See figure: 35*).

So then for our Best Probability Generator, we also compared a real path with a generated one. So we gave this generator simply the first 5 real positions from a true path and then it generated the next positions based on the RNN-LSTM model predictions. We see that this generator is really better than the random one. We can see that it tries to capture a human behavior (*See figure: 36*).

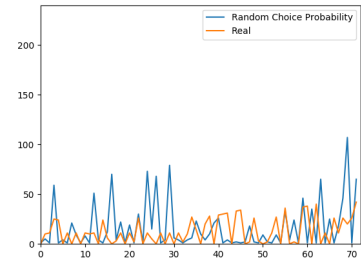




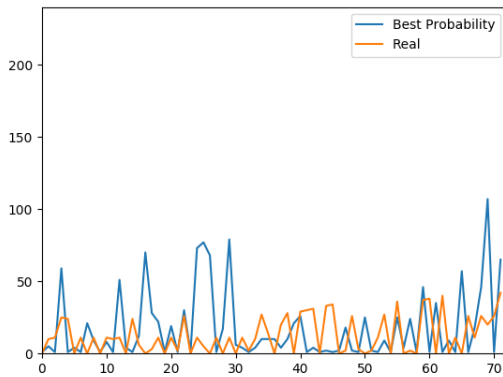
**Figure 35: Random Generated Trajectory vs Real Trajectory**



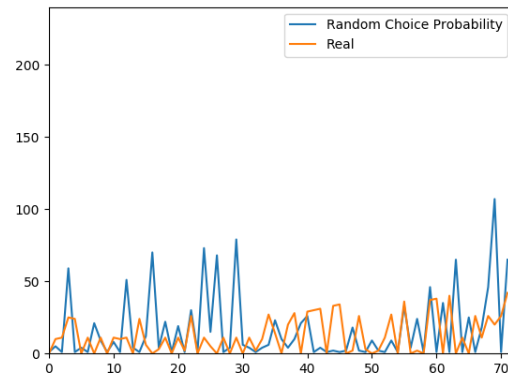
**Figure 36: Best Probability Generated Trajectory vs Real Trajectory**



**Figure 37: Random Choice Probability Generated Trajectory vs Real Trajectory**



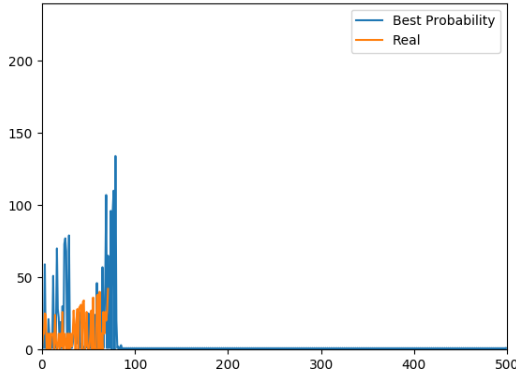
**Figure 38: Best Probability Generator with less than 100 steps**



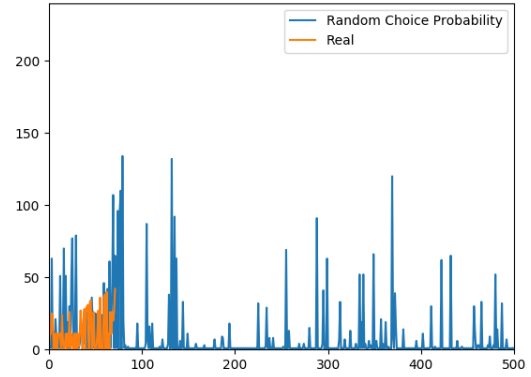
**Figure 39: Random Choice Probability Generator with less than 100 steps**

	Random	Best Probability	Random Choice Probability	Real
Mean	118.229	3.78	8.213	17.009
Median	117.81	2.2	2.72	8.3
Maximum	234.48	50.007	106.047	82.827
Minimum	2.013	0.293	0.073	0.413
Unique	122.087	9.747	27.953	30.393
Std	68.005	7.301	16.928	22.919
Skew	-0.006	4.571	3.57	1.737
Amplitude	116.233	24.857	52.987	41.207
Max slope	219.68	47.28	103.893	80.667
Correlation	-0.011	0.009	0.006	1.0
P-value	0.498	0.455	0.498	0.0

**Table 8: Some measures from the different generators**



**Figure 40: Best Probability Generator with 500 steps generated**



**Figure 41: Random Choice Probability Generator with 500 steps generated**

	Random	Best Probability	Random Choice Probability	Real
Mean	118.547	2.505	7.376	17.009
Median	118.77	1.87	2.4	8.3
Maximum	236.82	50.807	127.333	82.827
Minimum	0.12	0.2	0.053	0.413
Unique	209.287	10.04	43.267	30.393
Std	68.635	5.044	16.55	22.919
Skew	-0.001	7.189	4.07	1.737
Amplitude	118.35	25.303	63.64	41.207
Max slope	228.013	48.14	125.867	80.667

**Table 9: Some measures from the different generators**

Then for our Random Choice Probability Generator, we also compared a real path with a generated one from it. So as for the Best Probability Generator it needs simply the first 5 real positions from a true path and then it generates the next positions based on the LSTM model predictions, but with some random choice. We see that this generator is really better than the random one. We can see that it tries to capture a human behavior (*See figure: 37*). It clearly also seems to be more human than the Random Generator.

So we can clearly forget the Random generator, but what about the other two. They really seem to have similar results. When we look at each of them. They both seem to try to capture the human behavior (*See figures: 38 and 39*).

Then to try to find really a difference we choose to generate more steps, even though we have not enough true data to compare with. The generated paths obtained at this moment was clearly on average in favor of the Random Choice Probability Generator. Indeed this generator continues to have a path that looks like a human behavior. On the contrary, the Best Probability Generator nearly always seems to fall in recurrent cycles of same POIs generated (*See figures: 40 and 41*).

With these results, we can clearly consolidate the first thought that a generator based only on random is

really not recommended in synthetic trajectories. Indeed only bad results will be obtained in the applications mentioned before as traffic management. All results also consolidated the thought that some memory of past positions can help for better capture behavior of people. Indeed the results on generators based on the RNN-LSTM model show us that we clearly obtain better results than random or just taking best occurrences.

It shows also that taking only purely the higher probability given by the RNN-LSTM model for the next position is not perfect, particularly when we try to generate large steps. It can give us the problem of falling in a particular cycle without a lot of variances, so a monotone behavior without the little magic of the human kind for the change. The results conclude also that combining just a little bit of random in the generator can really help to produce a more human-like path.

It's really hard to say which one of the two generators has to be used: based only on best probability or with some random. It depends on what trajectories we need and on what kind of purpose we need it. For instance, if we need to predict only a little projection of the movement of one user, it will be better to use only the Best Probability Generator. In contrary, if we need to maybe simulate a large number of human movements, for example in traffic simulation, it will be better to use the Random Choice Probability Generator.

## 9 Discussion

During the prediction part of the project, the need to understand the context and its characteristics appeared. For instance, we saw already the importance of timesteps in the context of mobility. Moreover, we can nearly always adapt our models to the task needed. For instance, even if the other models than the RNN have no parameter such as timestep, we transformed the size of their input to fill the gap. Indeed by this way, we simulated timesteps of the RNN-LSTM model and so they could capture the human movement behaviors on their previous movements. We saw also that even if basic models seemed to try to capture some behaviors, it would have been very difficult to improve their performance given their lack of evolving parameters. Therefore the RNN-LSTM is adapted for predicting mobility trajectories, especially because of its powerful flexibility and ability to capture long-term dependencies.

Furthermore during the optimization using some timesteps clearly improved the results to finally obtain something that really seemed to integrate the behavior of users. The RNN-LSTM model responds very well when trying to figure out what is the best set of parameters. It confirms that a larger model with more neurons and hidden layers than a simple one tends to recognize more patterns and captures them. Until a certain limit, we see that increasing the size of the models increases their performances, but reaching this limit no major good effects occur. Sometimes it even decreases the performance and increases also the training time. We see that smaller models have worse results than our best one. We can not withdraw the possibility that with a longer training time, the larger models may have better results, but this is unlikely. With our setting, we are confident that we have found the best hyperparameters, especially when looking to the Mean, Median, Standard Deviation, Maximum, Correlation, and P-value.

The results from the generation stage confirm that generating a path based only on random is not imaginable for mobility trajectories. Therefore, using something to integrate the real pattern of the human mobility is the key. As is confirmed by the results of the RNN-LSTM model. Even if the model is not yet sufficient to generate perfect synthetic mobility trajectories, the Random Choice Probability Generator at least confirms that it can be possible. In further works, it would be interesting to continue improving the model, using some other features (time, delta time between movements, or others) or test it on other datasets.

## 10 Conclusion

The high level of movements of people has created new tendencies, new needs and also new problems. Moreover, the globalization of technologies has also provided the society with new opportunities. So a lot of ideas about applications linked to these movements exist or could be created. The need to obtain data about mobility exists and at the same time, it is very difficult to obtain them, particularly because of privacy [12]. It is why it is interesting to generate the data needed for mobility in order to need no more real one or at least less. It is difficult to capture the real behavior of humans in order to generate realistic trajectories, as shown by some existing work on bird migration [18] or city network [9]. Indeed most of them work only with random or/and stochastic process without really capturing global human mobility behavior. That is why we have looked for some models capable of capturing the real behavior of people such as machine learning techniques. Moreover, in our context, past positions influence significantly the trajectories. This is the reason why we focused on a model capable of learning long-term dependencies and RNN models seem to be appropriate [12] for this task.

After having compared different models, we found that the RNN-LSTM performs the best. The idea of keeping some past information about the trajectories to perform better is clearly confirmed in the domain of mobility. Even with basic models as Linear Regression model or Logistic Regression model the use of some past information allows performing better. Moreover, the RNN-LSTM model with its flexible configuration and its purpose of capturing long-term dependencies works even better for our field. With the tests done we found that a model predicts better with 5 timesteps, 5 hidden layers with 50 neurons each. Furthermore, with the prediction, we created and tested three different generators for transitions between points of interest. The first one is based exclusively on random. The second one is based only on best probability prediction from the RNN-LSTM model. The last one uses a random choice process on the prediction from our RNN-LSTM. This last generator procures better results by generating more realistic trajectories, especially when we try to generate numerous steps. Indeed its trajectories nearly never fall in redundant cycles of the same positions. Moreover, some measures extracted from it confirm its superiority, especially about Mean, Median, Unique, Standard Deviation, and Amplitude.

In conclusion, in our case, the present master thesis proves us that at some points the machine can capture and understand some human behavior in a context of geospatial trajectories. Even if the present work uses only a simplification of the problem by using only sequences of areas, we can see the efficacy of RNN-LSTM. It would be really interesting to go further in the context and try, for example, to add and test the impact of more features such as time, delta time, longitude and latitude. Moreover, we could also take a look at a model even more complex as Generative Adversarial Networks (GANs) which could show promising results. The GANs puts two models in competitions: one that generates fake data and the other one that attempts to predict if the data given is real or false. So it could be possible to use it in our case to generate synthetic mobility trajectories and even to integrate an RNN-LSTM in it. Ultimately we have proved that RNN-LSTM can be applied to generate realistic trajectories. Further development is necessary in order to create a generator usable for many applications in the fields of synthetic geospatial trajectories.

## 11 References

- [1] Ralph G Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Physical Review E*, 64(6):061907, 2001.
- [2] Jean-Daniel Medjo Me Biomo, Thomas Kunz, and Marc St-Hilaire. An enhanced gauss-markov mobility model for simulations of unmanned aerial ad hoc networks. In *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP*, pages 1–8. IEEE, 2014.
- [3] Jason Brownlee. Linear regression for machine learning. <https://machinelearningmastery.com/linear-regression-for-machine-learning/>, 2016. Consulted 05.01.2018.
- [4] Jason Brownlee. Logistic regression for machine learning, 2016.
- [5] Prathana Buranajun, Montalee Sasananan, and Setta Sasananan. Prediction of product design and development success using artificial neural network. 2007.
- [6] Cesium. Machine learning time-series platform. <http://cesium-ml.org/docs/index.html>. Consulted 05.01.2018.
- [7] Shreya Ghosh and Soumya K Ghosh. Modeling of human movement behavioral knowledge from gps traces for categorizing mobile users. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 51–58. International World Wide Web Conferences Steering Committee, 2017.
- [8] Nicolas E Humphries, Nuno Queiroz, Jennifer RM Dyer, Nicolas G Pade, Michael K Musyl, Kurt M Schaefer, Daniel W Fuller, Juerg M Brunnschweiler, Thomas K Doyle, Jonathan DR Houghton, et al. Environmental context explains lévy and brownian movement patterns of marine predators. *Nature*, 465(7301):1066–1069, 2010.
- [9] Bin Jiang, Junjun Yin, and Sijian Zhao. Characterizing the human mobility pattern in a large street network. *Physical Review E*, 80(2):021136, 2009.
- [10] A Karpathy. The unreasonable effectiveness of recurrent neural networks, 2015. *Disponivel em* <http://karpathy.github.io/2015/05/21/rnn-effectiveness>, 2016.
- [11] Minkyong Kim, David Kotz, and Songkuk Kim. Extracting a mobility model from real user traces. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–13. IEEE, 2006.
- [12] Vaibhav Kulkarni and Benoît Garbinato. Generating synthetic mobility traffic using rnns, 2017.
- [13] Christopher Olah. Understanding lstm networks. *GITHUB blog, posted on August, 27:2015*, 2015.
- [14] Sebastian Raschka. What is the role of the activation function in a neural network? <https://www.kdnuggets.com/2016/08/role-activation-function-neural-network.html>, 2016. Consulted 05.01.2018.
- [15] Scipy. Numpy.unique. <https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.unique.html>. Consulted 05.01.2018.

- [16] Scipy. `scipy.stats.pearsonr`. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html>. Consulted 05.01.2018.
- [17] Angela L. Swafford. Logistic population growth: Equation, definition & graph. <https://study.com/academy/lesson/logistic-population-growth-equation-definition-graph.html>. Consulted 05.01.2018.
- [18] Georgios Technitis and Robert Weibel. An algorithm for random trajectory generation between two endpoints, honoring time and speed constraints, 2014.
- [19] Tensorflow. An open-source software library for machine intelligence. <https://www.tensorflow.org/>. Consulted 05.01.2018.
- [20] Amey Varangaonkar. Top 10 deep learning frameworks. <https://datahub.packtpub.com/deep-learning/top-10-deep-learning-frameworks/>, 2017. Consulted 05.01.2018.
- [21] Wikipedia. Wikipedia the free encyclopedia. [https://en.wikipedia.org/wiki/Linear\\_regression](https://en.wikipedia.org/wiki/Linear_regression). Consulted 05.01.2018.
- [22] Wikipedia. Wikipedia the free encyclopedia. [https://en.wikipedia.org/wiki/Artificial\\_neural\\_network](https://en.wikipedia.org/wiki/Artificial_neural_network). Consulted 05.01.2018.
- [23] Chris Woodford. Neural networks. <http://www.explainthatstuff.com/introduction-to-neural-networks.html>, 2017. Consulted 05.01.2018.

## List of Figures

1	Sample of Linear Regression - Height vs Weight . . . . .	9
2	Distance to square of a Linear Regression [21] . . . . .	9
3	Logistic Population Growth [17] . . . . .	10
4	Sample of a Neural Network (with 3 features, 4 neurons on one hidden layer and 2 results possible) [22] . . . . .	10
5	A Neural Network with weights [14] . . . . .	11
6	A Neural Network with feed forward and back propagation [5] . . . . .	11
7	A Recurrent Neural Network [13] . . . . .	12
8	Simple RNN [13] . . . . .	12
9	RNN-LSTM [13] . . . . .	12
10	Tensorflow website [19] . . . . .	13
11	Some python code with Tensorflow on a multi-layers RNN-LSTM . . . . .	13
12	Plots [6] from EEG time series dataset [1] . . . . .	14
13	Python code [6] with Cesium to extract some selected features from a time series . . . . .	14
14	Dataset where each row represents the movements of one user between POIs . . . . .	15
15	Linear Regression model: True movement vs Prediction with 1 timestep and parameters from table 1 . . . . .	20
16	Linear Regression model: True movement vs Prediction with 5 timestep and from table 1 . . . . .	20
17	Logistic Regression model: True movement vs Prediction with 1 timestep and parameters from table 1 . . . . .	21
18	Logistic Regression model: True movement vs Prediction with 5 timestep and parameters from table 1 . . . . .	21
19	Neural Network model: True movement vs Prediction with 1 timestep and parameters from table 2 . . . . .	22
20	Neural Network model: True movement vs Prediction with 5 timestep and parameters from table 2 . . . . .	22
21	RNN-LSTM model: True movement vs Prediction with 1 timestep and set of parameters 3 from table 2 . . . . .	22



22	RNN-LSTM model: True movement vs Prediction with 5 timestep and set of parameters 4 from table 2 . . . . .	22
23	True movement vs Prediction (1 hidden layer) . . . . .	26
24	True movement vs Prediction (5 hidden layers) . . . . .	26
25	True movement vs Prediction (10 hidden layers) . . . . .	26
26	True movement vs Prediction with 10 neurons . . . . .	27
27	True movement vs Prediction with 50 neurons . . . . .	27
28	True movement vs Prediction with 100 neurons . . . . .	27
29	True movement vs Prediction with 1 timestep . . . . .	28
30	True movement vs Prediction with 5 timesteps . . . . .	28
31	True movement vs Prediction with 10 timestep . . . . .	28
32	Trajectory generated with Random Generator . . . . .	30
33	Trajectory generated with Best Probability Generator . . . . .	30
34	Trajectory generated with Random Choice Probability Generator . . . . .	30
35	Random Generated Trajectory vs Real Trajectory . . . . .	32
36	Best Probability Generated Trajectory vs Real Trajectory . . . . .	32
37	Random Choice Probability Generated Trajectory vs Real Trajectory . . . . .	32
38	Best Probability Generator with less than 100 steps . . . . .	32
39	Random Choice Probability Generator with less than 100 steps . . . . .	32
40	Best Probability Generator with 500 steps generated . . . . .	33
41	Random Choice Probability Generator with 500 steps generated . . . . .	33

## List of Tables

1	Settings of simple models during prediction . . . . .	20
2	Settings of complex models during prediction . . . . .	21
3	Basic settings during optimization . . . . .	25
4	Some measures from different number of hidden layers and true movement . . . . .	26
5	Some measures from different number of neurons and true movement . . . . .	27
6	Some measures from different number of timesteps and true movement . . . . .	28
7	Best parameters found . . . . .	29
8	Some measures from the different generators . . . . .	32
9	Some measures from the different generators . . . . .	33

## A

### Software, libraries and tools used

IntelliJ IDEA 2017.2.3

Tensorflow 1.3.0

Cesium 0.9.4

Python 3.5.4

Jupyter Notebook 5.3.0

Github