Master Thesis

Generating Synthetic Mobility Trajectories By Applying Machine Learning

BY

Yannick Patschke

Prof. Benoît Garbinato
M.Sc Vaibhav Kulkarni

Abstract

The present paper focuses on how generate synthetic mobility trajectories which seem human. So the purpose is to generate data that really captures human behavior. For this task we use models of machine learning to first try to see which one could predict next positions of people better, then optimize it and finally generate movements based on it. For that we focused on the idea that having some memory about past positions could be usefull. Indeed as movements of people go clearly into one direction at the same time, it is interesting to know what were their last steps in order to better predict or generate their directions. Among all models of machine learning the Recurrent Neural Network RNN seems to fit for the task and its specificities. Indeed the RNN is a model that can keep information about past events or past results. It confirms us it by the results that we obtained. So the use of a specific RNN named LSTM (Long Short-term Memory) resulting the fact that we clearly could capture human behavior and even that using past positions in a field as mobility trajectories is clearly important. We conclude that machine learning can be suprising and perform great as in many other fields. Indeed nearly all fields and data can work with machine learning and in our case that it could even capture complex information about data as human behavior.

Contents

1 Introduction **5**

2 Motivation **5**

3 Background **6**

3.1 Machine Learning . . . 6

3.1.1 Linear Regression Model . . . 7

3.1.2 Logistic Regression Model . . . 9

3.1.3 Neural Network Model . . . 9

3.1.4 Recurrent Neural Network Model . . . 12

3.1.5 Long Short-term Memory Model . . . 12

3.2 Tensorflow . . . 13

3.3 Cesium . . . 13

4 Related Work **14**

5 Methodology **14**

5.1 Prediction . . . 14

5.2 Optimization . . . 14

5.3 Generation . . . 15

6 Nokia Dataset **15**

7 Prediction **15**

7.1 Data Pre-processing . . . 15

7.2 Prediction Performance . . . 16

7.2.1 Linear . . . 17

7.2.2 Logistic . . . 18

7.2.3 Neural Network . . . 18

7.2.4 Long Short-term Memory . . . 19

7.3 Prediction Overview and Comparison 20

8 Optimize **21**

8.1 Hyperparameter Selection 21

8.1.1 Hidden layer(s) 22

8.1.2 Neurons 24

8.1.3 Timesteps 26

8.2 Hyperparameter Selection Values 28

9 Trajectory Generating **29**

9.1 Random Generator 29

9.2 Best Probability Generator 30

9.3 Random Choice Probability Generator 30

9.4 Performance and Comparison 31

10 Discussion **34**

11 Conclusion **35**

12 References **36**

1 Introduction

As nowadays the mobility of people has clearly never been so high with so much means of locomotion, it could be interesting to have the possibility to predict or even generate trajectories of people. Indeed there are several useful applications where predicted and generated trajectories can be involved: Traffic management, Urban planning, Consumer profiling. For now in practice it is still difficult to really generate trajectories of humans. Indeed the real problem is the human fact. Generating trajectories of projectiles has been done since many years with ballistic and even trajectories of some mechanic and automatic stuffs already have been done. Generating human trajectories is really more complex due to the fact that we are so different each other's. Humans have all their own habits, their own different life, their own different works and also on the same time their own specific trajectories in the life of every days. So capturing their real behavior and generating their trajectories is really a challenge.

In order to try to succeed in this challenge the approach that I selected was to find a way to really capture the human character of its trajectories. Indeed the only way to product a good generator of synthetic mobility trajectories is to extract behavior of real human trajectories. So for that the idea was first to create a model that simply predicted next positions of people based on their last position or even some of their last positions. Then find a way to generate different successive positions based on the model of prediction. To do all that I needed some skills and knowledge in informatic and mathematic. They can be regrouped in the field of machine learning.

Machine learning is really fascinating. It can be applied in so many fields and it would be too long to enumerate them all. Moreover machine learning has actually so many different models and nearly all of them could in a way or another be adapted to my project. One of these models seems specially fit for my approach concerning the use of some past positions, so some past informations to extract behavior. This model is named Recurrent Neural Network (RNN) which is specially created to learn on data where past events or past results influences the next result. So it's why RNN seemed to be appropriate for generating Synthetic Mobility Trajectories and why I focused on it.

2 Motivation

The increase of movements not only human movements, but also materials or informations, the number of shifting and movements in our society has never been so high in our history, 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 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 for the general public. So a lot of things that were one century ago reserved to rich or a particularly 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 increasing movements on people, informations and materials. It's really difficult to imagine the amount of data that we could collect just about a movement of goods in one day in a city, and how much about the number of letter ships or people moving to work. Getting all these informations is possible at the technological level, now that some technologies as gps and connected device are so widely used and accessible. All these different datas about mobility can be useful in some ways. If we think only about human mobility, we can already

think about a lot of fields where data about human mobility could be useful:

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

Urban planning: With behavior of movement of people, we could predict the best areas to construct, or maybe think how to increase existing area level.

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 about all the above mentioned fields on mobility traffic is that it needs a lot of data to obtain results which could make sense It's not enough to just 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 organisation to obtain enough data. Because you need to ask people directly, and really a lot of people.

The second main problem will result from the 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 mobility traffic data is just not imaginable. If it was the case, people would maybe participle, but the city or the organisations would need really a lot of funds. Fortunately I 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 for all the above reasons, generating synthetic geospatial trajectories could be an interesting thing. Indeed it allow us to would need really a lot less data, which would mean also a lot less problems about collecting them.

3 Background

There are already some helpful knowledge and tools really useful for my project. So in this section you can find some of them with their respective overview and their explanations. For instance we focus about a machine learning overview and some of its subsections related in my work. This section speaks also about some really interesting and relevant library existing.

3.1 Machine Learning

Nowadays Machine learning is really widespread even if most people don't really know its utility and the possibilities it can offer. Machine learning is basically, to teach to the machine to act in situations that differ from what we have shown or trained her. It's a process where we train the machine with examples and then give new but similar data and let the machine try to work with this 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 allows us to train the machine based on example data and their results. So basically during supervised machine learning, we show the machine our examples and their correct results several times. Until the machine has learned some pattern about the data, and understood that if we give her a 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. So when he has correctly read all these books, we can assume that he understands how read different words, and 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 and try to make better the next time. So we can say that supervised machine learning, it mainly use to understand the way to go from a point A (our input data) to a point B (our output data, the correct answer).

Unsupervised machine learning works differently; this time the machine will just train on examples without feedback of what is true or false, it means without correct answers to find of the example data. So imagine again a child who is given a pile of marble to tidy up separately, but with no other instruction about criteria to separate them. Maybe the child will separate them by color, size, weights or others, maybe mixed criteria. But 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 between some examples and try to find a way to group them together. So here it's an example about clustering and we can see that unsupervised machine learning works more to find some structure in the data given.

We can use or see unsupervised and supervised machine learning, nearly everywhere and any times, if we are attentive. In our society, we can really find traces of machine learning everywhere. For instance in our email 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 in my opinion machine learning will surely continue to extend in our society. Indeed these last decades have allowed really to see the power of computer increase. So it allows to process a large amount of data faster than before. It makes more interesting to use machine learning; indeed machine learning needs often to work with a large amount of train data to be efficient. So these last years have been really good for this field. We have seen more and more people using machine learning, not only in big organisations but also private people.

Indeed everybody doesn't need to have an expensive computer to make machine learning, but most of 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 more others. [30] 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 the Long Short-term Memory Model.

3.1.1 Linear Regression Model

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 and learn basic stuff in machine learning. So a linear regression model is basically a model which assumes a linear relationship 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, BO and $B1$ coefficients.

Sample of Linear Regression - Height vs Weight

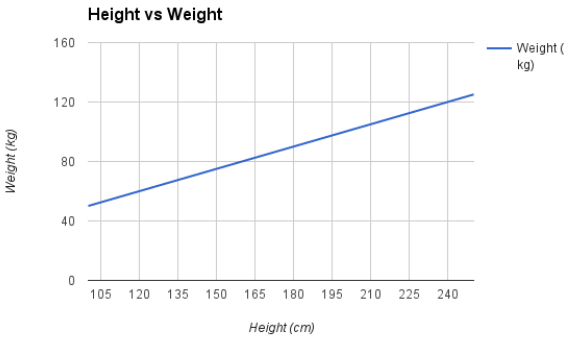

Figure 1: Source: MachineLearningMastery.com [3]

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. Then we square distances, and sum all of the squared errors together. This is the quantity that ordinary least squares seek to minimize. This quantity is often called cost function in machine learning.

Distance to square of a Linear Regression

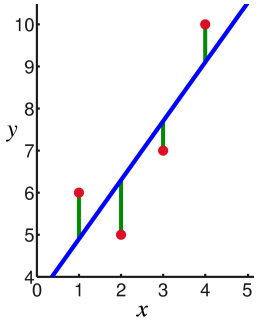

Figure 2: Source: Wikipedia [31]

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 values for each coefficient. The sum of the squared errors are calculated for each pair of input and output values. We also use in this a learning rate as a scale factor and the coefficients are updated in the direction towards minimizing the error. The process is repeated until a minimum sum squared error seems to be obtained. So at this 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 Model

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 François 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.

Logistic Population Growth

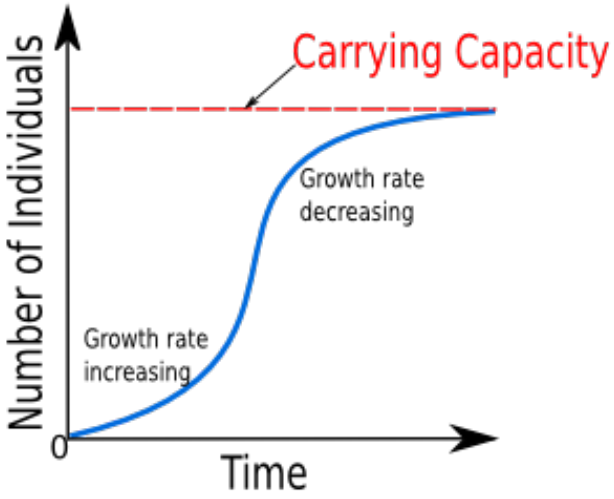

Figure 3: Source: Study.com [27]

So basically the function in our case can be represented as this $y = e^{(b0+b1*x)}/(1+e^{(b0+b1*x)})$. Where y is the predicted output, $b0$ is the bias and $b1$ is the coefficient for the input value. So each features of our input data has a coefficient that we need to determine with the train step as I 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 Model

The Neural network model is quite different. It's a more complex model and really 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 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 by many cells that simulate the brain. The Neural Network can be separated in 3 different parts:

1. Input layer: It contains input cells which are regrouped in one side of the network. These cells represent informations from the outside world that the network will attempt to assimilate. It's basically all features and informations 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 be represented as 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 by 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 number of 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 with machine learning in practice we can have some surprises. A small network could sometimes be better on complex data and in contrary a complex network can be good on simple things. Indeed other parameters can influence the performance of the network on the data.

Sample of a Neural Network
(with 3 features, 4 neurons on one hidden layer and 2 results possible)

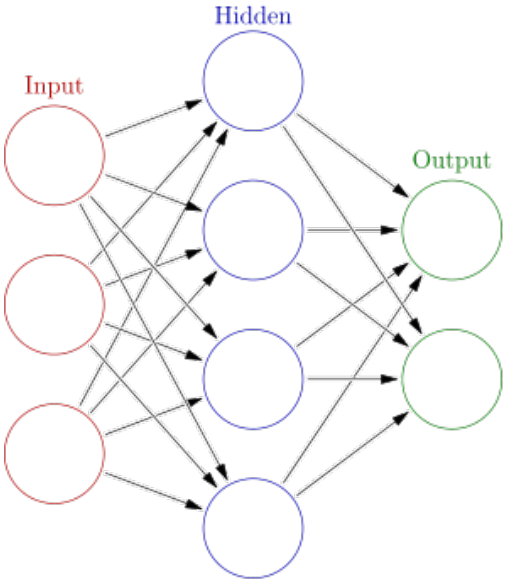

Figure 4: Source: Wikipedia [32]

The connections between one cell and another are represented by a number called a weight. Weights represent the influence that one unit has on another. So more the weights are high more the selected cell influences the cell linked.

A Neural Network with weigths

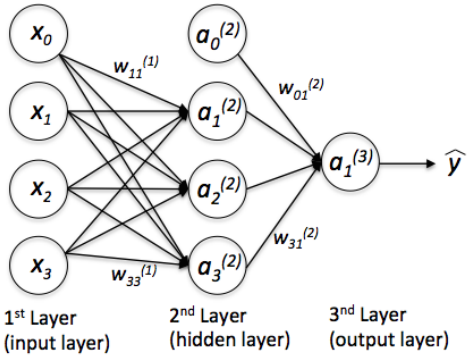

Figure 5: Source: KDnuggets.com [23]

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 througth 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 it's results. The second step is where the backpropagation process occurs. It is a feedback process, in order to compare the 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 [33] for more details about the model)*

A Neural Network with feed forward and back propagation

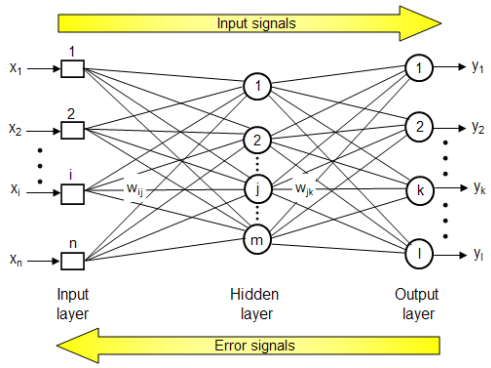

Figure 6: Source: ResearchGate.net [6]

To summary 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 informations 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 Model

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 informations also when we need to have some persistence. 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 good. Indeed a lot of informations about the word given is in its context. So the only way to really capture and determine these informations 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 leat a part of it. At only then a RNN makes sense. We can see a 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 his successor. So we have a persistence of some past events or some past results. *(See [16] for more details about the model)*

A Neural Network with feed forward and back propagation

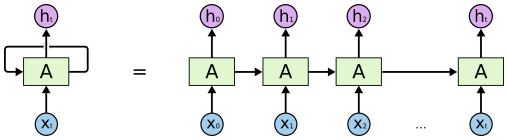

Figure 7: Source: colah.github.io [21]

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.

3.1.5 Long Short-term Memory Model

The Long Short-Term Memory (LSTM) is a special kind of RNN, which is capable of learning really the long-term dependencies. The LSTM does have the ability to remove, select and add pertinent informations from past events. So it can really retain important data from previous inputs and use that information to modify its current output. In contrary of a simple RNN which has less control about what kind of data from past events need to be retained a 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 a LSTM has clearly a good selective memory even on long-term. It's also wy the model is the main model of RNN used in practice. *(See [21] for more details about the model)*

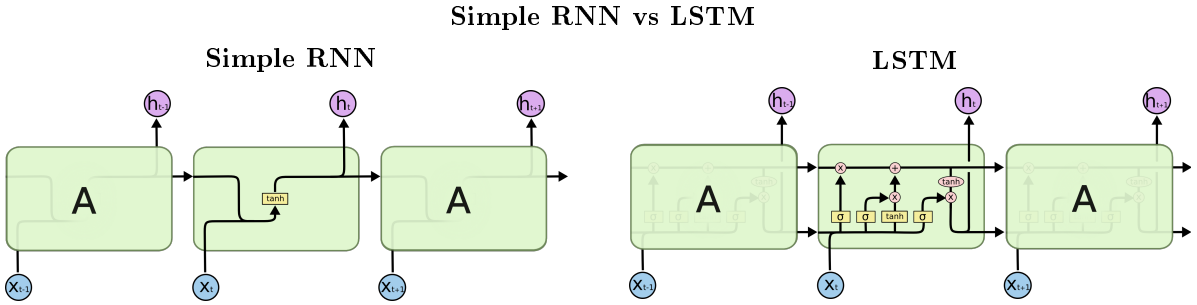

Figure 8: Source: colah.github.io [21]

3.2 Tensorflow

Building and using some models for machine learning from the scratch is totally possible. Indeed we only need some mathematic knowledge about it and some skills in programmation to do it, but a lot of libraries exist already to optimize it as Torch or MXNet [30]. So we didn't really need to loose time by creating these models from nothings. Tensorflow [29] is one of these numerous libraries for machine learning and the one I used for my 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 to work 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 runned on multiple platforms: CPUs, GPUs, mobile and others. It can also be used with different level of details. For example on one side we can use the Keras part of Tensorflow like using machine learning with really less needs and knowledge about how really works 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.

3.3 Cesium

Cesium [7] is also a useful library, but in contrary of Tensorflow it doesn't work to properly run Machine Learning models. Indeed Cesium is usefull in another specific field: time series. Indeed it can extract some real 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 much 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 in Python script. Then through its methods for my case simply select some useful features to extract on the relevant data. So Cesium can be really helpful for some little analysis of time series.

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 mine. So trying to discover and work all by myself without according regards to the works of others would be like wasting time. Indeed looking at model and research of other make me win a lot of time to understand similar problems and how handle some points. Moreover people have also worked on generation of mobility trajectories, but not necessarily about human trajectories. Indeed Technitis Georgios and Weibel Robert [28] 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 trajectoires seem good, but still are mainly based on random.

Another work which caught all my attention was the work of Bin Jiang and al [15]. They work 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 still constraint to follow street network. Then compare the results with real walkers and it seems that random walkers and real one have common behavior. All this pushed me to try to go further and really try to capture behavior of users. So I would do it, but not necessarily in a goal oriented to prove some facts about society or human behavior.

5 Methodology

Every project needs to have a plan to follow or to try to follow. So to achieve my project of creating generating synthetic mobility trajectories, I divided it in 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 datasets we have available, check where and maybe why they perform better than others. We need also to try to find their limits. When we are confident in a model and its performances, we can go to the next step to go further.

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 neurones or the number of hidden layers for a neural network, and so one. We need also to check the best way to give him data, and try to go to the limit of the model with the datasets in our disposition. So a big part of testing, switching some little things, comparing them, and trying to reach to conclusions and some results to finally find the best parameters for our model on our data.

5.3 Generation

The last part of the project is about generating data based on our model of predictions. So For example by giving our program a starting position and letting the model give us 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 fits with what we expect and if it seems near the human behaviour observed in our datasets. Finally give some conclusions about it.

6 Nokia 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 informations were 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 my country and near my 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, I needed to select a way to begin. Starting directly with positions (longitude, latitude) and time with machine learning is not necessary 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 I 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 I work now is simply movements between these points of interest. So all this data are put in a sequence. We have then a sequence of index, where each index represents one of these zones where users stay more than a certain time. For 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.

7 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 theirs last position or some of theirs last positions. Then I compared them to finally be able to argue in favor of one of them for the next steps of the project. For all that I first began with some data pre-processing to be able to be used in good way with all my future models.

7.1 Data Pre-processing

As said in the section *Nokia Dataset*, I have worked on a simplification of the problem with sequences of point 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 with some readings, I 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.

7.2 Prediction Performance

So now let's speak about the results of the model used. The model was set up to try to predict the next positions of a user based on his last position or some of his last positions. First I set the same simple parameters for all models used. So I set learning rate, bach size, number of epochs, training size, input, output, timesteps, hidden layers and neurons. In my 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 it's prediction between training cycles. On one hand, higher the learning rate is, faster the model will change and adapt his 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 his apprenticeship and be in error.

Batch size = 20 : It corresponds to the number of input data given to the machine in 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 we have done one epoch, we have given all training data to the machine to learn. So indeed the number of epochs corresponds to the number of time all training data will pass trough the training.

Training size = 0.7 : It is the percentage p of all our 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.

Output : It's also a vector but instead of 238 binaries, it is of 238 probabilities. So it corresponds to the probability of being in one each of this 238 POIs in next position. Obviously it is only the prediction of the model. But we can then take the maximum probability obtained from these different POIs and say that the model predicts this one as the next position.

Timesteps 1 until 5 : It corresponds to the number of previous positions that will be taken in 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 timesteps set to 2 corresponds in my 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 it's complexity.

Neurons = 20 : It is the number of neurons per hidden layers. Where each neuron tries to compute score based on the weights of theirs 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).

7.2.1 Linear

The Linear model was really a basic model. So I didn't expect it to work really well. The first result confirmed my thoughts. It was with only one timestep, so by trying to predict the next positon with only the last position of the user. We can see that the movement predicted was clearly not good. It didn't really fit with the true trajectories linked.

Linear model: True movement vs Prediction

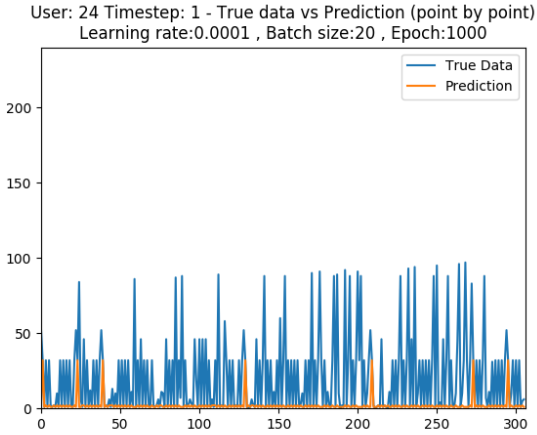

Figure 9: Generate with matplotlib [19]

So then I tried with more timesteps until 5. It seemed quickly to improve the performance of the predicted positions, even if I used a linear model.

Linear: 1 timestep vs 5 timesteps

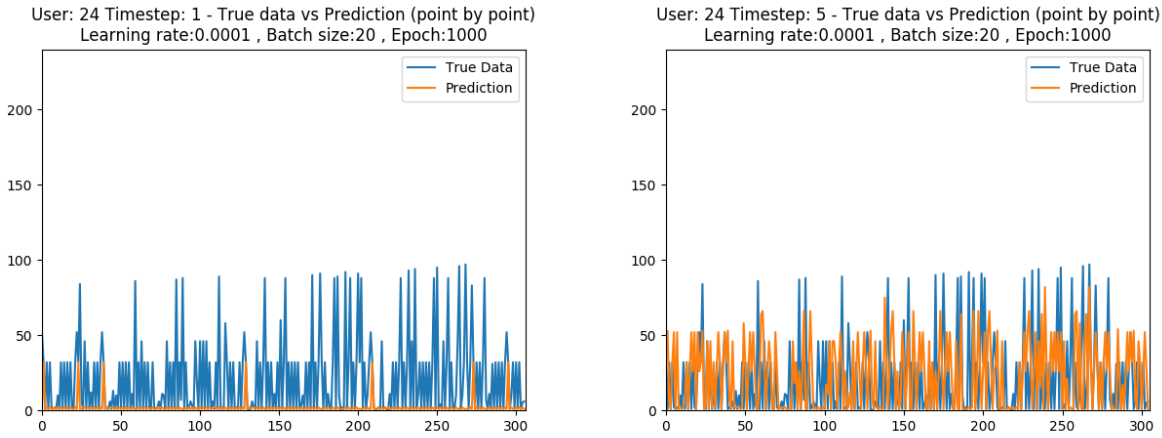

Figure 10: Generate with matplotlib [19]

These first graphs and results seemed finally not so bad, but if we wanted 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. So we can conclude that this model allows to show us that even now we can see that some timseteps can really improve results, but we can't continue to use it for capturing the behaviour of a human movement. We need a model that can capture more informations and perform better.

7.2.2 Logistic

The Logistic model was also quiet a basic model. So as the linear model I didn't expect much of it and the results also proved it to me. Even so the results are quiet good for a so simple model.

Logistic: 1 timestep vs 5 timesteps

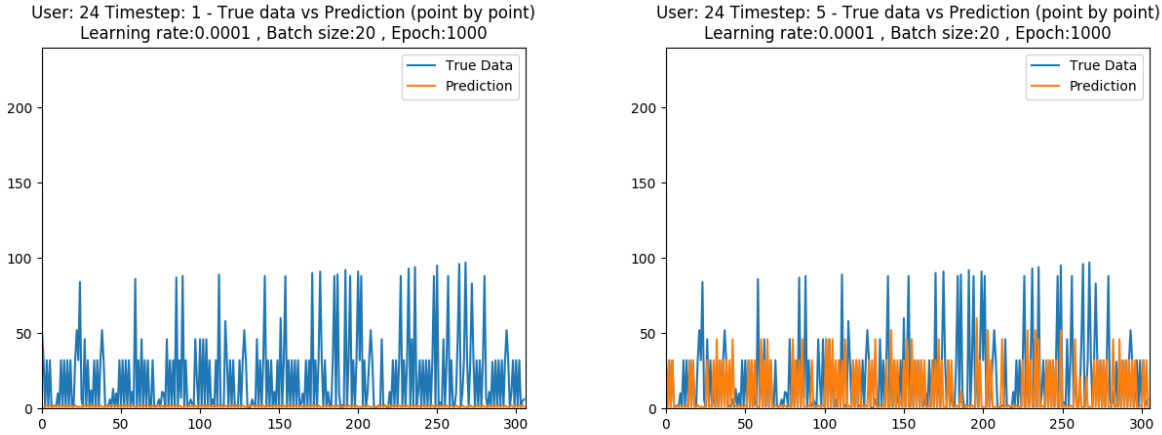

Figure 11: Generate with matplotlib [19]

Same problem as the linear model, it shows us that having more timesteps is already better, but really work with it will be hard. Its results seem too much mechanical and without the human touch and as the Linear Regression Model it will be hard to really improve it, because of its lake of parameters and flexibility. So I concluded that we needed a model more complex, flexible and that we could really optimize.

7.2.3 Neural Network

First with the neural network, I was also disappointed and upset of my first results with only one timestep.

NN model: True movement vs Prediction

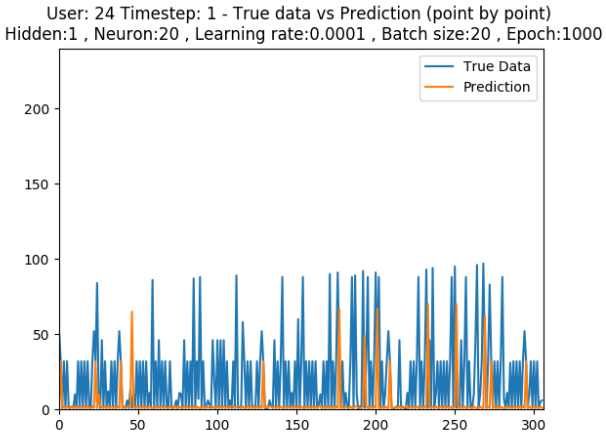

Figure 12: Generate with matplotlib [19]

It seemed to be as the basic models (linear and logistic) for prediction not really accurate. Then after a second look to really check with more timesteps, I saw that the prediction seemed really more near a human behaviour.

NN: 1 timestep vs 5 timesteps

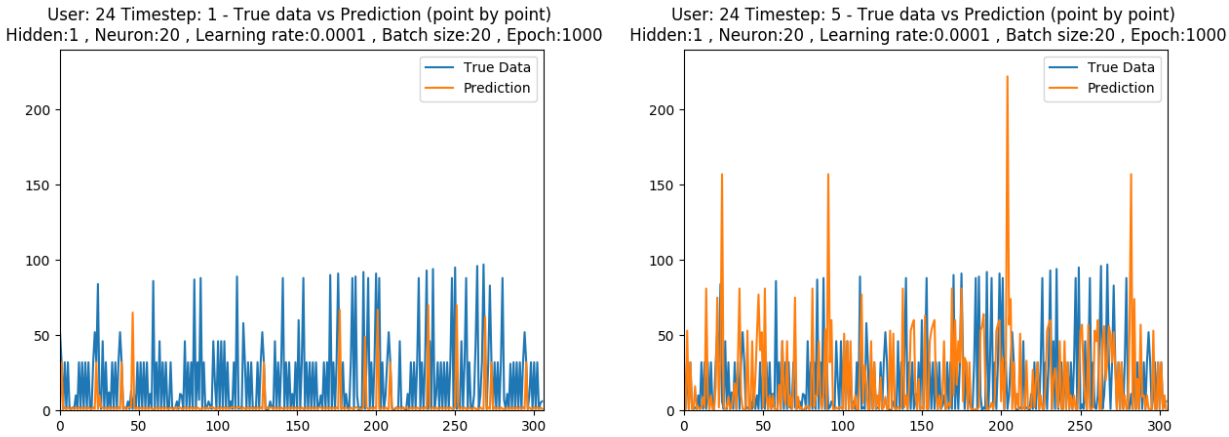

Figure 13: Generate with matplotlib [19]

We can see that even with some basic settings the model is already trying to catch a more real behavior of movement than a simple model as the Linear and the Logistic.

7.2.4 Long Short-term Memory

For the LSTM model, nearly the same effect occurs when we look at trajectory predicted by the model.

LSTM: 1 timestep vs 5 timesteps

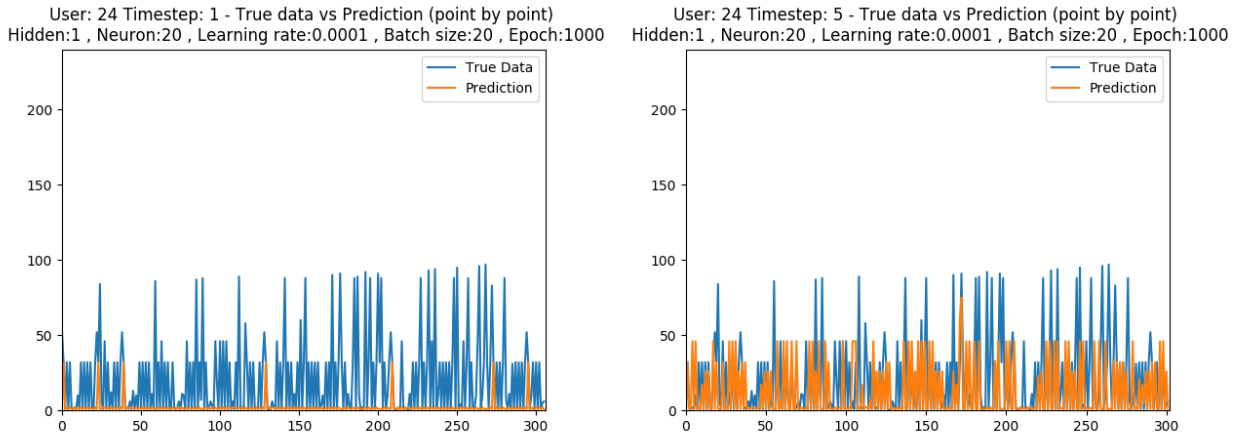

Figure 14: Generate with matplotlib [19]

After having seen the real behaviour 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 that having some memory of past information can really help us to have better results, even we saw it only a little bit.

7.3 Prediction Overview and Comparison

Indeed in all my models, we can see that they capture some informations and behaviour from the data given to them, but only the Neural Network and the Recurrent Neural Network seems to give us something near the reality more often than basic models. We can always 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 we have less parameter to really improve results. Indeed with parameters of the Logistic model and Linear model we can only try to change some basic stuffs for their training as the number of epochs, the learning rates, so without really improving or really changing them. It is as trying to improve a model that is already near its full capacity, so very complicate. In contrary of the NN and 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. So it's why, I choose not to spend more time with a basic model and try next to optimize the Recurrent Neural Network: my LSTM.

The Neural Network seemed also really promising, but it is just because I tried to simulate the potential of the LSTM as with the basic models. Especially I 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's not really useful, and even more the way I gave it 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 memory of past informations or results. To simulate this memory I simply increased their input data given, so finally they had really larger inputs when working with more timesteps. Indeed when the 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 I simulated timesteps of the LSTM and by the way also had inputs clearly larger. So the time that these models take to process it is also higher. So it's also why I choose to clearly continue with only the LSTM.

8 Optimize

The next big step was to try to find the best or at least good parameters for my models, parameters that could give me interesting results. So set and test some parameters then try to change them. For this, I tried to select some different range of values for my parameters and tried to compare their results. For instance I 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 I continued to increase it, but on the contrary if the results were worse, I decreased the parameters. And then I followed the same process for some of the other parameters.

It was clear that this task could take a lot of time, if we know that there are 150 data's users to train, so as much models to train. Having time and computer power limited, I choose to only train and make results for a part of my data. I choose to take only 25 users with more data, so in my case more movements between POIs. I choose this number of users because I 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 my users and their areas visited.

I used my models with different parameters on these users. Thereafter I obtained also different results for each of my 25 users selected. And I 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 I choose two simple ways to evaluate the results:

1. Manually by simply looking at the movements predicted for each users and comparing them with some real one.
2. Statistically by selecting some measures that seemed interesting. So these measures were used on the movement predicted and the real one, then compared with each other. These measures have been then again compared with the results based on other parameters, and I choose the parameter that seemed to have measures nearest to those from true data.

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

8.1 Hyperparameter Selection

Concerning my parameters in comparison with the ones in the section *Prediction*, I just increased the batch size and the number of epochs, because now I wanted to train the machine with only users that had more transition, so more data. Moreover I 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 trainings, so I choose to increase 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 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 user with really few transition, but now I restricted my job with only users with more transitions. So the batch size could be higher easily. So I set first some basic parameters:

- Learning rate = 0.0001
- Batch size = 100
- Number of epochs = 5000
- Training size = 0.7

Then as I said some measures were taken in consideration to compare the results of my different settings. These measures were extracted from the predicted path of my models. So the path consists of each points 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. [25]

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. [26]

P-value: The p-value roughly indicates the probability of an uncorrelated system producing datasets that have a Pearson correlation at least as extreme as the one computed from these datasets. [26]

8.1.1 Hidden layer(s)

First, I tried to find the best number of hidden layers so I 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 we have 50 neurons per hidden layers and a timesteps of 5. We can see some features extracted from generate path predicted (point by point) of my different results and the real mean based on true movement from my 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 I 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.

Some measures from different number of hidden layers and true movement

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

Figure 15: Source: Generate with matplotlib [19]

Then for instance we can see that with the following graphs that compare movement generated point by points that 5 hidden layers is better than a unique hidden layer. We can see that the model with 5 hidden layers tries to capture more transitions and clearly more different positions.

LSTM: 5 Hidden vs 1 Hidden layers

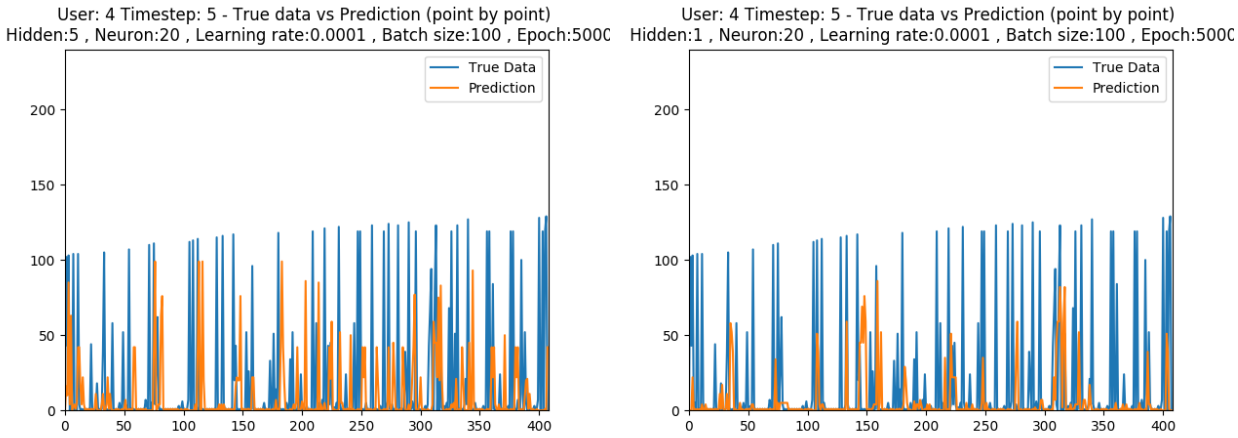

Figure 16: Generate with matplotlib [19]

Then we can also see that more hidden layers decrease a lot the results. It seems that the model with too much hidden layers begins to only predict the POIs with more occurrences. We can presume different

effects. Maybe there is 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.

LSTM: 5 Hidden vs 20 Hidden layers

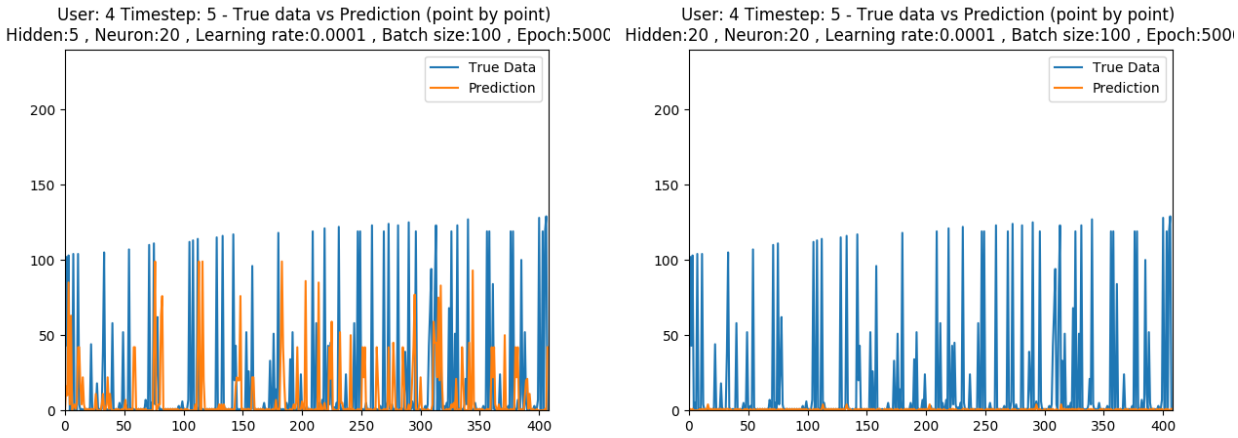

Figure 17: Generate with matplotlib [19]

However all these results seem to be in favor of 3 or 5 hidden layers, but accordingly to some predictions of some users, it seems better to keep 5 hidden layers. So I kept these parameters for the following.

8.1.2 Neurons

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

Some measures from different number of neurons and true movement

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

Figure 18: Source: Generate with matplotlib [19]

We can see it here that too few neurons per hidden layer will clearly decrease results. In the same way too much 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 is 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 less neurons than 50 is a bad choice.

LSTM: 50 neurons vs 10 neurons

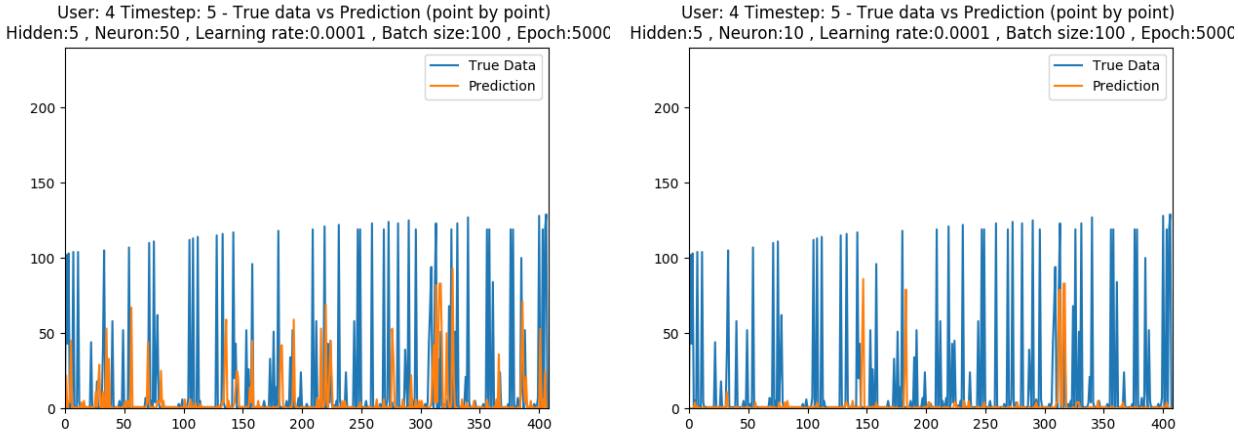

Figure 19: Generate with matplotlib [19]

In the same way it confirms that too much neurons will not really improve results and as said will just

increase the training time.

LSTM: 50 neurons vs 100 neurons

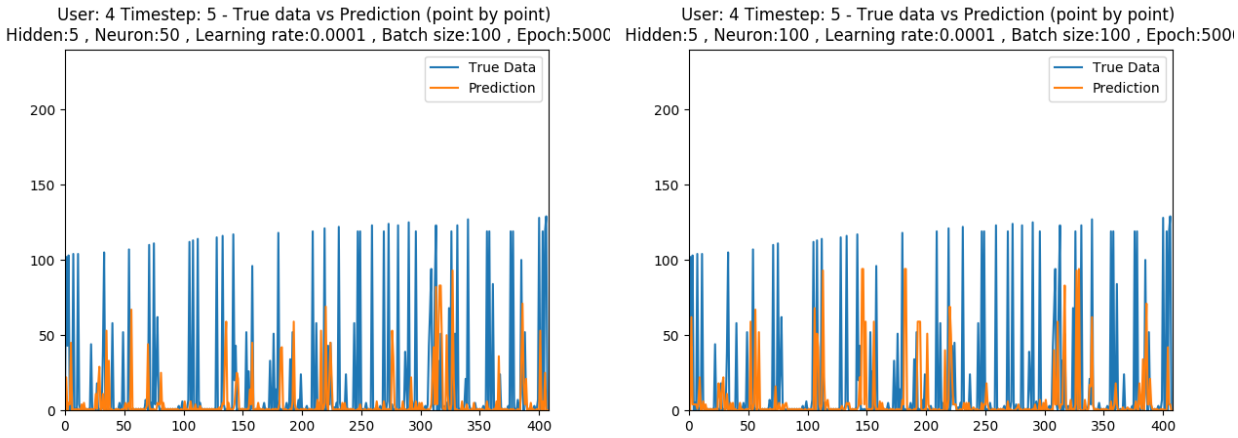

Figure 20: Generate with matplotlib [19]

So I set neurons per hidden layers to 50 for the following.

8.1.3 Timesteps

Then I tried to find the best timesteps, so the best memory of last positions to take in account for predictions. So I tried different timesteps with my previous best parameters: 1,3,5,7 and 10 timesteps. It resulted that 5 timesteps is 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 having 5 timesteps.

Some measures from different number of timesteps and true movement

	1 T	3 T	5 T	7 T	10 T	Real
Mean	4.068	13.565	17.843	17.47	17.692	22.057
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	46.64
Std	7.008	22.41	25.169	25.47	25.896	33.329
Skew	6.444	3.097	2.489	2.318	2.296	2.229
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

Figure 21: Source: Generate with matplotlib [19]

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 comparing with the real one. So here it clearly is better to choose the model with 5 timesteps. Indeed this model perform 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 less timesteps than 5 is a really bad choice.

LSTM: 5 timestep vs 1 timesteps

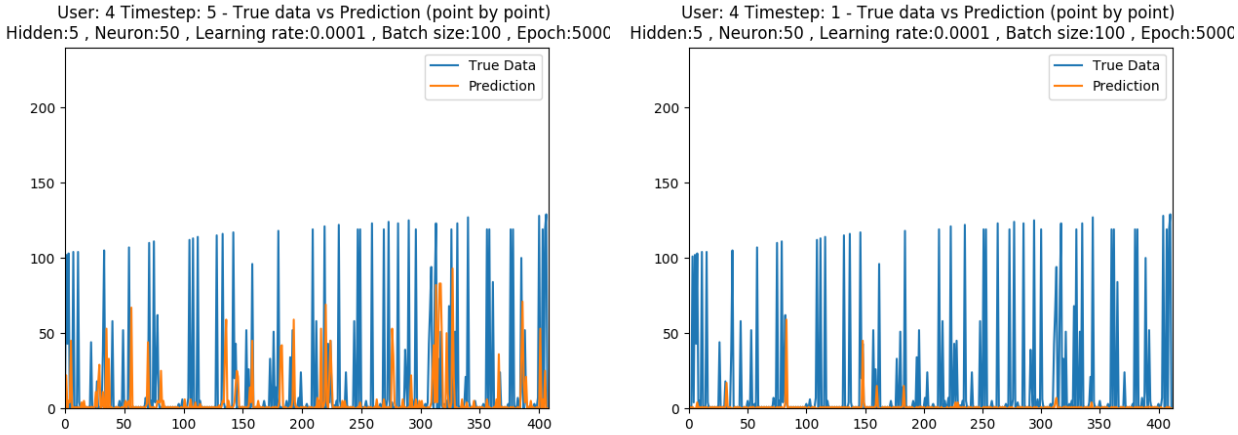

Figure 22: Generate with matplotlib [19]

It is less obvious that too much timesteps is bad with the same type of graph, but we can still conclude that more timesteps don't really improve the results.

LSTM: 5 timestep vs 10 timesteps

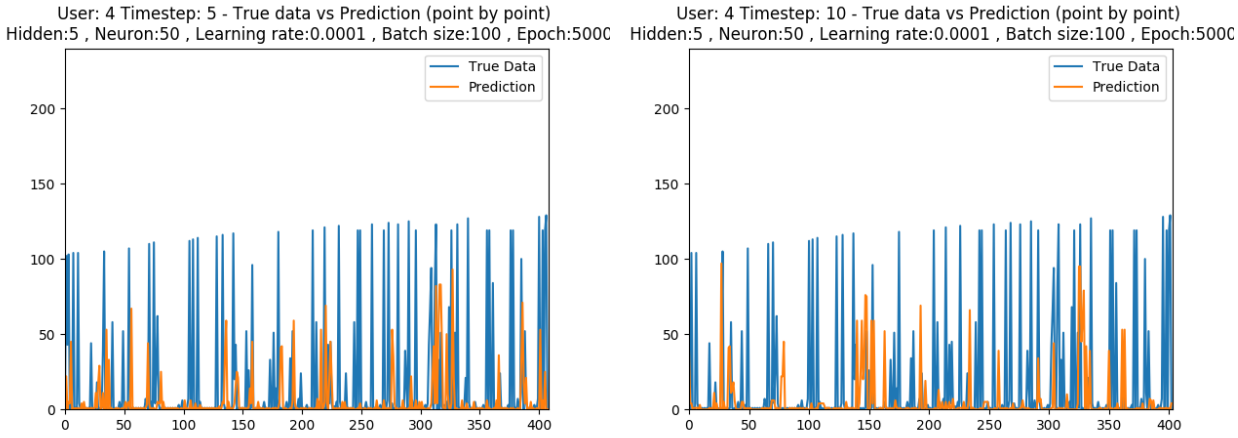

Figure 23: Generate with matplotlib [19]

So it's why I set the timesteps to 5 for the following of the project.

8.2 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 my final selected parameters, but on average worse. Indeed machine learning as RNN can be really complicate 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 my test, 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 to 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 mean. Indeed a model with 5 timesteps seems again to perform better on median and mean.
4. Too much hidden layers, neurons or timesteps will clearly increase a lot the time spent for 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 informations 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.

I found that the following parameters seem to respond well for prediction. So finally kept them for the last step: Trajectory Generating.

- Learning rate = 0.0001
- Batch size = 100
- Number of epochs = 5000
- Training size = 0.7
- Timesteps = 5
- Neurons = 50
- Hidden layers = 5

9 Trajectory Generating

Now to generate the trajectories, I had no need to run a lot of different models. So I used all my data for training a global model. So I created my new model based on the previous best parameters found, but trained with all the users. So my 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 equal 1. So I have two ways to generate the trajectories:

1. By taking every time the last 5 positions and select the position that has the highest probability in our output vector of 238 probabilities.
2. By taking 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 if we want to predict large steps, because a user's behavior goes sometimes out of the box, so out of his recurrent path.

9.1 Random Generator

In order to confirm my first thought that random model cannot capture a real human behavior, I used as 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 my other generators give clearly better results. So it just generates the next positions randomly among the point of interests.

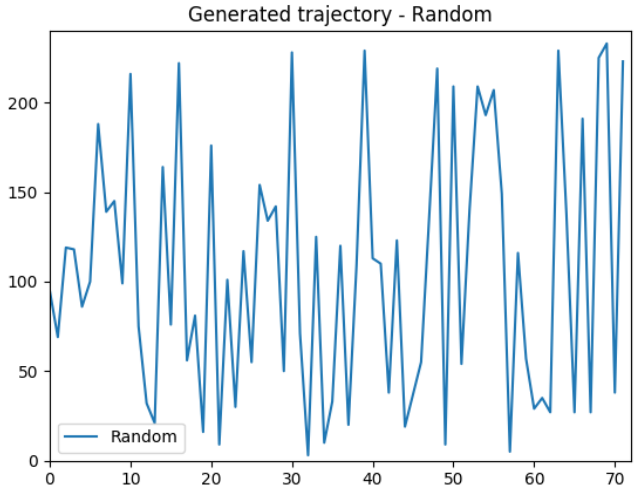

Figure 24: Source: Generate with matplotlib [19]

9.2 Best Probability Generator

The second generator only based on the best probability predicted. So it really relies on my global model of prediction with the parameters found in the last section. The global LSTM mode gives us probabilities for the next positions. 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.

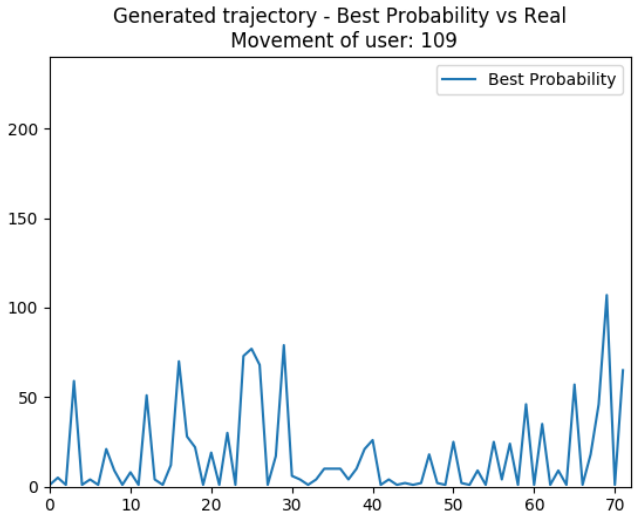

Figure 25: Source: Generate with matplotlib [19]

9.3 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 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 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 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 days go by the same path, but may be 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 I said, this generator uses random, we see also that it is firstly based on the LSTM predictions; so it uses a bit of random based on behavior captures from users.

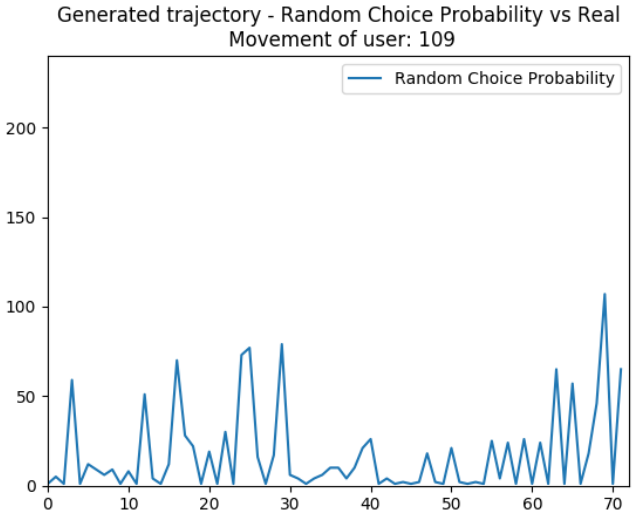

Figure 26: Source: Generate with matplotlib [19]

9.4 Performance and Comparison

In order to really compare performances, I 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 then a human one. It's really obvious that for all spoken applications of trajectories, we can really not take a generator like this one.

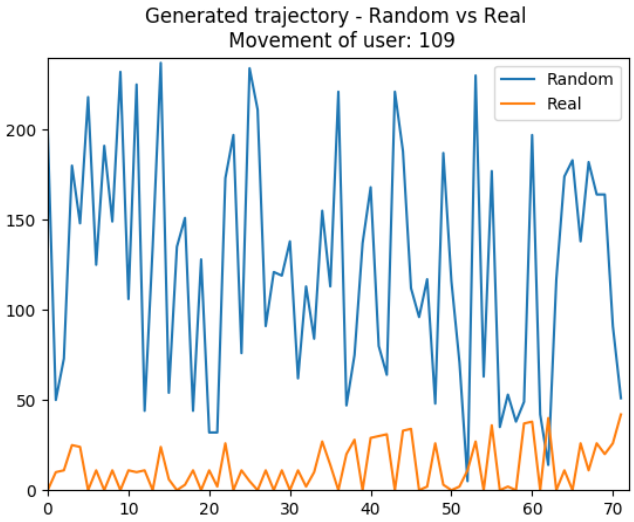

Figure 27: Source: Generate with matplotlib [19]

So then for my Best Probability Generator, I also compared a real path with a generated one. So I gave

to this generator simply the first 5 real positions from a true path and then it generated the next positions based on the 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.

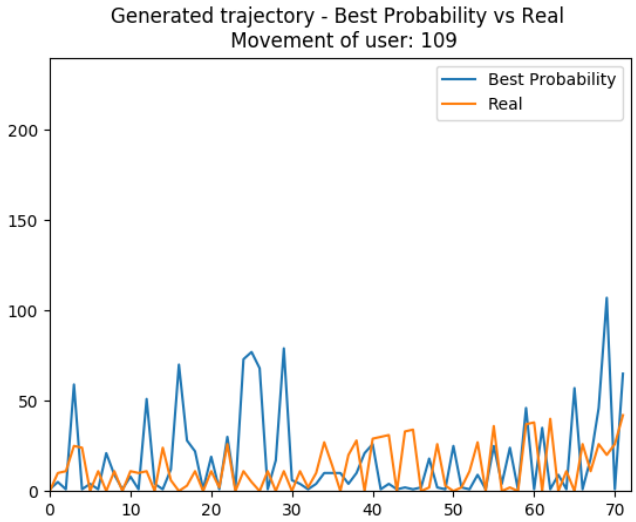

Figure 28: Source: Generate with matplotlib [19]

Then for my Random Choice Probability Generator, I 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. It clearly also seems to be more human than the Random Generator.

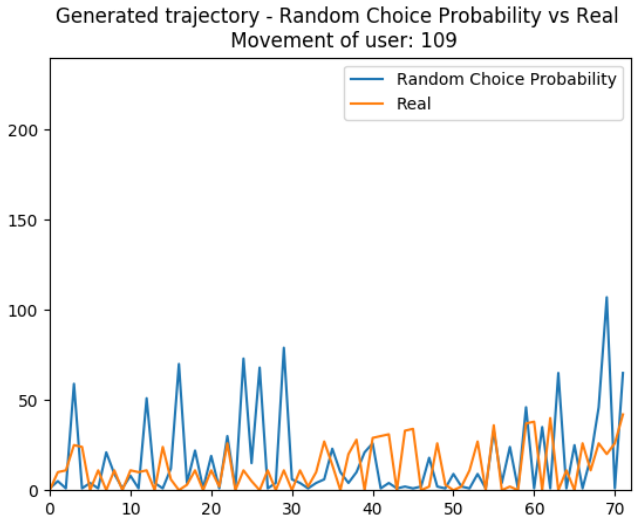

Figure 29: Source: Generate with matplotlib [19]

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.

Best Probability Generator vs Random Choice Probability Generator

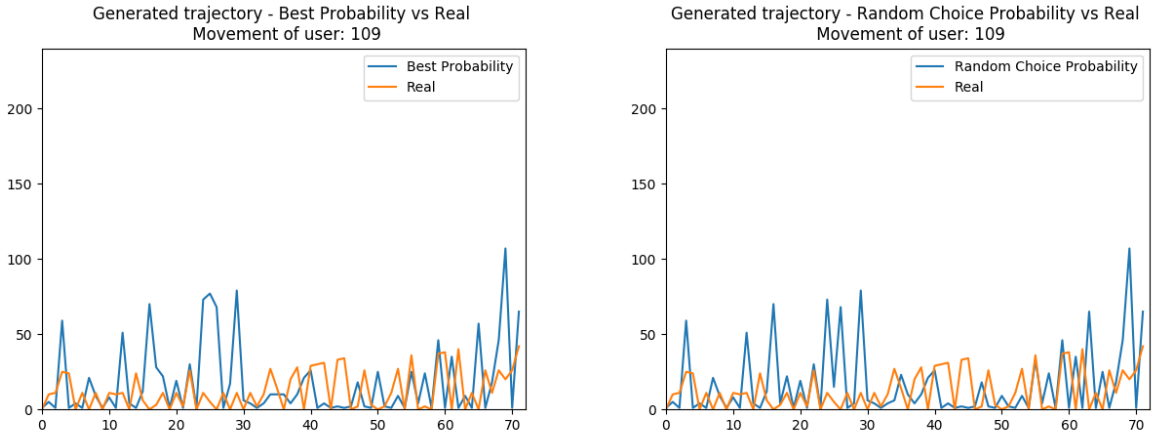

Figure 30: Generate with matplotlib [19]

Then to try to find really a difference I 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 in average in favor of the Random Choice Probability Generator. Indeed this generator continues to have 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.

Best Probability Generator vs Random Choice Probability Generator

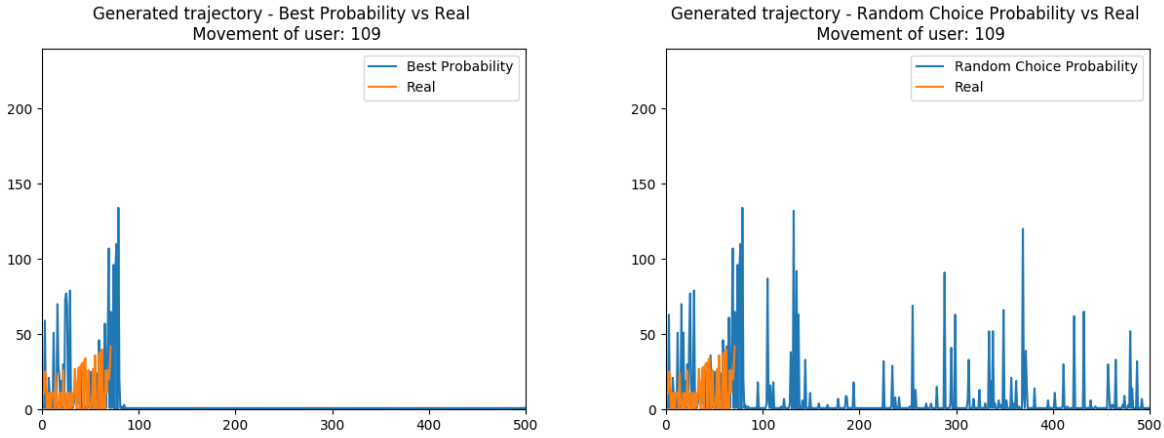

Figure 31: Generate with matplotlib [19]

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 my results also consolidated my thought that some memory of past positions can help for better generate capture behavior of people. Indeed the results on generators based on the 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 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 behaviour without the little magic of the human kind for 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 between 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. But if we need to may be simulate a large number of human movements, for example in traffic simulation, it will be better to use the Random Choice Probability Generator.

10 Discussion

The prediction part of the project has shed light on the fact that even basic models can be used on complex purposes. Even just to have some points of reference and comparisons. Indeed this first part showed us that we need and can adapt our data to fit in model even if it does not seem be configured for it. For instance I transformed the problem of capturing informations based on previous positions by giving another size to my input for the Linear Regression model, Logistic Regression Model and even the Neural Network Model which did not have a parameter as timesteps. By this way I simulated timesteps of my LSTM model and the idea that I had that we needed to capture the human movement behaviors with some history of these movements. We see also that even if basic models seem to try to capture some behaviors as the LSTM did with simple parameters, it would have been very difficult to improve their performance given their lack of evolving parameters. So the LSTM seemed to be a good idea in the fields of predicting and also generating mobility trajectories, especially because of its powerful flexibility and ability to keep memory of past results.

Furthermore during the prediction part or during the optimization using some timesteps clearly improve the results to finally obtain something that really seems to integrate the behavior of users. The LSTM model responds very well when we try to figure out what was the best set of parameters. Indeed it confirms us that larger model with more neurons and hidden layer 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 also increase the performances, but reaching this limit no major good effects occur. Sometimes it even decreases a lot the performance and increases also the training time. If we can clearly say that smaller models have clearly worse results than my best one, it's more litigious with bigger ones. Indeed we can make assumptions that maybe with more epochs during the training part it would maybe increase the performance of these larger models. As a result of their larger size (more neurons or/and more hidden layers) they theoretically need more time to be trained. So may be with more time we could try to increase the number of epochs to see if it's possible to obtain better results with a larger model.

After all during the generation part we clearly confirm that generate path with only random is not imaginable for mobility trajectories. So using something to integrate the real pattern of the human mobility is a key. Indeed the LSTM model confirms us that point. If not yet sufficient to generate perfect synthetic

mobility trajectories, the Best Probability Generator and the Random Choice Probability Generator at least confirm us that it can be possible by looking at the results obtained. So in further works in my opinion we could maybe continue improving even more the model or try by using some other features (time, delta time between movements, or others) in order to perform better and test it on other dataset. Moreover we could also take a look at a model even more complex as Generative adversarial networks (GANs). Indeed GANs seems to be more and more used in complex fields. It basically simply puts two models in concurrences: one that tries to generate fake data and the other one that tries to say if the data given are real or false. So it will maybe be possible to use it in our case to generate synthetic mobility trajectories and even may be to integrate LSTM on it. These are just conjectures, but in further works it would be worth trying it.

11 Conclusion

The high level of movements of people has created new tendencies, new needs, new problems and also new opportunities. Moreover the globalisation of technologies has also provided the society with new opportunites. So a lot of ideas about applications linked to these movements exist or can be created. So the need to obtain data about them exists and at the same time the difficulties to obtain them push us to be creative. It is why it would be interesting to generate the data needed about mobility in order to need no more real one or at least less. So in other terms it is clearly interesting to generate synthetic mobility trajectories. As we have viewed it is also difficult to capture real behavior of humans in order to generate some movements that seem human. That's why we look for some models capable of capturing real behavior of people and the good idea is the machine learning. Especially LSTM models which could be very useful in the context of the importance of memory about past positions to really understand and integrate the movements of people.

It seems that LSTM performs well. The idea of keeping some past informations about the trajectories is clearly confirmed. Indeed if we use some informations about past positions even basic models as Linear Model or Logistic model perform better while working with movements and their predictions. The LSTM model with its flexible configuration and its purpose of capturing long term dependencies works very well for our field. With the tests done we can assume that having 5 timesteps, 5 hidden layers and 50 neurons perform better to generate transitions between points of interest.

In conclusion we can clearly approve the common thought that machine learning can be useful in many areas. In our case it really helps us for trying to capture real behavior of users and to try to summarize them. The present paper shows us that in some points the machine can capture and understand some human behaviour in a context of geospatial trajectories. Even if the present work uses only simplification of the problem by using only sequences of areas, we can see the potential of machine learning, particularly the power of LSTM. It will be really interesting to go further in the context of geospatial trajectories and try to perform better by adding and testing the impact of more features; time, delta time, longitude, latitude, and so one. My work at the moment ends up with a global conclusion that we can now really use computers to analyse, predict, cluster and even capture human behavior. I hope that me or other people will go further and apply a better model on the subject in order to finally create a generator that will be really useful for applications on the fields of synthetic geospatial trajectories.

12 References

[1] Jason BROWNLEE. "Gentle introduction to models for sequence prediction with recurrent neural networks". Adresse : https://machinelearningmastery.com/models-sequence-prediction-recurrent-neural-networks/. consulted 28.12.2017.

[2] Jason BROWNLEE. "Gentle introduction to the adam optimization algorithm for deep learning". Adresse : https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. consulted 28.12.2017.

[3] Jason BROWNLEE. "Linear regression for machine learning". Adresse : https://machinelearningmastery.com/linear-regression-for-machine-learning/. consulted 28.12.2017.

[4] Jason BROWNLEE. "Logistic regression for machine learning". Adresse : https://machinelearningmastery.com/logistic-regression-for-machine-learning/. consulted 28.12.2017.

[5] Jason BROWNLEE. "Mini-course on long short-term memory recurrent neural networks with keras". Adresse : https://machinelearningmastery.com/long-short-term-memory-recurrent-neural-networks-mini-course/. consulted 28.12.2017.

[6] Prathana BURANAJUN, Montalee SASANANAN, and Setta SASANANAN. "Prediction of product design and development success using artifical neural network". page 7, 2007.

[7] CESIUM. "Machine learning time-series platform". Adresse : http://cesium-ml.org/docs/index.html. consulted 05.01.2018.

[8] Bertil. CHAPUIS, Arielle MORO, Vaibhav KULKARNI, and Benoît GARBINATO. "Capturing complex behaviour for predicting distant future trajectories". page 10, 2016.

[9] Kirill FUCHS. "Machine learning: Classification models". Adresse : https://medium.com/fuzz/machine-learning-classification-models-3040f71e2529. consulted 28.12.2017.

[10] João Bártolo. GOMES, Clifton PHUA, and Shonali KRSHNASWAMY. "Where will you go? mobile data mining for next place prediction". page 12, 2013.

[11] Danijar HAFNER. "Introduction to recurrent networks in tensorflow". Adresse : https://danijar.com/introduction-to-recurrent-networks-in-tensorflow/. consulted 28.12.2017.

[12] Erik HALLSTRÖM. "How to build a recurrent neural network in tensorflow". Adresse : https://medium.com/@erikhallstrm/hello-world-rnn-83cd7105b767. consulted 28.12.2017.

[13] Trevor HASTIE, Robert TIBSHIRANI, and Jerome FRIEDMAN. *The Elements of Statistical Learning*, volume 2. Springer, 2001.

[14] G. E. HINTON, N. SRIVASTAVA, A. KRIZHEVSKY, I. SUTSKEVER, and R. R. SALAKHUTDINOV. "Improving neural networks by preventing co-adaptation of feature detectors". page 18, 2012.

[15] Bin JIANG, Junjun YIN, and Sijian ZHAO. "Characterizing the human mobility pattern in a large street network". page 17, 2009.

[16] Andrej KARPATHY. "The unreasonable effectiveness of recurrent neural networks". Adresse : http://karpathy.github.io/2015/05/21/rnn-effectiveness//. consulted 28.12.2017.

[17] Vaibhav KULKARNI and Benoît GARBINATO. "Generating synthetic mobility traffic using rnns". page 4, 2017.

[18] Zachary C. Lipton, John Berkowitz, and Charles ELKAN. "A critical review of recurrent neural networks for sequence learning". page 38, 2015.

[19] MATPLOTLIB. "Matplotlib". Adresse : https://matplotlib.org/. consulted 05.01.2018.

[20] Michael NIELSEN. "Improving the way neural networks learn". Adresse : http://neuralnetworksanddeeplearning.com/chap3.html. consulted 28.12.2017.

[21] Christopher OLAH. "Understanding lstm networks". Adresse : http://colah.github.io/posts/2015-08-Understanding-LSTMs/. consulted 28.12.2017.

[22] MONIK PAMECHA. "A noob's guide to implementing rnn-lstm using tensorflow". Adresse : http://monik.in/a-noobs-guide-to-implementing-rnn-lstm-using-tensorflow/. consulted 28.12.2017.

[23] Sebastian RASCHKA. "What is the role of the activation function in a neural network?". Adresse : https://www.kdnuggets.com/2016/08/role-activation-function-neural-network.html. consulted 05.01.2018.

[24] Warren S. SARLE. "How many hidden units should i use?". Adresse : ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hu. consulted 28.12.2017.

[25] SCIPY. "Numpy.unique". Adresse : https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.unique.html. consulted 05.01.2018.

[26] SCIPY. "scipy.stats.pearsonr". Adresse : https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html consulted 05.01.2018.

[27] Angela L. SWAFFORD. "Logistic population growth: Equation, definition & graph". Adresse : https://study.com/academy/lesson/logistic-population-growth-equation-definition-graph.html. consulted 05.01.2018.

[28] Georgios TECHNITIS and Robert WEIBEL. "An algorithm for random trajectory generation between two endpoints, honoring time and speed constraints". page 6, 2014.

[29] TENSORFLOW. "An open-source software library for machine intelligence". Adresse : https://www.tensorflow.org/. consulted 05.01.2018.

[30] Amey VARANGAONKAR. "Top 10 deep learning frameworks". Adresse : https://datahub.packtpub.com/deep-learning/top-10-deep-learning-frameworks/. consulted 28.12.2017.

[31] WIKIPEDIA. "Wikipedia the free encyclopedia". Adresse : https://en.wikipedia.org/wiki/Linear_regression. consulted 05.01.2018.

[32] WIKIPEDIA. "Wikipedia the free encyclopedia". Adresse : https://en.wikipedia.org/wiki/Artificial_neural_network. consulted 05.01.2018.

[33] Chris WOODFORD. "Neural networks". Adresse : http://www.explainthatstuff.com/introduction-to-neural-networks.html. consulted 28.12.2017.