M.Sc.Eng. Thesis
Master of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Predicting Patterns of Human Mobility

Peter Edsberg Møllgaard (s144256)

DTU 2020

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Richard Petersens Plads
Building 324
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

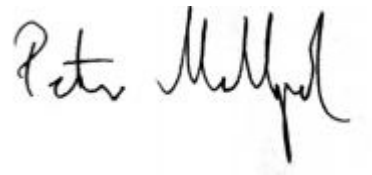
Human mobility prediction provides valuable information in fields like traffic management, disaster response and more. Research in mobility of masses has been thoroughly explored and now, with smart phones, smart watches, cars and other devices accommodating GPS, exploring mobility of individuals has been made possible. An upper bound of predictability of human mobility was early on suggested to be 93% where several studies were able to recognize this with prediction rates above 90%. However, the temporal and spatial resolution of the data has been shown to be an important input and this has called for research that explores the limits of *next location data* instead of continuous location data. Using next location data state of the art networks has gotten prediction rates up to 23.2%.

In this study we explore the predictability of human mobility using the Sensible DTU data set. We show that the theoretical upper limit on mobiliy prediction in this dataset is 65.9% and we train an artificial recurrent neural network model to make actual predictions. Specifically, a *long short term memory* (LSTM) *model* is constructed, which achieves an average *prediction rate* of 34.6%. This performance is obtained by combining embeddings of generalized and individual information across users and by using a novel prediction score that is able to generalize across users. The result is a huge improvement on several baseline models provided in this thesis as well as an improvement to results found in similar work in the litterature (23.2%). We note, however, that a fair comparison of the methods found in the literature to the methods provided in this thesis would require them to be tested on the same data.

Preface

This master thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a master degree in Mathematical Modelling and Computing.

DTU, February 26, 2020



Peter Edsberg Møllgaard (s144256)

Acknowledgments

I would like to thank my supervisor Professor Sune Lehmann for the guidance during this project. He has been supportive and helpful throughout, answering questions and sharing his knowledge to help me shape this thesis. The weekly meetings and feedback helped me overcome challenges met on the way.

I would also like to thank Ulf Aslak, former PhD, who guided me in creating my first neural network, and bounced ideas and methods to tackle problems.

Lastly, to my close family; thank you for the support, both mentally and academically. Discussing crazy ideas and encouraging me to continue doing my work days when motivation were low.

Contents

Summary **i**

Preface **iii**

Acknowledgments **v**

Contents **vii**

1 Introduction **1**
1.1 Overview . . . 1
1.1.1 Human mobility . . . 1
1.2 Literature review . . . 2
1.2.1 Upper bound on the predictability of human mobility . . . 2
1.2.2 Predictability of human mobility . . . 3
1.3 Contribution of this thesis . . . 4

2 The Data Set **7**
2.1 GPS to stop location . . . 7
2.2 Stop locations . . . 8

3 Theory **9**
3.1 Shannon entropy . . . 9
3.1.1 Temporal entropy estimation algorithm . . . 11
3.1.2 Entropy to predictability . . . 11
3.2 Neural Networks . . . 12
3.2.1 Feed Forward Neural Network . . . 12
3.2.2 Recurrent Neural Networks . . . 16
3.2.3 Long Short Term Memory (LSTM) . . . 18
3.2.4 Notes on Neural Network . . . 21
3.2.4.1 Optimizers . . . 22
3.2.4.2 Embeddings . . . 23
3.2.4.3 Weight initialization . . . 23
3.2.4.4 Gradient clipping . . . 24

4 Entropy Estimation **25**

4.1 Entropy convergence 25
4.2 Upper bound 26

5 Data Exploration **29**
5.1 Baselines 34

6 The Network **37**
6.1 Initial network 37
6.2 Final network 41
6.2.1 Prediction vs entropy 51
6.2.2 Exploration prediction 52
6.2.3 Transfer learning 52
6.3 Test set predictions 53

7 Conclusion **59**

Bibliography **61**

CHAPTER 1

Introduction

1.1 Overview

As technology has advanced the last decades so has the amount of data collection in many forms. We are in the age of ”big data”, as expressed in The Economist [1], which opens up a new world of exploration when combined with the growth in computing power. Many algorithms like artificial neural networks [2] were introduced decades ago, but had limited impact for a long time. The success of these algorithms in recent years is to a great extend driven by the immense leap in data and computing power required to be efficient.

Nowadays technology is everywhere and we log nearly everything for safety, practical as well as economical reasons. This has allowed questions to be answered which was previously nearly impossible to approach, e.g. the ”small world problem” which asks about the minimum length of connecting chains in social networks. This problem was attempted to be solved by Milgram et al [3] , in the 1960s, but now with social platforms like Facebook, which has 2.5 billion users[4], the problem was finally answered, with six degrees of separation, i.e. there is a maximum of 6 social links between any human on earth.

1.1.1 Human mobility

Human mobility is a term used to describe the dynamics of humans in space. It is a central part of human behavior that is still poorly understood. With the development of personal devices that has a GPS, such as smart phones, wrist bands, cars, etc, data now exists to explore this challenge. With our society aiming at being as efficient as possible, the ability to predict crowds of people is valuable in fields like traffic management [5], disaster response [6], health monitoring [7], spreading of diseases [8], smart cities [9] and many more areas. In contrast to earlier research, that primarily were focused on mobility of masses, this project will focus on the individual. Different types of data has been used in individual mobility predictions, usually collected from phones. These types includes GPS, wifi network, call detail record (CDR) and travel surveys. CDR is the location of the cell tower which a phone is connected to every time a phone call is made. The different types have different temporal and spatial resolutions with GPS being the most accurate since it is a ”continuous” stream of data collection, whereas the other three depends on phone usage. In this study we

use data which origins from a continuous GPS data stream, but transformed into locations instead of latitude and longitude coordinates as explained in [10], which will be briefly touched upon later.

1.2 Literature review

Human mobility on an individual level was first suggested to follow the patterns of Levy flights, which is a random walk where the jump distance comes from a power law distribution. This was discovered by following how bank notes traveled in the US [11], and shortly after researched again with the use of cell phones data [12]. Here the jump distances was found to follow a truncated power law distribution, which suggested that human mobility was more restricted than the bank notes. The same patterns were confirmed by Rhee, et al. with the use of GPS data [13]. They found that the parameters of the truncated power law depended on the geographical settings, and was not the same for everyone. Additionally they studied the waiting time between each jump which was also suggested to follow a truncated power law. However, Zhao et al. found that power laws may not be the best way to represent travel in space by studying different transportation modes; walking, train, driving and biking [14]. They proposed a log-normal distribution as a better fit to travel distances compared to the truncated power law.

Unsurprisingly, humans do not move around totally at random. We move somewhere with a purpose (e.g. work, social activities, sport, etc.). It was shown by Song et al. that the probability of revisiting a location is proportional to the amount of previous visits [15]. Furthermore, when looking at social activities the time between meetings of acquaintances was found to follow a heavy tail distribution similar to a truncated power law (we meet again) [16]. Not surprisingly it was found that those who call each other tended to share the same location [17], i.e. people who share locations are more likely to be related. Wang et al. turned the table and tried to predict future friends by looking at the trajectory of individuals, which showed great predictive power [18].

1.2.1 Upper bound on the predictability of human mobility

Presently, the prediction of human mobility is often separated into two challenges; (1) finding a theoretical upper bound and (2) making actual predictions. Finding an upper bound has been explored several times with varying results. It was first estimated by Song et al. to be 93%, [8], followed up by Lin et al. who estimated the upper bound to be around 90% [19]. Song et al. did this by logging the connected cell tower of phones every hour, which gave an average resolution of 3 km^2. This lead to questions about the temporal and spatial resolution. An investigation into the spatial resolution was explored and lowered the upper bound to around 81%-85% by Smith et al. [20] and Lu et al. [6]. The temporal resolution however still raised a challenge

and, as Ikanovic et al. suggested, the lower and upper bound of predictability must follow

$$\Pi^{(max,min)}(\Delta s, \Delta t) \to 1 \quad \text{when} \quad \Delta s \to \infty \quad \text{and} \quad \Delta t \to 0, \tag{1.1}$$

where $\Pi^{(max,min)}$ is respectively the upper and lower bound for the prediction, Δs is the spatial resolution and Δt the temporal resolution [21]. This means that if the temporal resolution is very low, i.e. short time intervals between readings, and the size of locations is high then the upper and lower bound of predictability is 100%. E.g. if the spatial location was the solar system, then we would never leave that location. And if we had 1 billion readings a second, the probability of you leaving that location in the next instance is infinitely small. To deal with the temporal resolution the approach of looking at the next location was explored [22, 23]. This meant that a sequence like $[xxxxxxyyyzzz]$ would be reduced to $[xyz]$, where x, y, z are arbitrary locations. Both Cuttone et al. [10] and Ikanovic et al. [21] found that people tended not to move very often which further emphasized the need of next location predictions to avoid the problems with the resolutions. Furthermore, they derived a natural length scale for human locations to deal with the arbitrariness of the spatial resolution.

1.2.2 Predictability of human mobility

We now turn our focus towards the task of making actual predictions rather than just finding upper bounds. Several models have been explored, e.g. non-linear time series [24], naive Bayes [25], Markov chains [26], Gaussian mixture models [22], principal component analysis (PCA) [27] and neural networks [28, 29]. Some reached prediction rates over 90%, but as discussed earlier the temporal and spatial resolutions are at play and makes the predictions differ between models and data sets. Next location prediction has also been attempted and Gao et al. took the most recent suggested algorithms, for next location prediction, and compared them against their own [28]. The better performing models were all neural networks. They used four different data sets from New York, Singapore, Houston and California. The best Top@k prediction rate for each data set is shown in Table 1.1, where a Top@k prediction rate is the rate of which the true location is in the k highest suggestions from a model. These results are all generated by Gao et al.'s best performing algorithm; their own variational attention model. It should be noticed how different the prediction rates are depending on the data set. This is crucial to keep in mind when comparing models.

	Top@1	Top@5	Top@10
New York	23.21%	41.03%	47.45%
Singapore	12.13%	24.78%	30.69%
Houston	7.69%	16.98%	21.36%
California	8.35%	17.16%	20.31%

Table 1.1: Table showing the best Top@k predictions rates for 4 different data set, presented in [28].

1.3 Contribution of this thesis

In this thesis we will further explore individual mobility prediction using neural networks, specifically long short term memory (LSTM) models, using GPS data from the Sensible DTU data set (explored in chapter 2). We will use the transformed GPS location data and only look at the next location in order to avoid the challenges with the temporal and spatial resolution. We will compute a theoretical upper bound on the predictability of mobility for the Sensible DTU data set. We will estimate the entropy of the individual trajectories and use this estimator as a tool to decide which trajectories are long enough to consider for predictions. The overall structure of the thesis is as following:

Chapter 2 - An introduction of the Sensible DTU data set. In this section we introduce the Sensible DTU data set, looking at where it was collected, who is included and what the data set contains.

Chapter 3 - A section covering the used theory, split up into two main parts:

- An introduction to Shannon entropy, how it is estimated and how it is turned into a theoretical upper bound of predictability.
- A coverage of the foundation and motivation behind neural networks. Contains the theory of basic feed forward neural networks, basic recurrent neural networks, and finally the LSTM network.

Chapter 4 - Entropy estimation and check of convergence for each trajectory in the Sensible DTU data set. This is used to remove sequences from the data set that has not converged to ensure data quality. We also look at the theoretical upper bound of predictability of the data set.

Chapter 5 - An exploration of the data set. We go through a general exploration of the data looking at distributions, trends, etc. We also explain the choices behind the pre-processing of the data set, and how we split up the data in training, validation, and test sets. Lastly we present six baseline models to get an idea of the lower bound of predictability and what we aim to surpass.

Chapter 6 - A presentation of the network and motivation behind choices. Here we start from the initial idea of our network, and describe the process to our final network. We present the prediction rates and explore the embedding space of the final network.

Chapter 7 - Conclusion of the findings. We conclude and discuss how the networks have performed relative to the baselines and previous work, and look into the findings of exploration predictions and the embedding space.

A repository containing the code can be found here. The data is not included due to the data sensitivity.

CHAPTER 2

The Data Set

In this study we use data from the Sensible DTU project also known as the Copenhagen Networks Study [30]. In that project more than 800 smartphones (Google Nexus 4) were given out to students at The Technical University of Denmark (DTU) in September and during the following months in 2013. Around half of the phones were handed out to freshmen, whereas the other half were handed out to undergraduate students. A year prior Sune Lehmann, professor at DTU Compute, had been granted 200 smartphones which provided an exploitative basis to sort our potential technical and unforeseen difficulties before receiving the remaining smartphones.

Every participant receiving a smartphone had to fill out an online questionnaire made by people at the University of Copenhagen from the departments of anthropology, health, psychology, and economy. In this questionnaire most participants also agreed to let information from different apps, such as Facebook, to be recorded. The purpose was to capture several factors about the individual which could prove important for their behavior like the Big Five Inventory (BFI) [31]; Openness, Conscientiousness, Extraversion, Agreeableness, Neuroticism (OCEAN) and many more.

The Sensible DTU dataset includes information about calls, texts, GPS coordinates, Bluetooth signals, and screen activity. Every instance of captured data includes a timestamp as temporal interactions is an important factor for the experiment. Since it is personalized data it is not public and authorization is needed to work with the data set.

This thesis will focus on pre-processed data from the raw GPS data from individual users. The raw GPS data includes a latitude and longitude coordinate for each time step which can be converted into a series of locations when the user moves from one stationary location to another.

2.1 GPS to stop location

The conversion from GPS data to locations was done by Cuttone et al., where they tried to eliminate the problem of self-transitions, and only include data points when an individual changed location [32]. Cuttone et al. defined a stop as a location where the following location were not further away than some threshold, σ, in a given time interval from time t to $t + \Delta t$. The threshold was chosen to be $\sigma = 50m$ since that approximated the size of a building and if the location at time $t + \Delta t$ were not within the threshold, a new stop was then detected.

The centroid of each stop sequence was then set to the median of the locations' latitude and longitude readings. Furthermore to remove uninteresting stops, i.e. waiting for the buss etc., they decided to only consider stops longer than 15 minutes. Lastly they used the clustering algorithm DBSCAN, [33], to cluster the stops together with the distance parameter $\epsilon = 50$m and minimum amount of points $min_pts = 2$. This length scale was used as an approximation for the size of a building.

2.2 Stop locations

Every user now has a sequence of stop locations with a corresponding sequence of timestamps for arrival and departure of the given location. A random piece of user data could be:

$$\begin{aligned} \text{stop_loc} &= [1, 2, 3, 2, 5, 1, 2, 3, 1, 2, ...] \ , \\ \text{time} &= [(1572296587, 1572296598), (1572299587, 1572299887), ...] \ . \end{aligned}$$

The time is shown in unix time which is the amount of second since the 1st of January, 1970 UTC time. Since the data only shows changes in locations the sequence will never have sequentially repeated locations.

A hierarchical stop location sequence is also available where every location is within some bigger cluster containing other locations. The amount of "higher" cluster differs from user to user, but is between 2 and 5. In other words; if we had an user with 3 hierarchies of clusters, a sequence could be; [[1,3,5],[1,3,2],...] where the first location could represent [Denmark, Copenhagen, Tivoli]. Each id is unique over the higher orders of the hierarchical representation. This means that if Tivoli is id 5, there will be no other id 5 even outside Copenhagen or Denmark. This conversion was done by Alessandretti et al, [34], but the paper is unpublished as of the writing of this thesis.

CHAPTER 3

Theory

This chapter will first introduce Shannon entropy which in short is a measure of order in data. It has been proven that given sequential data this measure can be converted into an upper bound of predictability of next state predictions, which is useful knowledge when trying to do actual predictions [8]. Afterwards a throughout explanation is made of the steps going from vanilla feed forward neural network, to recurrent neural networks (RNN), and up til recent long short term memory (LSTM) neural networks. Some extra notes on the optimization of the training is also mentioned.

3.1 Shannon entropy

This section introduces the definition and motivation of Shannon entropy, together with an estimation algorithm. This is particular useful since it is proven that the Shannon entropy can be converted into a theoretical upper bound of predictability [8], which we can use to get an idea of the performance of our predictions. It is also useful to see when/if the entropy of the individual sequences has converged, so that we can remove the ones that are not converged from our data set, to make sure the sequences contains enough information.

Entropy is a term used in many fields with different definitions, but represents a common concept of how disordered a system is or how the energy is distributed [35]. The higher the entropy the more disordered the system and vice versa. The concept of entropy was first discovered when investigating the loss of energy when studying engines [36]. The concept of entropy in information theory was introduced by Claude Shannon in 1948, and is the theory to be used in this project [37]. From now on we will refer to Shannon entropy as just entropy.

Today information is stored in computers using bits which is either zeros or ones, and since space is not infinite, the minimal way to represent data is important. The entropy of a system may intuitively be understood as the average amount of bits that needs to be shared to describe the system state. An equivalent formulation is to view entropy as the average number of yes/no questions that needs to be answered to obtain the system state.

The mathematical formulation of the entropy is

$$S = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i), \tag{3.1}$$

where $P(x_i)$ is the probability of state x_i and n is the total amount of states. Since bits only has two possible values the logarithm with base 2 is used. Had there been more possible values the entropy would have been lower since we could represent information more compact.

Another way to think of entropy is as how much information do we get on average from an observation. Lets say we have a system of four possible states, $x \in [A, B, C, D]$. If all the states had the same probability the entropy would be $-4 \cdot \frac{1}{4} \log_2 \frac{1}{4} = 2$. However if the system had the state probabilities $P(A) = 1/2$, $P(B) = 1/4$ and $P(C) = P(D) = 1/8$ then the entropy would be 1.75. Lets assume that these states represented the weather. If the states of the weather is skewed towards a small number of states, then on average we need less bits to communicate its state, since we can use short sequences of bits to represent the common states. Note that giving each state the same probability will always result in the maximum entropy. This can be shown in many ways, one of which is forming the Lagrangian and solving. If, however, it was known that state A always happened, i.e. $P(A) = 1$, then the entropy would be 0, since we would need to send 0 bits on average to communicate its state.

When working in the real world the actual probabilities are very rarely known. Therefor an approximation of the probabilities for each state is needed to estimate the entropy. Luckily we are often dealing with a finite amount of states, especially in information theory, which makes the problem a bit easier. An easy upper bound of the entropy can therefor always be computed by giving each state the same probability, calculated as follows

$$S^{rand} = -\sum_{i=1}^{N} \frac{1}{N} \log_2 \frac{1}{N} = \log_2 N, \tag{3.2}$$

where N is the amount of possible states. However, this is a measure of random guessing, which does not use any temporal information other than the states. To introduce some temporal information we make a crude assumption that the probability of every state is uncorrelated of the previous state, $P(x_i|x_{i-1}) = P(x_i)$ for $\forall i$. If this was the case, then the uncorrelated entropy would be

$$S^{unc} = -\sum_{i} P(x_i) \log_2 P(x_i), \tag{3.3}$$

where $P(x_i)$ is the amount of times state x_i has occurred out of the total amount of observations. This is the entropy of a location event unconditioned on all previous events. However, we are interested in the entropy of a location event conditioned on the whole history of events leading up to it. This is something entirely different since the next event is highly correlated with the preceding events. We can therefor use

the information provided by the preceding states to say something about the next state, thereby lowering the entropy of the next state setup compared to S^{unc}. Given a trajectory of locations $T = \{x_1, x_2, \cdots, x_N\}$, then including the temporal correlations the entropy of the trajectory is estimated as

$$S^{temp} = -\sum_{T' \subset T} P(T') \log_2 P(T'), \tag{3.4}$$

where $P(T')$ is the probability of finding the exact sub sequence T' in the trajectory T. To motivate the temporally correlated entropy, three different estimations of the entropy has been introduced, where it must be true by construction that $0 \leq S^{temp} \leq S^{unc} \leq S^{rand} < \infty$. It is not possible to obtain the exact temporal entropy without infinite data, but we can get a good estimate when the sequence is long compared to the dynamics. The sequence is determined to be long enough if smaller subsets of the sequence yield the same entropy estimate.

3.1.1 Temporal entropy estimation algorithm

An algorithm which can be used to estimate the temporal entropy is now introduced. The algorithm is based upon a compression method called Lempel-Ziv compression [38, 39]. The algorithm has been proven to converge towards the actual entropy of a time series given enough data. The estimation is computed as

$$S^{temp} = \left(\frac{1}{n} \sum_i \Lambda_i \right)^{-1} \ln n, \tag{3.5}$$

where n is the length of the series and Λ_i the length of the shortest sub-string from i that has not happened prior to i. As $n \to \infty$ the estimator is proven to converge towards the actual entropy [39].

3.1.2 Entropy to predictability

Now that an estimation of the entropy is known, a conversion into an upper bound of predictability is desired. This can be achieved using Fano's inequality, [8], to get

$$S_i = -[\Pi_i \log_2 \Pi_i + (1 - \Pi_i) \log_2 (1 - \Pi_i)] + (1 - \Pi_i) \log_2 (N - 1), \tag{3.6}$$

where Π_i is the upper bound of predictability, N the is number of unique locations and S_i is the entropy for trajectory i. This equation is transcendental, which means it cannot be solved algebraically. Instead the roots are found numerically of the following equation

$$0 = -[\Pi_i \log_2 \Pi_i + (1 - \Pi_i) \log_2 (1 - \Pi_i)] + (1 - \Pi_i) \log_2 (N - 1) - S_i. \tag{3.7}$$

3.2 Neural Networks

An estimation of the entropy has been introduced with a corresponding theoretical upper bound of predictability. However, we are interested in doing actual predictions, specifically using neural networks. This section will introduce the fundamentals of neural networks. This includes how data affects the architecture and choices of the specific network, how they are trained and tricks to avoid different ”traps”. We will specifically go from vanilla neural networks where the network does not care about temporal structure in the data, to recurrent neural networks which takes time dependencies into account and then specifically long short term memory models which is an extension of the recurrent neural networks.

Deep learning in general has been growing immensely the last few decades due to the advancement in computational power of machines and the expanding size of data sets. This growth is especially due to GPU/TPU's which are very efficient at performing tensor operations. More data effectively means that models needs less specificity since the data explores the space itself which allows models to have less assumptions and just rely on the input data. More data however makes bigger models possible which requires more computing power.

3.2.1 Feed Forward Neural Network

A neural network (NN) is often motivated by how a brain functions with neurons and edges which carries the information through the system. The simplest NN is a feed forward NN where information is only carried one way as in Figure 3.1. This network assumes four input attributes and predicts two output attributes, e.g. given height, weight, age and gender it predicts body fat percentage and muscle mass. From left to right we have:

1. The input layer which is the data.
2. A hidden layer.
3. The output layer which gives the prediction.

But what does the network actually do? For every hidden layer (only one hidden layer in the example but more could be added) the nodes have an activation function attached. The weighted sum of values from all the downstream nodes goes into these activation functions, which then continues in the next layer. Common activation functions are the Sigmoid function, Hyperbolic tangent, ReLU and Softmax to name a few. The purpose of the activation function is to act as a ”gate” between the input of a node and its output forwarded to the next layer, or just to transform the data in a meaningful way for the network. Two important attributes of activation functions are that they need to be differentiable, almost everywhere (which will be elaborated on when discussing back-propagation), and non-linear. If they are not non-linear then no matter how many layers it would be equivalent to a single layer network. It should

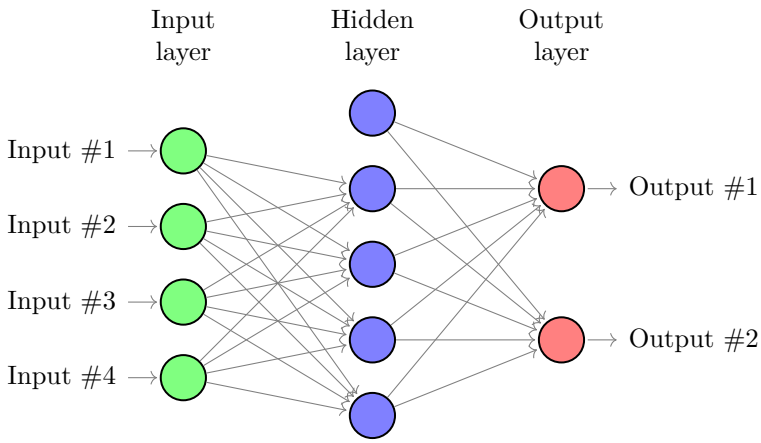


Figure 3.1: Feed forward neural network.

be noted that each node also has a bias term. Often the bias term is shown as an extra node with value 1 and only outgoing edges.

For every node in a given layer there is an edge connecting it to nodes in the next layer (except for the output node), and with each edge a scalar is associated. Every time a value runs along an edge it is multiplied with this scalar. Given a toy example, Figure 3.2, where the activation function of the hidden layer is the hyperbolic tangent (tanh) and the activation function of the output layer is the Sigmoid function, the calculation would be

$$Output = Sigmoid(tanh(x \cdot w_1 + b_1) \cdot w_2 + b_2) \tag{3.8}$$

where x is the input, w_1 and w_2 are the edge values and b_1 and b_2 are the bias terms.

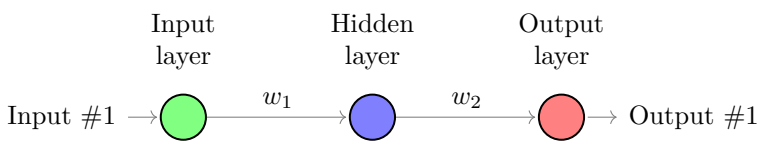


Figure 3.2: Simple feed forward neural network.

Now that the feed forward NN has been introduced, lets see how it is trained. The weights, w_i, needs to be updated and for that backpropagation comes into play. Backpropagation is a technique to update the weights to ensure that an error is minimized. However, in order to do that the error needs to be defined, which is also called the loss function. Just like activation functions many loss functions exists. The primary job for the loss function is to define the error for our problem. A common

loss function for continuous outputs is the mean squared error

$$MSE = \frac{1}{2}\sum_i \frac{(y_i - t_i)^2}{N}, \tag{3.9}$$

where y_i is the prediction and t_i is the target. For classifiers cross entropy is often used

$$H(p,q) = -\sum_i p(x_i)\log q(x_i), \tag{3.10}$$

where $p(x)$ is the probability distribution from the data and $q(x)$ is the model probability distribution. This measure is especially useful when classifying using one-hot encodings since $p(x) = 0$ for $\forall x \neq x_{target}$ and $p(x_{target}) = 1$ which simplifies the summation to $H(p,q) = -\log q(x_{target})$. We will be using the cross entropy as loss function in this thesis, since we are concerned with a classification problem.

Cross entropy provides a nice loss function, since minimizing it corresponds to maximizing the log-likelihood of the data, as seen in Equation 3.11.

$$\log \prod_i q_i^{p_i} = \sum_i p_i \log q_i = -H(p,q). \tag{3.11}$$

The cross entropy is minimized when $p = q$, which means that by minimizing the loss we find the model q that best reproduces the data distribution p. This can be showed by using Lagrange multipliers. The important properties of a cost function is that it can be written as an average over loss functions for individual training samples, and that it has to be differentiable such that backpropagation can be used. This is important as backpropagation is done on every training sample and then averaged to get the actual correction step (the step that updates the weights).

Backpropagation uses the fact that the negative gradient of a function always points towards the direction which minimizes the function in that particular spot. So by iteratively taking small steps in the direction of the negative gradient we approach a local minimum of the function, which - if we are lucky - is also the global minimum. Methods are here used to try to avoid getting stuck in local minimas, which will be elaborated on below. Computing the gradient of the loss function, with respect to all the weights, w, and biases, b, is a bit more complicated since the network could be very complex. However, the chain rule helps us to overcome this problem. We use the following notation:

- C is the loss function.
- w^l_{jk} is the weight from the kth neuron in the $(l-1)^{th}$ layer to the jth neuron in the lth layer.
- a^l_j is the activation function in the jth neuron in the lth layer.

- z_j^l is the linear transformation of the output of the previous activation function in the jth neuron in the lth layer, calculated as

$$z_j^l = b_j^l + \sum_i w_{ji}^l a_i^{l-1}. \tag{3.12}$$

To calculate the gradient we need the partial of the lost function for each weight. Looking at w_{25}^2, as shown in Figure 3.3, the partial of the loss function, C, with respect to w_{25}^2 would be computed as

$$\frac{\partial C}{\partial w_{25}^2} = \frac{\partial C}{\partial a_2^2}\frac{\partial a_2^2}{\partial z_2^2}\frac{\partial z_2^2}{\partial w_{25}^2}. \tag{3.13}$$

Example: Go back to our toy example in Figure 3.2, and let us use the squared error as our loss function. To express the network in the presented notation we have $a_1^1 = \tanh$, $a_1^2 = Sigmoid = \sigma$, and $C = \frac{1}{2}(y-t)^2$. We want to compute the gradient and since we are going backwards we start with $\frac{\partial C}{w_{11}^2}$. Using the chain rule we get

$$\frac{\partial C}{\partial w_{11}^2} = \frac{\partial C}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{11}^2}. \tag{3.14}$$

We now compute each partial derivatives of (3.14)

$$\frac{\partial C}{\partial a_1^2} = \frac{\partial}{\partial a_1^2}\left(\frac{1}{2}(a_1^2 - t)^2\right) = (a_1^2 - t), \tag{3.15}$$

$$\frac{\partial a_1^2}{\partial z_1^2} = \frac{\partial}{\partial z_1^2}\left(\sigma(z_1^2)\right) = \sigma(z_1^2)(1-\sigma(z_1^2)), \tag{3.16}$$

$$\frac{\partial z_1^2}{\partial w_{11}^2} = \frac{\partial}{\partial w_{11}^2}\left(a_1^1 w_{11}^2 + b_2\right) = a_1^1. \tag{3.17}$$

Inserting this back into (3.14) then gives

$$\frac{\partial C}{\partial w_{11}^2} = (a_1^2 - t)\sigma(z_1^2)(1-\sigma(z_1^2))a_1^1. \tag{3.18}$$

We do the same for w_{11}^1, starting again with the chain rule

$$\frac{\partial C}{\partial w_{11}^1} = \frac{\partial C}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial a_1^1}\frac{\partial a_1^1}{\partial z_1^1}\frac{\partial z_1^1}{\partial w_{11}^1}. \tag{3.19}$$

Here we see that we can reuse a lot of the partial derivatives from earlier. We only

need to compute $\frac{\partial z_1^2}{\partial a_1^1}$, $\frac{\partial a_1^1}{\partial z_1^1}$ and $\frac{\partial z_1^1}{\partial w_{11}^1}$

$$\frac{\partial z_1^2}{\partial a_1^1} = \frac{\partial}{\partial a_1^1}(a_1^1 w_{11}^2 + b_2) = w_{11}^2, \tag{3.20}$$

$$\frac{\partial a_1^1}{\partial z_1^1} = \frac{\partial}{\partial z_1^1}(\tanh(z_1^1)) = 1 - \tanh^2(z_1^1), \tag{3.21}$$

$$\frac{\partial z_1^1}{\partial w_{11}^1} = \frac{\partial}{\partial w_{11}^1}(x w_{11}^1 + b) = x, \tag{3.22}$$

$$\tag{3.23}$$

which gives us

$$\frac{\partial C}{\partial w_{11}^1} = (a_1^2 - t)\sigma(z_1^2)(1 - \sigma(z_1^2))w_{11}^2(1 - \tanh^2 z_1^1)x. \tag{3.24}$$

Now that we can compute the gradient we know which direction will minimize the error in that particular spot.

As we just saw in the example the earlier layers can reuse a lot of the computations, which helps the algorithms to do backpropagation without brute forcing it. In practice NN libraries, like Tensorflow[40] and PyTorch[41], have built in backpropagation so the user does not have to implement the derivatives, etc. by themselves.

Every time when the gradient is computed and the network takes a step, it is not necessarily a full step in the direction of the negative gradient. Before taking a step the negative gradient is multiplied with a scalar, called the learning rate, to regulate the step size. With this the updated weight, $\tilde{w}_j$, can be expressed as

$$\tilde{w}_{jk}^l = w_{jk}^l - \lambda \frac{\partial C}{\partial w_{jk}^l}, \tag{3.25}$$

where λ is the learning rate. The learning rate is a parameter for how fast the network learns, but also ensuring the networks convergence. It is a value that differs with every problem, and is usually picked in the range $\lambda \in (0, 1]$.

3.2.2 Recurrent Neural Networks

A normal feed forward network does not care in which order the data is passed through the network. All the data is passed through, and then an average of the loss is computed and back propagated which leads us to another type of neural networks, recurrent neural networks (RNN). As the name suggest RNNs are networks which are computed recurrently. Given data which has sequential importance a network which can utilize the order is desired. An intuitive example is the alphabet. When a human sees a sequence "ABCDEFGH" and asked what the next letter might be, "I" is the obvious answer, but this is only due to the sequential structure of the string. Had

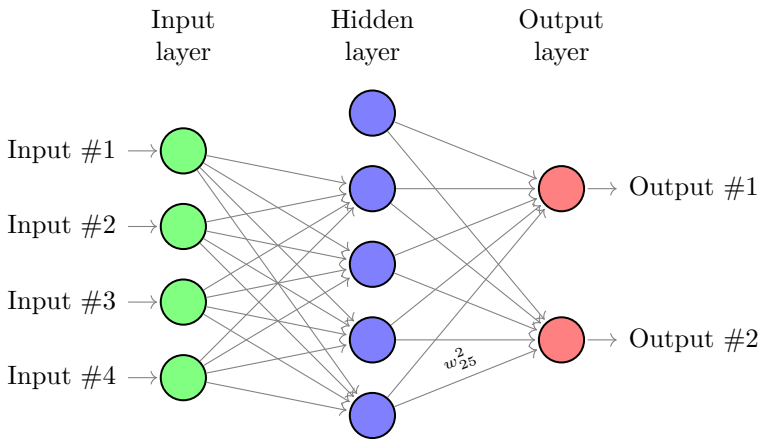


Figure 3.3: Feed forward neural network.

the letters been randomly swapped like "FCBAHGDE" it is not so obvious anymore. The overall structure of a RNN is as shown in Figure 3.4 where x_t is the input, h_t is the state and A is the actual network.

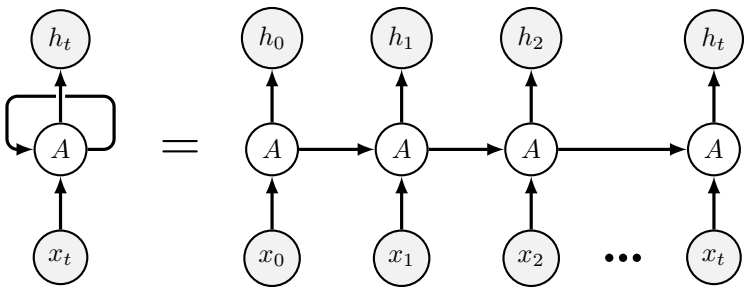


Figure 3.4: Recurrent neural network.

Overall Figure 3.4 shows how the network reuses information from the previous time step. A vanilla RNN is then when the network, A, is as in Figure 3.5. It basically consists of a weighted sum between the input and the previous state, and a feed forward neural network into an activation function, hyperbolic tangent (tanh), which then gives the output/hidden state for the next RNN cell. Here the activation function tanh is important to avoid exploding values. For example, lets assume that the weights in the network is multiplied with 3 each step, then the values grows exponentially fast which is not of interest. The tanh function ensures the values stay in the interval $[-1, 1]$. The goal with this network architecture is to remember previously seen information, the hidden state, while guessing on new information. However, in practice during gradient descend, backpropagation has to go through long chains

of computations (given long sequences of data) and this introduces a new problem called the vanishing gradient. As discussed when talking about backpropagation the derivative of a weight, w^l_{jk}, early in the network depends on the derivative of all the other weights, activation functions, etc., which uses the output of the given weight. This means that when computing the gradient of the loss function relative to that weight, w^l_{jk}, and using the chain rule one end up with a lot of scalars multiplied together

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial a^l_j}\frac{\partial a^l_j}{\partial z^l_j}\frac{\partial z^l_j}{\partial w^l_{jk}}, \tag{3.26}$$

where $\frac{\partial C}{\partial a^l_j}$ needs to be expanded again. In short the calculation ends up being similar to $\nabla = x_1 \cdot x_2 \cdot x_3 \cdot \dots \cdot x_n$ for some high $n \in \mathbf{N}$. When multiplying a lot of numbers together the risk of getting a number which is either close to 0 or infinity increases, thus the terms vanishing gradient or exploding gradient. Recurrent networks are special in that the same recurrent unit is used to calculate a gradient over and over as we propagate the gradient backwards. To avoid exploding gradients we build networks where the average multiplicative factor of each backpropagation step is below 1. However, this also means that the backpropagation signal dies out in an exponential fashion, which is why the loss at some time step can only be connected to the inputs a few steps back. The trained network therefore only has ”short term memory”.

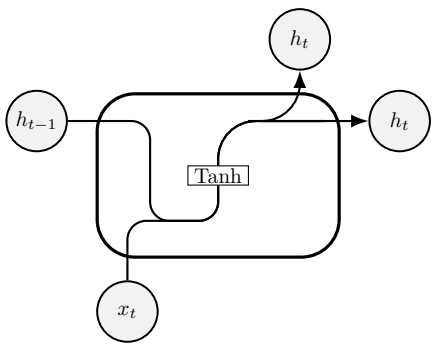


Figure 3.5: Recurrent neural network cell for a vanilla network.

3.2.3 Long Short Term Memory (LSTM)

To combat the problem of short term memory of vanilla RNNs, primarily two new types of networks has been developed, Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU). The GRU and LSTM networks are very similar. Where LSTM uses three gates (as explained further below) GRU only uses two, which makes it computationally faster to update. Chung et al. compared the GRU architecture

against LSTM and others and found that GRU had close if not same performance as the LSTM architecture [42]. We choose to go with the LSTM architecture in this project and introduce the ideas behind the architecture more thoroughly.

LSTM were first introduced by Hochreiter and Schmidhuber in 1997 but has since been refined and become popular due to its high performance [43]. The architecture of a LSTM cell can be seen in Figure 3.6. This network is simply put into the network node, A, in Figure 3.4.

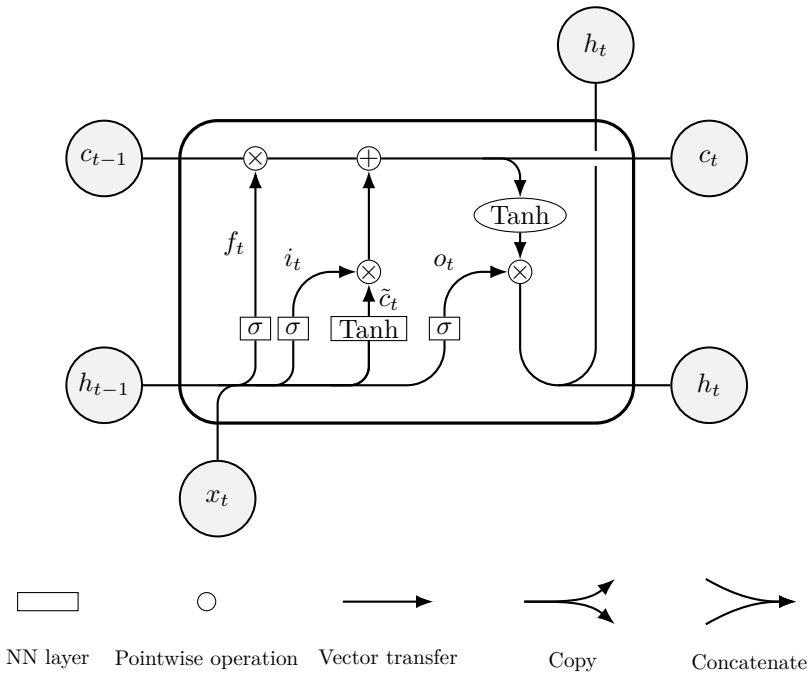


Figure 3.6: Long Short Term Memory (LSTM) cell.

A new state c_t passing through time t is introduced, and often called the cell state. As seen from Figure 3.6 the information from the cell state only goes through a few linear transformations from c_{t-1} to c_t and acts like a highway through the network carrying the information from earlier. In short the network can be separated into 4 steps; forget gate, input gate, cell state, and output gate.

Forget gate is the first step as outlined in Figure 3.7, where f_t is this steps output stated in (3.27). Here the aim for the network is to choose what information to forget from the previous cell state. It is concatenating the previous hidden state with the new input and then running it through a NN sigmoid layer. The sigmoid function will produce values between 0 and 1, i.e. choosing to forget or remember previous

information.

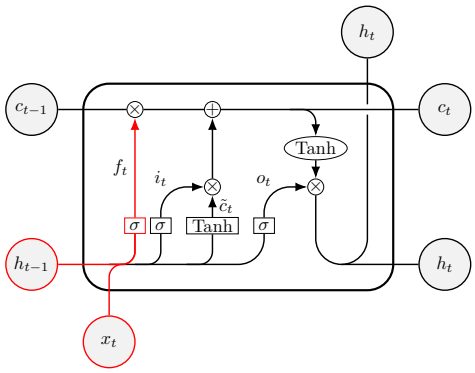


Figure 3.7: LSTM cell with outlined forget gate.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.27}$$

The second step is the *input gate*, see Figure 3.8, where i_t and $\tilde{c}_t$ is this steps outputs stated in (3.28) and (3.29). The purpose of this gate is to figure out which new information should be added to the cell state. First the input gate layer i_t is computed which tells the network what part of the cell state we want to update. Secondly the actual new information $\tilde{c}_t$, that could be added to the cell state, is computed.

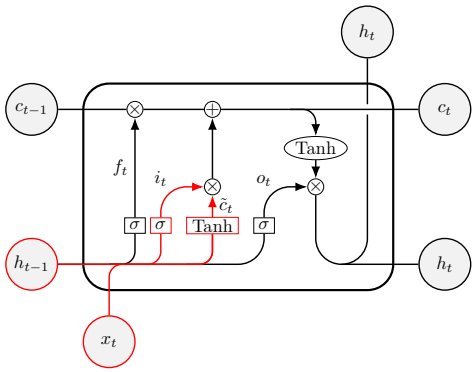
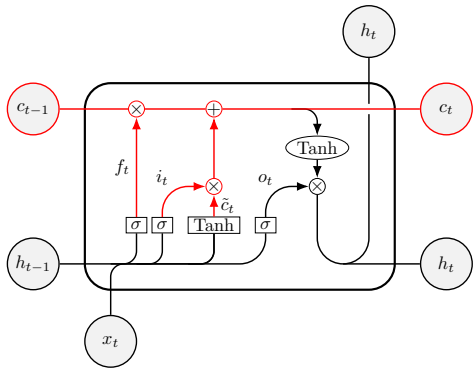


Figure 3.8: LSTM cell with outlined input gate.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.28}$$

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{3.29}$$

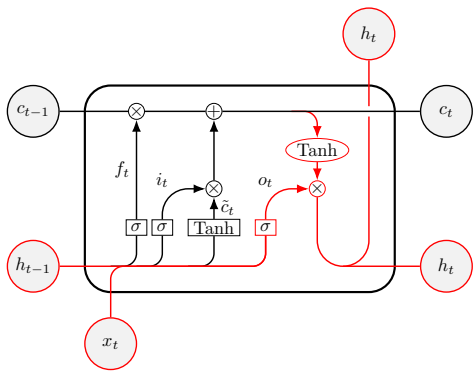
The third step is actually updating the *cell state*, see Figure 3.9, where c_t is this steps output stated in (3.30). This is where the network combines the two previous steps; the first step is point-wise multiplied on the output from the forget gate to forget irrelevant information, and the second step adds new information from the input gate.



$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{3.30}$$

Figure 3.9: LSTM cell with outlined cell state.

Lastly the fourth step is the *output gate* which produces the actual output from the network, see Figure 3.10. Here o_t and h_t is this steps output stated in (3.31) and (3.32). The output is based of the cell state, compressed through the point-wise tangent hyperbolic function and then point-wise multiplied with the output of the sigmoid function of the input data to select which information is relevant to pass on.



$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.31}$$
$$h_t = \tanh(c_t) \odot o_t \tag{3.32}$$

Figure 3.10: LSTM cell with outlined output gate.

The weights are updated just as shown earlier, for a normal NN, but with way more steps.

3.2.4 Notes on Neural Network

There is a large number of network models and training methods available and figuring out what works best for each problem is an art by itself. In this section the we explain the methods used in this thesis to train our neural networks.

3.2.4.1 Optimizers

Doing backpropagation can be tough due to the immense amount of computing. Imagine 1.000.000 data points running through a network with 10.000 weights. That means that the network would have to compute 10.000.000.000 derivatives to take one step. This is not feasible, and therefor optimizers have been developed with the purpose to speed up the process. The most known optimizer is the stochastical gradient descent (SGD), which basically only computes the derivatives for one data point and then takes a step. SGD is often combined with batches which are subsets of the data with a size defined by the user. The data is split into batches and then parsed one batch at a time. The network will here compute the derivative for all the points in that batch and then take a step based on the average gradient. The hope is that a subset of the data is representative for all of the data, and that the model learns faster by taking more steps with subsets of the data. Approximating the gradients with batches also has the extra benefit that models can escape local minimas. Running through all batches in a data set once is called an epoch.

One of the most used optimizers at the time of writing is the ADAM optimizer [44]. ADAM combines RMSprob (explained in a moment) and SGD with momentum. Momentum is used when dealing with a complex problem, since the gradient might change a lot locally, which makes the steps feel random even though there is an overall structure. Imagining a mountain side. It is clear which way is down, but with all the small rocks on the way the gradient locally changes back and fourth. Therefor momentum is introduced which basically keeps the momentum from previous steps to ensure a smoother descent. Same goes if a flat surface is encountered. Here the momentum should help coming out, instead of getting stuck and taking very small steps. The math is

$$v_t = \gamma v_{t-1} + \lambda g_t, \tag{3.33}$$

$$w_t = w_{t-1} - v_t, \tag{3.34}$$

where v_t is the accumulated gradient at time t, γ is a hyper parameter of how much momentum should be kept, λ is the learning rate and g_t is the gradient for weight w_t. RMSprob uses the same concept but looks at the derivative of the change. E.g. if the gradient keeps jumping up and down the vertical direction but is stable in the horizontal direction, we would assume that the vertical jumps is most likely noise. RMSprob therefor squares the gradient element wise and adds it on the old "momentum"

$$v_t = \gamma v_{t-1} + (1-\gamma) g_t^2, \tag{3.35}$$

$$w_t = w_{t-1} - \lambda \frac{g_t}{\sqrt{v_t + \eta}}, \tag{3.36}$$

where γ is the proportion parameter. As seen the learning rate λ is not replaced but rather corrected in each dimension. Both approaches has its gains and losses which is why the ADAM optimizer was suggested [44]. This optimizer combines the two

approaches into one:

$$v_t = \gamma_1 v_{t-1} + (1 - \gamma_1) g_t, \tag{3.37}$$

$$s_t = \gamma_2 s_{t-1} + (1 - \gamma_2) g_t^2, \tag{3.38}$$

$$w_t = w_{t-1} - \lambda \frac{v_t}{\sqrt{s_t + \eta}} \cdot g_t, \tag{3.39}$$

which is shown to work better than SGD in most cases. Another benefit is that it scales the learning rate according to the data which makes the initialized learning rate less of a factor.

3.2.4.2 Embeddings

It is not possible for networks to work directly on non-numerical data. A computer only understands numbers. It is therefor necessary to convert data which contains words or classes into a numerical domain and this is where embeddings comes into play. An embedding of a variable is a representation in the numerical domain. E.g. a network does not understand the words "yes", "no" and "maybe", but if projected onto a line with "yes" being 1, "maybe" being 0 and "no" being -1 it could be processed. Often a one dimensional space is not enough and the network needs to find its own meaning. Therefor, when using data like words, an embedding space is defined with some fixed dimension, n, in which the available words is distributed. The embedding of the data is then trained just like the rest of the network.

3.2.4.3 Weight initialization

The initialization of the network weights can have a huge impact on how well the network trains. One will typically initialize weights using some random distribution. Initializing with very large values can be problematic since it can result in slow learning. This is due to most activation functions, $a(z)$, being asymptotic with the line $y = 1$ for $z >> 0$. This means that when computing the derivatives of $a(z) = a(xw+b)$ for $w >> 0$, then $\frac{\partial a(z)}{\partial z} \approx 0$.

Initializing the weights with very small values can also result in slow learning, since all inputs will result in small outputs that are very similar.

A particular problem arises if one initializes the weights to be all zero. The symmetry results in the weights all being updated the same way. This effectively means that no matter how big the layers are, they are equivalent to just one node in each layer.

In short, thinking about the initialization is important. Optimally, one initializes the weights such that the signal propagated forward and backward by the network keeps a steady size that neither vanishes nor explodes. Different initialization methods have been researched, but random initialization is the common theme. Since the weights are put into a sum, $z = \sum_i w_i a_i + b$, we want the magnitude of w to be relative to the size of the sum, i.e. the amount of weights in a layer. Imagine having

millions of weights in the same layer, then z will get really big if we are not careful with the sizes of the weights. When using $tanh$ as the activation function Glorot et al, found that initializing the weights from a normal distribution yielded the best results [45]. This with mean zero and variance $Var[w^l] = \sqrt{\frac{1}{n^{l-1}}}$, where l is the layer, w^l the weights of that layer and n^{l-1} the size of the previous layer, i.e. the amount of weights going into a node in layer l. This initialization method is also called Xavier initialization, and is the one used in this thesis.

3.2.4.4 Gradient clipping

To avoid exploding gradients, a method called gradient clipping can be used. It essentially forces all gradient values individually to be in a certain range, by clipping values higher than the maximum value, or lower than the minimum value. Normally this is used such that the gradient can not be larger than 1, which nullifies the exploding gradient problem [46].

CHAPTER 4
Entropy Estimation

Before looking into the actual predictions we wanted to get an idea of what we should expect from the data. As part of normal data procedure we want to know if the data has the quality to be trusted when predicting. When talking about quality it is not the data collection, but if the sequences are long enough to contain enough data for the entropy to converge. After figuring out which sequences that had converged the upper bound of predictability will be touched. It should be noted that we will use the full data set rather than just the training data. We are not training models here, so there should be no risk of overfitting.

The code for entropy estimation is written by Bjørn Sand Jensen [47] in MATLAB, and rewritten to python by a former DTU master student Mikkel Hansen.

4.1 Entropy convergence

The entropy convergence of the Sensible DTU data has formerly been explored in [48] but on a different pre-processed location data set. The sequences was constructed by looking at the connected cell tower ID's and comparing continuous data streams with data only captured when calling or sending a text message. In our case the locations are constructed directly from the GPS latitude and longitude data, as explained in chapter 2, which might affect the convergence and upper bound of predictability compared to the former.

To get an idea of how the entropy converges the estimated entropy for every individual in our training set is plotted for different lengths of the sequences with a step size of 10, see Figure 4.1.

In Figure 4.1 we notice how the estimator seems to converge for most sequences, but this is quite hard to fully see. Therefore we plot the difference in each step instead, which can be seen in Figure 4.2. We have chosen to plot the slope between every 50th data point, $slope = |f(x) - f(x+50)|$. This is to see that the estimator is stable over a longer period, while not capturing every jump in entropy when a new location pop up.

Figure 4.2 looks quite promising but to express explicitly that the data is converged we need more evidence. We choose the same approach as in [48] and say that the estimator has converged if the slope is less than 0.05 at least 8 out of 10 times in a row. This can be used due to the fact that new explorations in a sequence will

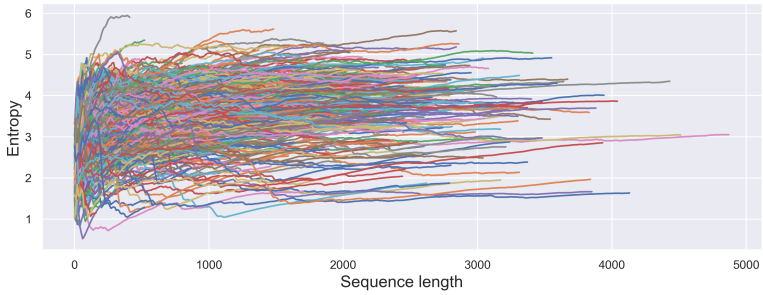


Figure 4.1: Convergence of the entropy estimator for each sequence length with jumps of ten. It is seen how the estimator seems to converge for most sequences.

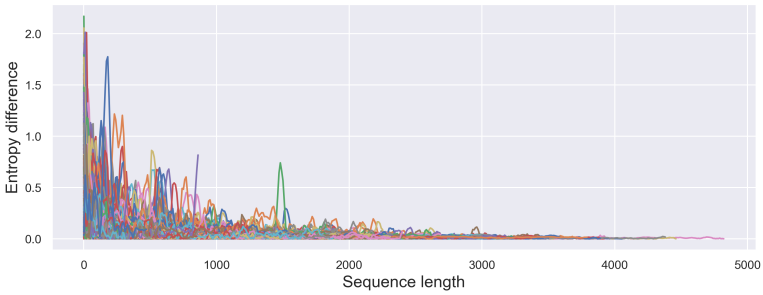


Figure 4.2: Slope of entropy estimator for each sequence length. The plot shows how the slope decreased over time which suggests the entropy estimator converges for most sequences.

introduce small jumps in the estimated entropy which is unavoidable. Out of the 851 sequences in our data set, this methods sorts out 196 of them as not converged and leaves us with 655 sequences where we believe the entropy has converged. Thus we only move forward with these 655 sequences.

4.2 Upper bound

Now looking at the theoretical upper bound, we know that the entropy can be transformed into an upper bound of predictability from [8]. So by calculating the estimation of the entropy, of all the 655 users found above, and the respective predictability we get the distributions shown in Figure 4.3, with a mean entropy of 3.74. This is equivalent with the entropy of a uniform distribution with 13.4 states, $2^{3.74} = 13.4$, which shows how much information is stored in the data. The figure shows that the entropy estimator for the users is in the interval $ent_{est} \in [1.6, 5.6]$ and the predictability

in $pred_{est} \in [0.5, 0.86]$ with a mean predictability of 65.9%.

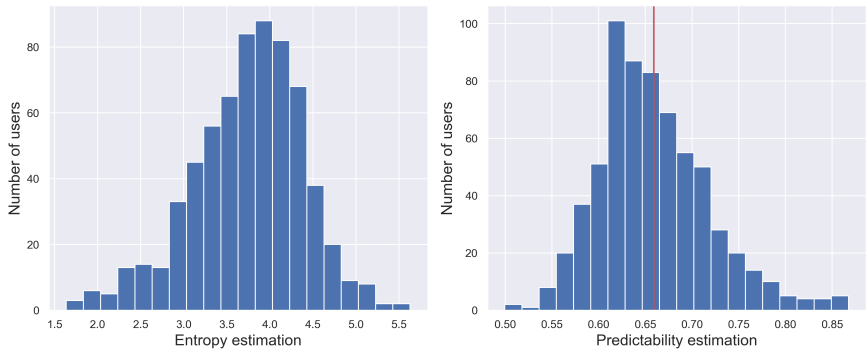


Figure 4.3: Distribution of user entropy and predictability for users **after preprocessing**. It it shown that the upper bound of predictability averages at 65.9%, but it goes between 50% and 86%.

Doing the same for all 851 users we get the distributions shown in Figure 4.4. Here the entropy estimator lies in the interval $ent_{est} \in [1, 6]$ with predictability $pred_{est} \in [0.3, 0.88]$ and predictability mean 64.3%. When comparing the two data sets we see that the preprocessed set is more compact and has less "extreme" values just as desired.

In total we end up with an upper bound of predictability on 64.3% for the Sensible DTU data. This means that no matter what we cannot get a higher prediction rate than 64.3% on this data set.

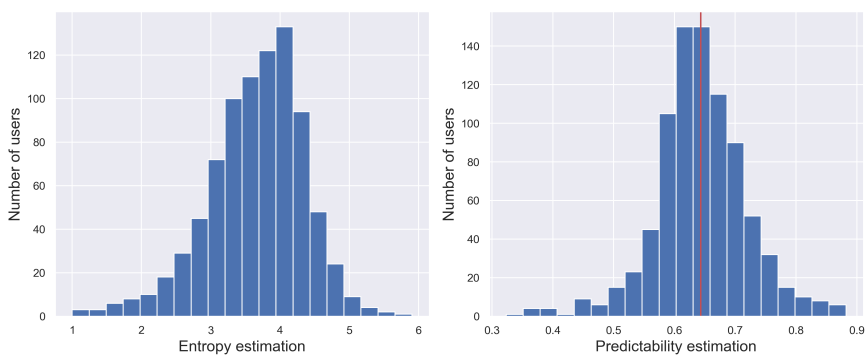


Figure 4.4: Distribution of user entropy and predictability for **all users**. Looking at the upper bound of predictability it is seen that there are more extreme values in both the lower and the upper end. This is likely outliers so the removal worked as desired.

CHAPTER 5

Data Exploration

The total data set consists of 851 mobility sequences of individuals starting in 2013 where the longevity of the data collections varies due to errors, change of phone etc. Since this thesis attempts to fit a machine learning model, specifically a LSTM, we split the data into a training, validation and test set. Since we are dealing with temporal data it is not possible to split the data into random sets. Instead we use the first 90% of the sequence lengths as training data, and then use the last 10% as validation data and test data. The last 10% of the sequences of users with id less than 600 will be used for validation and the rest as test data. This split is illustrated in Figure 5.1. Note that this is a simplified illustration since the sequences do not have same lengths. In this section of data exploration we will use all the data due to the low amount of data available.

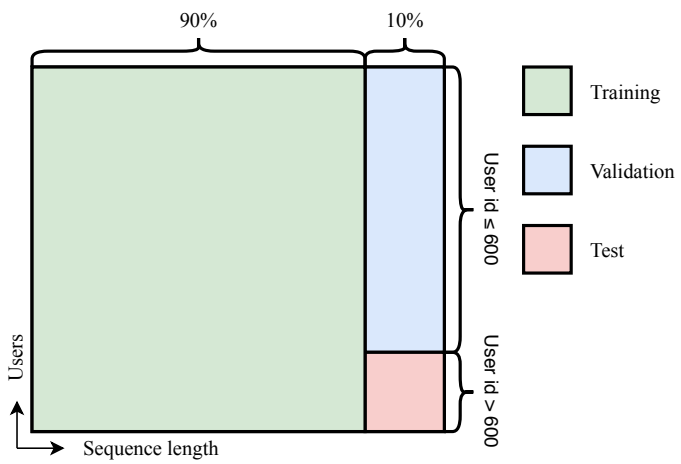
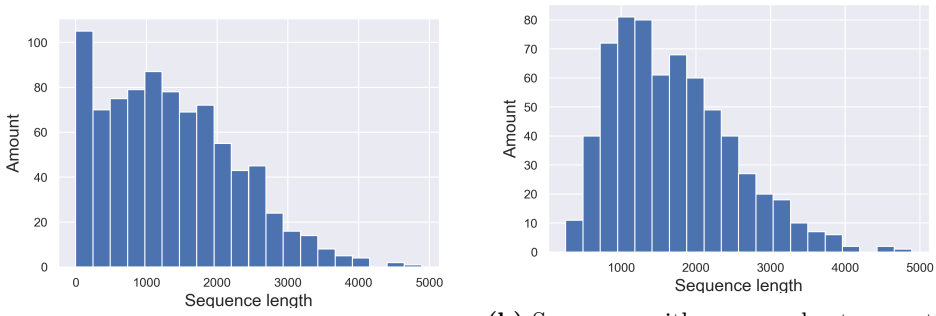


Figure 5.1: Simple illustration of how the data is split up. Note that the sequence lengths are not the same so this is just an illustration. It it seen how the data in the temporal direction is split up into 90% and 10%. The 90% (training set) is used for training using all the users. The last 10% is again split into a validation set (blue) and test set (red) depending on the user ID.

Looking at the size of the Sensible DTU data we can see that the sequence lengths

varies quite a bit, see Figure 5.2. The shorter sequences can prove a problem when trying to fit a model since there simply is not enough data for the models to learn. However comparing the distributions of the full data set and the one where the entropy estimator had converged, Figure 5.2, we see that most of the shorter sequences are removed. Even if most of the short sequences are removed we wanted to make sure there are enough data for our models to make proper predictions, we therefor applied another criteria saying we only use sequences longer than 1000. This left us with 507 users; 427 for validation and 80 for testing.



(a) All data

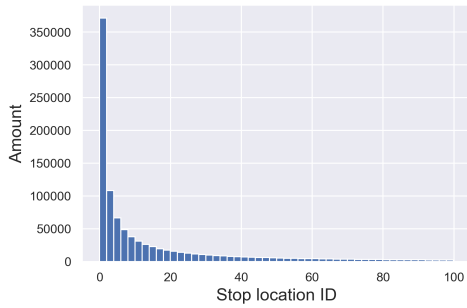
(b) Sequences with converged entropy estimator

Figure 5.2: Distributions of sequence lengths for the users before (a) and after preprocessing (b). We see how most of the short sequences are removed and the longer ones tend to persist.

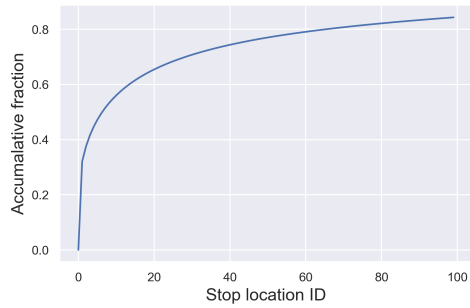
To help fit the model to the data, the labels of the locations were sorted according to frequency of the training set. This is to help models generalize across users. E.g. the likelihood of the home location being in the top three most visited locations are quite large, and information about that location is quite valuable (such as the home location is usually the place to be during the night). Instead of learning individually the model can then use this information across many users. We sort the locations based on the training set to avoid giving the training data extra information about the validation and test set. We kept track on how the labels were swapped and used the same swaps on the test and validation set. The swap was due to the fact that we wanted to retain information across users, and even if the users did not have exactly the same distributions it would still hold information. This decision is discussed further later on. Note that some locations did not occur before the test set. These locations were given an ID higher than any other which results in the network not being able to guess it.

Plotting the location ID distribution, see Figure 5.3a, we see that a large amount of locations are put into the first few locations, with the distribution looking like exponential decay. More specifically in Figure 5.3b we see that the first 20 locations

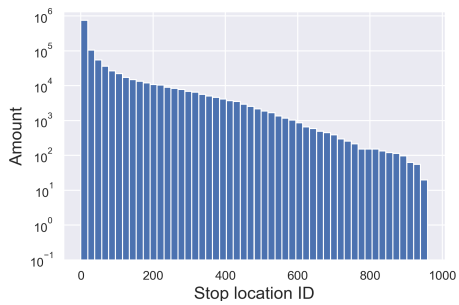
accounts for more than 65% of all the data points. Looking at Figure 5.3c, where the y-axis is logarithmic scaled, we see that the distribution is not totally linear, thus similar to exponential decay. By the long tail we see that some stop locations are rare due to peoples exploration. This can be a problem when predicting which is discussed later on.



(a) Distribution of location IDs.



(b) Accumulative distribution of location IDs.

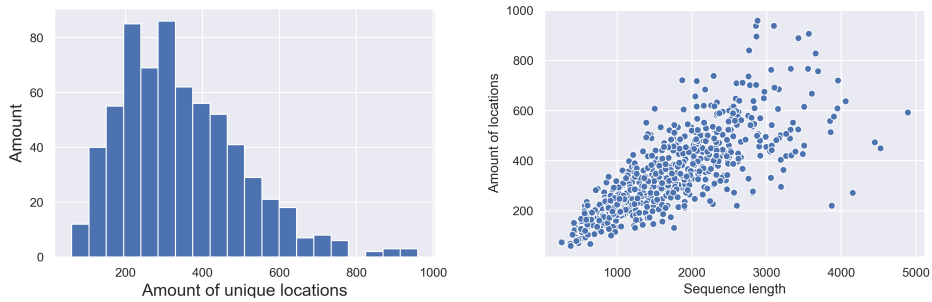


(c) Distribution of location IDs with a logarithmic scale on the y-axis.

Figure 5.3: Distribution, accumulative distribution and logarithmic scaled distribution of the amount of each location for all of the users. We see how the first few locations accounts for most of the data. Specifically the first 20 locations accounts for more than 65% of the data.

Looking at the distribution of the amount of locations the users have visited, see Figure 5.4a, we see that it is very similar to the distribution of sequence lengths, see Figure 5.2b. This is not very surprising since a larger amount of locations visited usually means more unique locations visited. Plotting them against each other we clearly see the correlation, see Figure 5.4b.

However, what really is going to hurt the predictions is exploration. We cannot predict locations that we have never seen. So of all the locations that has been visited how many of them are only visited once? In Figure 5.5 we see that the fraction of locations only visited once per user averages at 68%. But this does not mean that 68%



(a) Distribution of amount of uniquely locations.

(b) Correlation between sequence length and maximum location ID.

Figure 5.4: It it unsurprisingly seen that the distribution of the amount of locations visited for each user is correlated with the sequence length of the same user.

of the visits are explorations. Therefore we look at the fraction of explorations, see Figure 5.6. Here it is seen that on average 21.6% of locations switches are explorations, and it should be noted that every location is an exploration at some point. All of this is really emphasizing that, for the sake of correct predictions, it is important to predict the lower locations id right, since the higher once are most likely impossible to predict.

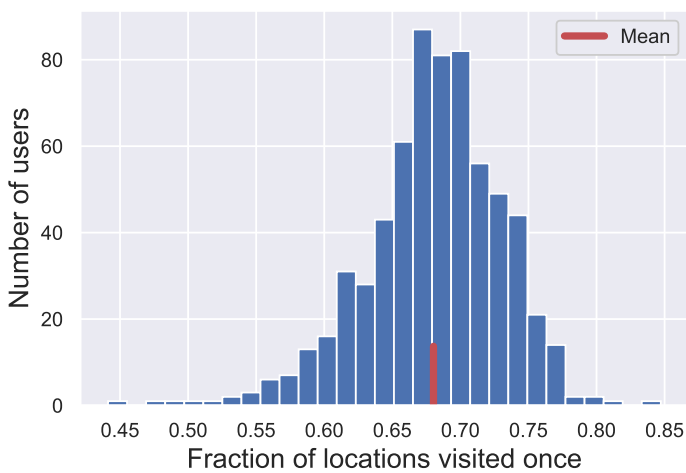


Figure 5.5: Fraction of locations that are only visited once. We see that on average 68% of unique locations can never be guessed.

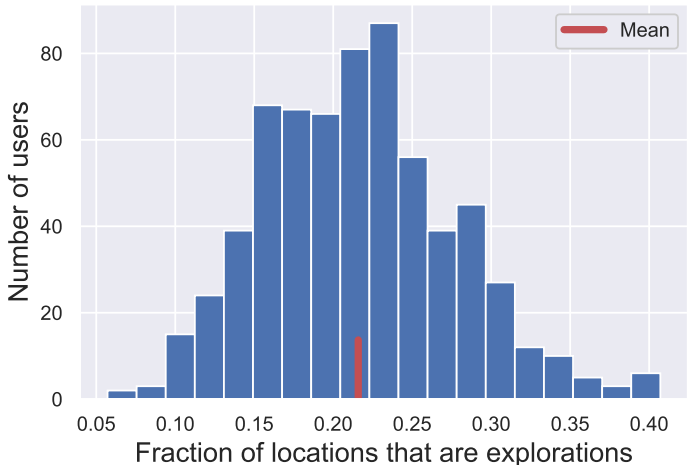
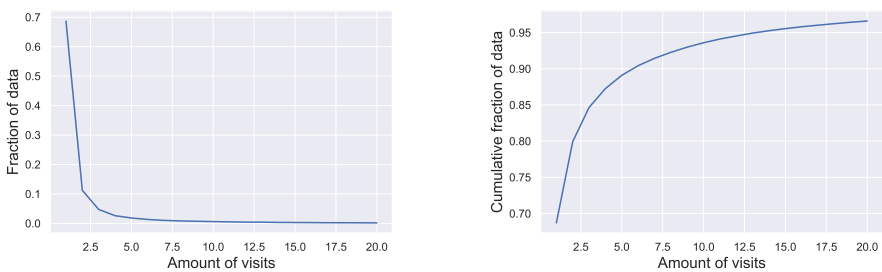


Figure 5.6: Fraction of data that are explorations. We see that on average it is impossible to predict 21.6% of all next locations since they are never seen before.

In Figure 5.7 we see that locations visited more than 5 times only accounts for 11% of the locations. With this the take away from the data exploration is the emphasize on the importance of the few locations which are possible to predict.



(a) Fraction of data which has x amount of visits.

(b) Cumulative raction of data which has x amount of visits.

Figure 5.7: It is shown that most locations are visited very rarely. E.g. locations visited at a maximum of 3 times accounts for more than 85% of the total locations. This means that most locations are very hard if not impossible to predict since they are rarely seen in the trajectory of the users.

5.1 Baselines

To get an idea of a lower bound of predictability and see how other models perform we fitted some simple models to the data. This was done after the data was preprocessed by the removal of potential unconverged or short sequences. We selected six prediction baseline models and evaluated how they performed:

1. **Most common.** The simplest model which always guesses on the most occurring location.
2. **Most common but not self.** A model which guesses on the most occurring location. If current location is the most occurring location, it guesses on the second most occurring.
3. **Markov order 1.** A Markov model of order 1 guesses on the most occurring transition away from the current location.
4. **Markov order 1 with twist.** A Markov model of order 1 with a twist, which given a location guesses on the location from which the user switches to the most but only if this location is visited more than n times. If it is not then guess on the in general most common. This is due to the high amount of explorations in which jumps could be random.
5. **Most common given time slot.** A model which given the hourly time slot guesses on the most common location.
6. **Markov order 1 given time slot.** A model which combines the Markov order 1 and hourly time slot.

We evaluated baselines on the validtion data i.e. the users up to id 600. Model 4's parameter n was fitted by simply testing $n = [1, 2, ..., 10]$ where $n = 2$ gave the best result. Jumping straight into the results we can see the prediction distributions for each model in Figure 5.8 with corresponding mean prediction rate in Table 5.1.

Baseline models	**Mean prediction rate**
1. Most common	20.4%
2. Most common but not self	23.0%
3. Markov order 1	26.6%
4. Markov order 1 with twist	26.7%
5. Most common given time slot	20.2%
6. Markov order 1 given time slot	24.4%

Table 5.1: Averaged prediction rates for baseline models. Full distributions is shown in Figure 5.8.

Looking at the prediction scores it was not surprising that just guessing on the first model, *most common*, performed the worst due to the simplicity and lack of

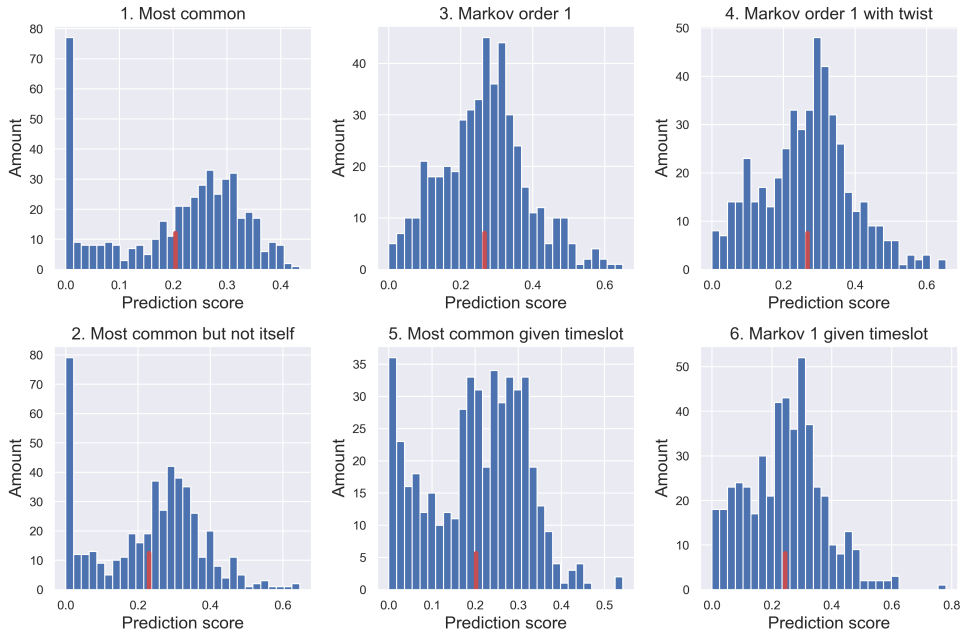


Figure 5.8: Distributions of predictions for each baseline with the mean prediction marked as the red line, see Table 5.1 for average mean predictions

temporal information. Same goes for model two, *most common but not self*, which is basically the same but with information of how the data is formed. A bit surprising was model 5, *most common given time slot*, which performed even worse than model 1. We had thought that the time slot mattered more in the decisions of location switches. Unsurprisingly all the Markov models performed the best, as they are more complex models. Also when looking at the prediction distributions we see that the "most common" models, 1,2 and 5, have a lot of prediction scores close to 0. Since we are testing on the last 10% of the data, which as a minimum is 100 locations, it is not unlikely that the user is on a vacation, sleeping at a partners place or something similar, which would result in a zero prediction rate. When the models are too simple it seems as they do not have a chance to correct themselves to these changes. If the user is on a vacation it will be hard for any model due to a lot of new locations, but if it is a less common situation, like visiting parents that had happened before, the more complex models shines. This is clear by the prediction distributions. The Markov models has less variance in their predictions, even though it looks like model 6, *Markov order 1 given time slot*, does still have problems. This is likely due to the shortage of data, i.e. it is too complex for its shallow vision. When standing in an uncommon location, the data has very little information, especially when looking at

hourly time slots. On the other hand model 3 and 4 deals with all of the data at all times and therefor does not end up in as much of a shortage of data.

Overall we end up with an average prediction lower bound of 26.7% found by model 4, *Markov order 1 with twist*. This means that we we will not be satisfied by a result lower than this prediction rate when testing our network.

CHAPTER 6

The Network

The initial goal was to make a neural network which sole purpose was to make the best predictions on next locations of human mobility. Here we had the choices of going with the Long Short Term Memory (LSTM) network or a Gated Recurrent Unit network (GRU), which are both extensions of the vanilla recurrent neural networks (RNN). Both networks are quite similar with the GRU network being a bit more simple in construction, which makes it more computationally efficient [42]. This makes it great for processing huge loads of data with nearly the same performance as a LSTM but since we did not have an overload of data we went with the more complex LSTM.

The implementation was done in python version 3.7.3 using PyTorch version 1.2.0 as the main tool.

6.1 Initial network

The overall architecture of the initial network is illustrated in Figure 6.1 and will be discussed throughout this section. A crucial feature of the network is the embedding of the data. Neural networks cannot directly deal with data consisting of classes, and even though our location data consists of integers they do not have any direct relation to each other. To deal with this problem we embedded the data and trained the embedding. Not all of the inputs needed to be embedded since they did have relative meaning, like time of the day. As introduced in chapter 2 the data consists of the location sequences, and time of arrival and departure. We suspected there were information in both the sequences and the departure times, so we decided to split the departure time into several inputs:

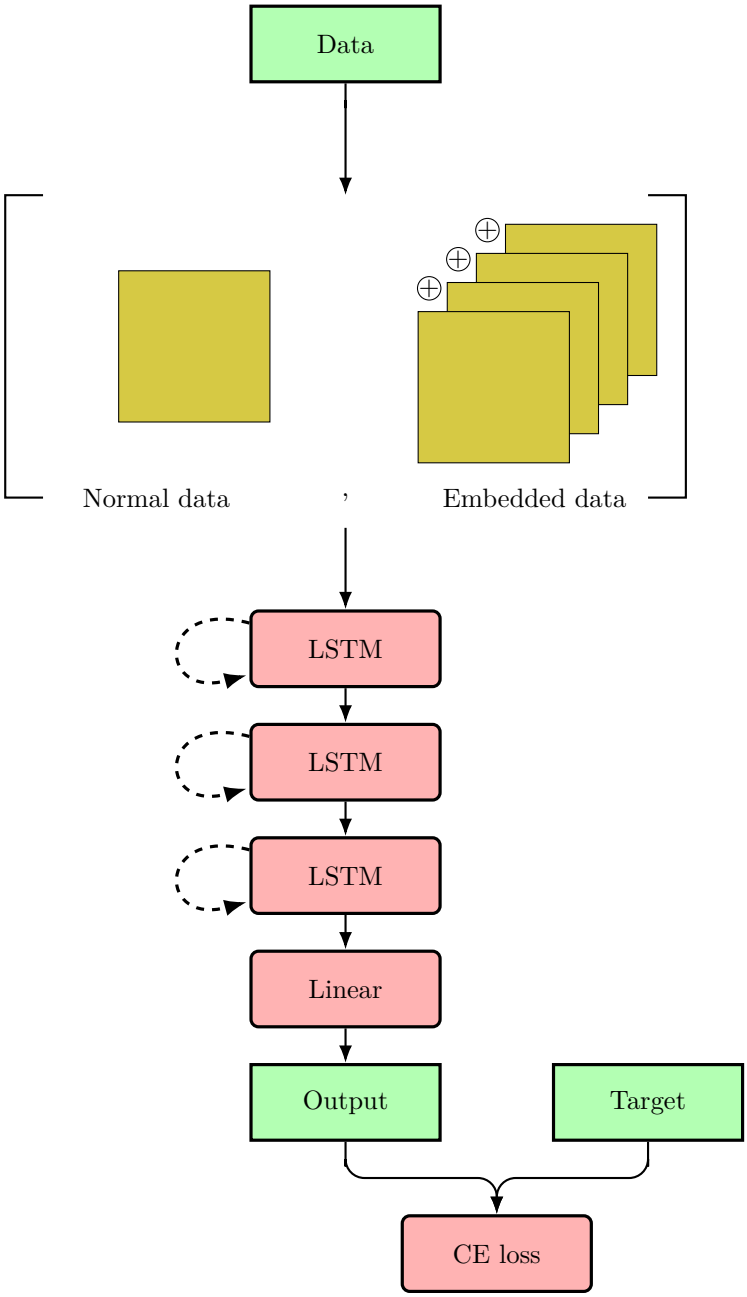


Figure 6.1: Graphical illustration of the initial network. The red boxes are different data transformations, and the green boxes are a clarification of what the data is at that point in the network.

- *Time of departure* is relevant but has the problem that 23:59 and 00:00 is technically really close to each other but in values this is not directly clear. We therefor parameterize the data down on a circle just like a normal watch, but with 24 hours on it instead of 12 hours, by the use of cosine and sine.

$$\text{time} = \left[\cos\left(2\pi \cdot \frac{\text{time in second}}{\text{second per day}}\right), \sin\left(2\pi \cdot \frac{\text{time in seconds}}{\text{second per day}}\right)\right] \tag{6.1}$$

  Here *time in seconds* divided by *seconds per day* acts as the fraction of how far around the circle the time is. And with this parameterization 23:59 and 00:00 is really close to each other just as desired.

- *Day of the week* i.e. Monday, Tuesday, etc. We use the integers from 0 to 6 to represent each week day. This is however another class and we will therefor also train an embedding for the day of the week.

With this we ended up with the inputs for the initial network as shown in Table 6.1. It is also marked which of the inputs are embedded and which are not.

	Network 1	Embedded
Location id	×	×
Cosine time	×	
Sine time	×	
Week day	×	×
Person id	×	×

Table 6.1: Data inputs for the initial network, and what attributes that got embedded.

Ignoring batches for a second, we needed to define how to pass the data into the network. Here we drew inspiration from natural language processing models with text, and took all the sequences and extended them with a "stop token" in between them. This stop token is a token that is only shown whenever there is a switch in sequence. The purpose for this token is to tell the network that a new sequence is coming, and the hidden state of the previous sequence should "reset". We selected different stop tokens depending on the sequence attribute. I.e. for locations we choose the number 925 since it was higher than any other location id, 859 for person id, etc. Basically it is a token that is only shown between sequences, e.g. $[x_{(1,1)}..., x_{(n,1)}, \langle token \rangle, x_{(1,2)}, ...]$, where $x_{(n,k)}$ is the nth value in sequence k. When using batches we took this long sequence and just reshaped it into the desired size and cut off the remaining data that did not fit the reshaping.

Then the next question is how to deal with the embeddings. First we needed to figure out the dimensionality of the space we wanted the embeddings to be in, and secondly how we feed the embeddings to the network. The dimensionality is a hyper-parameter that we needed to fit after the network was built. Therefore we

started with focusing on the embeddings. We thought of two different approaches of dealing with the embeddings. First solution was to concatenate each embedding on the end of the other, and the other solution was to sum them up given they had the same dimensionality. Here we selected the second approach to keep the network small, thus defining the embeddings with the same dimensionality. Now we were left with the summation of the embedded data, see Equation 6.2, and the normal data which we concatenated to have the full input to the network. We did not need to weight the sum since the network should learn the optimal weighting internally in each embedding when running back propagation.

$$Embedding\ matrix = \sum_i embedding(data\ type) \tag{6.2}$$

The written structure of the network ended up being; $input \rightarrow embedding \rightarrow LSTM \times 3 \rightarrow linear/\ decoder\ layer$, see Figure 6.8 for a visualization. Now that we had the output we needed to define a loss function, and as mentioned earlier we selected cross entropy since we are doing classification. The final piece to get our network running was the optimization algorithm which does the stochastic back propagation. We selected to go with the ADAM optimizer since it finds the learning rate by itself and uses a combination of several methods which has been seen to generally work better. Since it is not possible to shuffle our data set, due to temporal information, we instead generated a random number, i, between zero and the sequence length, $i \in [0, seq_length]$, which decides the initial index of the batches. E.g. lets say we have 100 data points and a sequence length of 20. Instead of always splitting the data up in the batches, b_n, the same way ($b_1 = [x_1, ..x_{20}]$, $b_2 = [x_{21}, ..., x_{40}]$ etc.) we would get a random index, i between 0 and 19 and start from there. Thus we get $b_1 = [x_i, ..x_{i+20}]$, $b_2 = [x_{i+21}, ..., x_{i+40}]$. This way we ensured some randomness in the training. However when we wanted to predict on a test set we needed to ensure that the hidden state was warmed up. So there we needed to fix the index i such that the last batch directly matched the following test data. E.g. lets say our data set is 100 points, and the test set is 10% of the data. This leaves us with a training set of 90 points and 10 test points. If we have a sequence length of 7, and we want the batches from the training set to align with the test set, we need the last point of the last batch to be point 90. By selecting $i = 90 \mod 7 = 6$ we achieve exactly this.

Lastly we needed to optimize the hyper parameters for the network, specifically:

1. Amount of LSTM layers.
2. Sequence length.
3. Hidden dimension.
4. Embedding dimension.

As with all types of machine learning models, we wanted to keep the models as simple as possible to avoid overfitting, and minimize the amount of computations. Therefore

when figuring out the hidden and embedding dimensions the approach is to start low and then increase the values until the network begins to overfit. When the model begins to overfit one can either introduce some regularization or keep track of the validation scores and stop the training early. We choose the later and found the values for the dimensions to be $hidden\ dim = 13$ and $embedding\ dim = 15$.

To check whether the network was implemented properly we attempted to overfit on less complicated data. We did this by feeding the network the same data from user 1 ten times in a row and tested if it was able to overfit. Here the network was able to get a prediction score of $99.8\% \approx 100\%$ which made us trust the implementation was correct.

When trained the network gave the results seen in Table 6.2

	Prediction rate
Network 1	28.0%

Table 6.2: Next location prediction rate for network 1.

This result was slightly better than the previously tested baselines, 26.7%, see Table 5.1, but we wanted to do better. The baselines were all based upon the individual location history, whereas our model was a generalized method for all users. One problem we had with the current approach was that we assumed that the location distributions for each person was the same. Plotting the location distributions of four random users we see that it is not entirely true, see Figure 6.2. The distribution were similar but not equivalent.

6.2 Final network

During our discussions of what we wanted to aim for, two new ideas came up. We wanted to capture more individual information and also explore if we could let the network find similarities between the different user locations. Location 0 is likely home for most users since it is the place they visit the most, 1 is school/job, etc., but they could also be switched if you always go to the gym during lunch and come back.

We tried to tackle the first problem of individual information by introducing a new kind of input data. Instead of only relying on the fact that users have somewhat similar location distributions and patterns we also let each user have 20 unique location id's locked just to them. We did this by giving location 0-19 their own label and let 20 contain the rest of the locations for each person. After that we let the user id decide the twentieth interval to run in, i.e.

$$location\ type = location\ rank \mod 20 + user\ id \cdot 20 \tag{6.3}$$

This means that person 4 will have the unique locations of $80-99$ assigned to him/her. By giving that as an extra input to the network we hoped to give a more personalized prediction. We selected the number 20 from the fact that it captures around 65% of

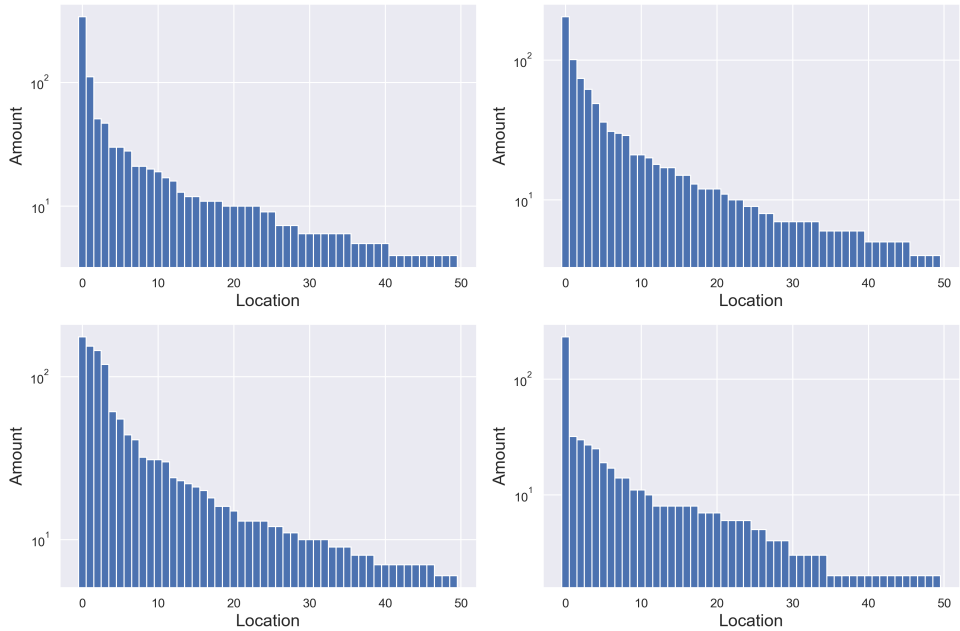


Figure 6.2: Location distributions of the first 50 locations for four random users. It is seen that the location distribution for each user is similar but far from equivalent, so assuming they are equal is not feasible. Note that the y-axis is logarithmic scaled.

the data, see Figure 5.3, and since 68.6% of all locations are only visited once, see Figure 5.5. The first 20 locations covers most of the information that is possible for the network to capture. Just like the location ranks, we also embedded the location and added it to the summation of embeddings.

The second type of information we thought would bring some extra performance was if the current location was an exploration or not. If someone was already out exploring they might continue to explore or go home. Adding these two new inputs we ended up with the new input table, see Table 6.3.

Despite of the extra information we still needed to figure out how to let the network capture the similarities between user locations of different users. Here we drew on inspiration from research of natural language processing, specifically word2vec [49], which tries to train embeddings of words. By training a network on text it was possible to learn the embedding information about which words was connected and had similar meaning. E.g. the famous example:

$$k\bar{i}ng - m\bar{a}n + wo\bar{m}an \approx qu\bar{e}en \tag{6.4}$$

Here $te\bar{x}t$ means the embedding of the word "text". One advantage of words and sentences is the interconnecting rules binding associated words together. Previously

	Old Network	Final Network	Embedded
Location (rank location)	×	×	×
Cosinus time	×	×	
Sinus time	×	×	
Week day	×	×	×
Exploration		×	
Person	×	×	×
Personal location (type location)		×	×

Table 6.3: Data inputs for the network, and what attributes that got embedded.

we did not have this for our locations and we hoped to find some kind of meaningful representation of the locations. Optimally we hoped to have home being in one cluster, work/school being in another cluster, etc., even if they didn't share the same location id. In order to force the training of the embeddings we wanted to include it more in the process of predicting, especially since the training of the embedding happens in the start of the network, which due to the vanishing gradient problem and the way gradient descent is being calculated, updates very little. We therefore introduced a new way of estimating and evaluating our predictions.

Instead of letting the final linear layer, of the network, estimate a probability for each class, we instead force it to estimate the embedding of the target location. We do this by letting the output dimension of the layer fit the embedding dimension and then evaluate the target scores by taking the dot product between the estimated embedding and the embedding of the target. We took the decision to restrict the network to only train and predict the first 19 locations. This mean that location 20 that represents all other locations would never be predicted, which effectivly means locations higher than 19 cannot be predicted. By using this restriction we took the embedding of all 19 locations of the given user and calculated the dot product with the output from the network. This vector was now our definition of the logits vector which we plugged into the cross entropy as the loss function. In math terms:

$$logit_n = out \cdot \bar{loc}_n \qquad \text{for} \quad n \in [u_id \cdot 20, u_id \cdot 20 + 19], \tag{6.5}$$

where out is the output of the network, u_id is the user id and loc_n is the n'th location. The idea behind was that in a point cloud centered around the origin, the dot product of two vectors created from the two points should be high if the points are close to each other and low if they are not. So when computing the score for every possible point and run it through our loss function the network should push the points away from each other. Another very important effect of this change is that the network can learn to give locations types (home, work, etc) the same embedding across users even if they do not have the same location id. E.g. if fitness was location 4 for one user, and 7 for another user, then the network has the possibility to learn that the embeddings of each location are the same. Afterwards the network can use the generalized information about what you do after fitness. To achieve this we defined

the target as the summation of the embeddings of location type, location rank and person id. I.e.

$$\bar{loc}_n = \bar{location\ rank}_n + \bar{location\ type}_n + \bar{user\ id}_n. \tag{6.6}$$

This way the target is a combination of the generalized information, *location rank*, and individual information, *location type* and *user id*. The updated network architecture is shown in Figure 6.8.

Now that we had the network setup we trained the network 10 times to see if the initialization had a big influence on the networks performance. In every training loop we did, the network tested its performance every 5th epoch on the validation data, and if the prediction rate got lower twice in a row (over 10 epochs), we stopped the training and assumed it had begun overfitting. We saved the best iteration of the network and the performance for each of the 10 training loops. It took around 65 epochs on average to converge, with a running time of 1.7 hour per loop/ trained model. The results is shown in Figure 6.3, showing box plots of the top@k prediction for $k = 1, 3, 5$, where $k = 1$ is the normal prediction rate. We see that the network was quite consistent in predicting and that the initialization did not matter much. The data used for training and prediction did not include our testing data, only the training and validation data, see Figure 5.1.

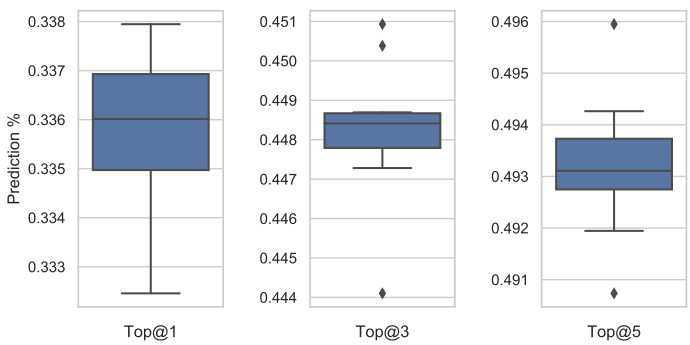


Figure 6.3: Boxplots showing the prediction performance of the network with 10 different start seeds to see the influence of the initialization.

Figure 6.3 shows that the initialization only had a small influence on the predictions. Compared to our best baseline model, which had an average prediction rate of 26.7% (see Table 5.1), we now got an average of 33.6%, for Top@1, which was a great improvement. Apart from top@1 we also got an average prediction rate of 44.8% for top@3 and 49.3% for top@5. This means that if we take the top three guessed locations of the network we can predict the next location with a probability of 44.8% on average compared to 37.4% had we guessed on three most common, see

Figure 5.3b. Nearly half of the locations are included in the top three. Looking at the top five locations we get a prediction rate of 49.3% compared to 44.5% on just guessing on the five most common. Since we have restricted the model on the first 20 locations we know that a hard upper bound of predictability, besides the theoretical, from the data is 65%, see Figure 5.3b.

Besides predicting the other goal of the network was to see if we could train the embedding to find some underlying structure, specifically for the three most common locations, since they most likely represent something similar across users, like home, work, and other important places. Using the network with the highest average prediction rate, 33.6%, we explored how the embeddings had been altered to see if anything stood out. However to find structure in a 13 dimensional space, we needed to do some dimensionality reduction. Several methods is available and already implemented in python like PCA, TSNE and UMAP. PCA however is a simple transformation of the basis, where TSNE and UMAP are nonlinear dimensionality reduction techniques. The purpose of those algorithms is to find clusters in high dimensions and then do a nonlinear projection down into two or three dimensions. UMAP is the newer algorithm which tried to solve some problems with TSNE, and we choose to go with that.

Since we wanted to compare the locations across users, we took the embeddings of the type data and ran it through the UMAP transformation. We then colored each location according to modulo 20, i.e. location $46 \mod 20 = 6$. We ended up with the result shown in Figure 6.4. Unfortunately we did not see any useful patterns, and neither when looking at the individual locations, see Figure 6.5.

We had especially hoped to see some structure around the first three locations, but it seems like that the network have learned to predict through the LSTM layers, and not the actual embedding. We tried to combat this problem by removing the ranking data from the target embedding, however this proved a greater challenge for the network since it had to learn which twenty range to guess in. The hope was that this would force the type embedding to be trained and hopefully capture more of a structure. After training, the network got a prediction rate of 31.7%, which was lower than before (33.6%), but still a plausible attempt. Looking at the embedding however we saw it had captured a bit more structure, compare Figure 6.6 with Figure 6.4. Again we plotted each type location separately shown in Figure 6.7. Clearly there are some kind of structure, but specifically what the structure is, is unclear. Only location 1 seemed to be more determined, where the rest of the locations switched from the left to the right cluster. A guess for this behavior could be that the network learns not to guess on itself by splitting even and odd numbers up. However just like the previous network no geographical information seem to be present in the embedding. Most of the actual learning seems to be stored in the LSTM layers.

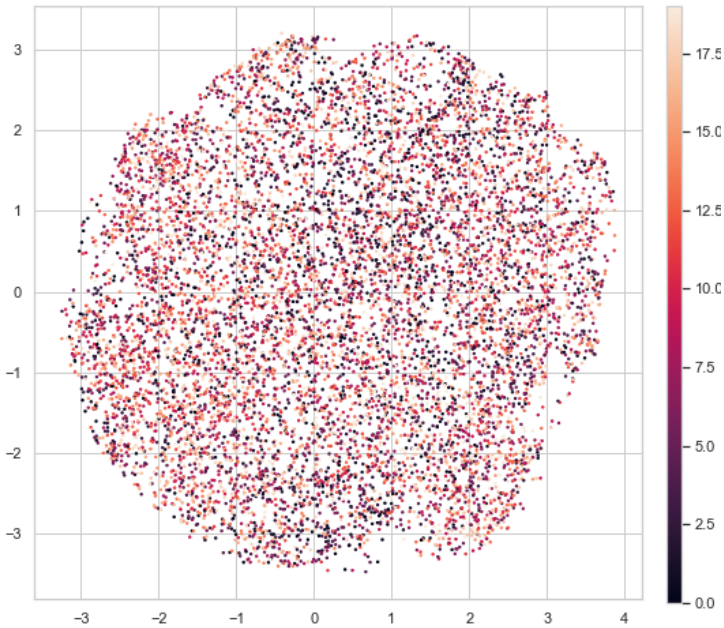


Figure 6.4: Embedding of the type locations transformed down into 2 dimensions with UMAP. There is no clear structure in the embedding which means our network does not learn a location specific embedding. Since it still looks quite uniformly distributed the LSTM layers seems to learn most of the connections between locations.

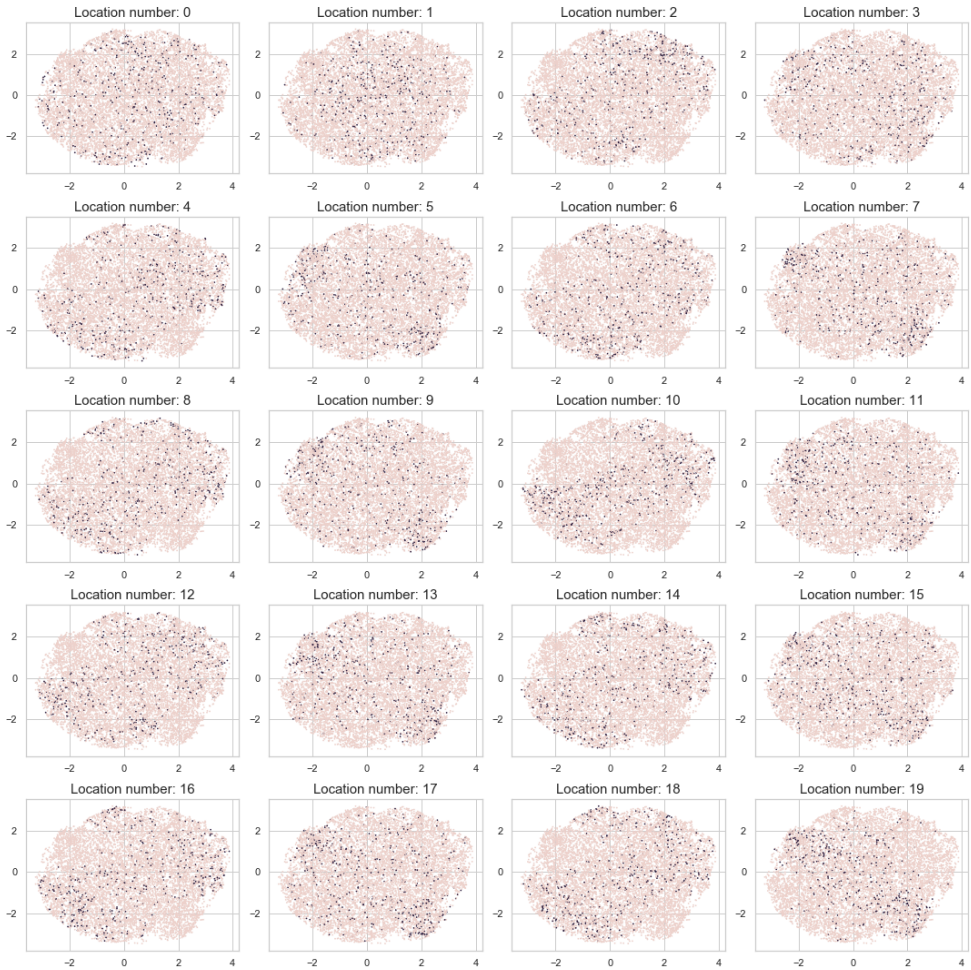


Figure 6.5: Embedding of the type locations transformed down into 2 dimension with UMAP colored individually according to location. The black dots represents the explored locations and the pink the rest. Again no clear structure is seen, which probobly means it is the LSTM layers that learns the temporal connections between locations.

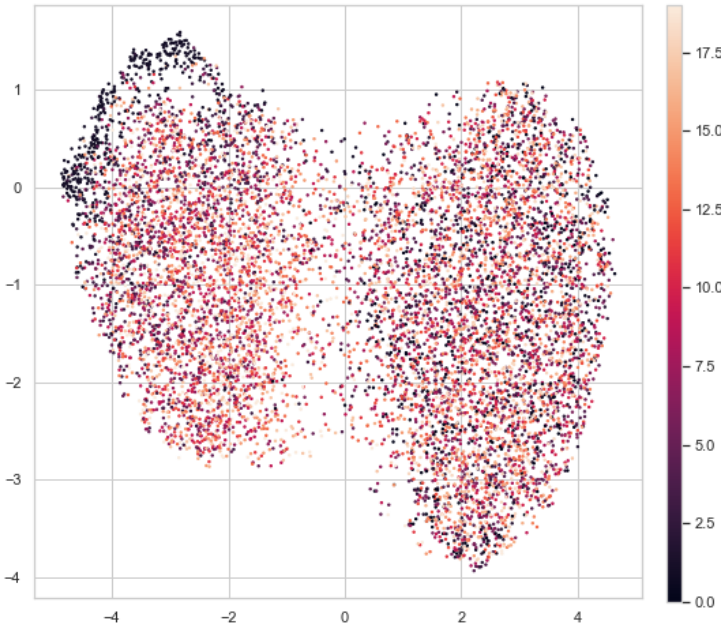


Figure 6.6: Embedding of the type locations transformed down into 2 dimension with UMAP using the modified network. Here more structure is seen as the embedding is split into two clusters. Also another cluster on the upper left is noticed.

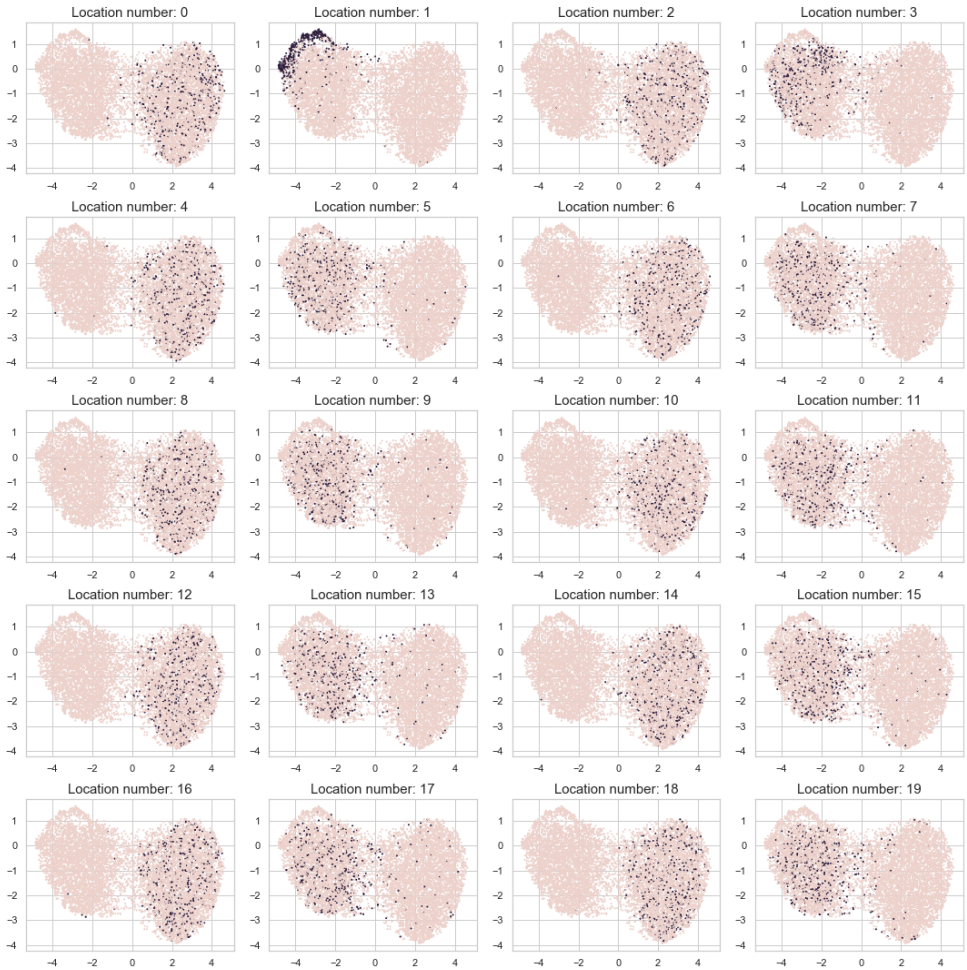


Figure 6.7: Embedding of the type locations transformed down into 2 dimension with UMAP using the modifed network colored individually according to location. The black dots represents the explored locations. Here we noticed how odd locations are in the left cluster and even locations are in the right. Other than that we see how location 1 has a clear location compared to the others. An explanation of the split could be due to the network trying not to guess on itself. Unfortunately we still do not see any geographical connection between the embeddings.

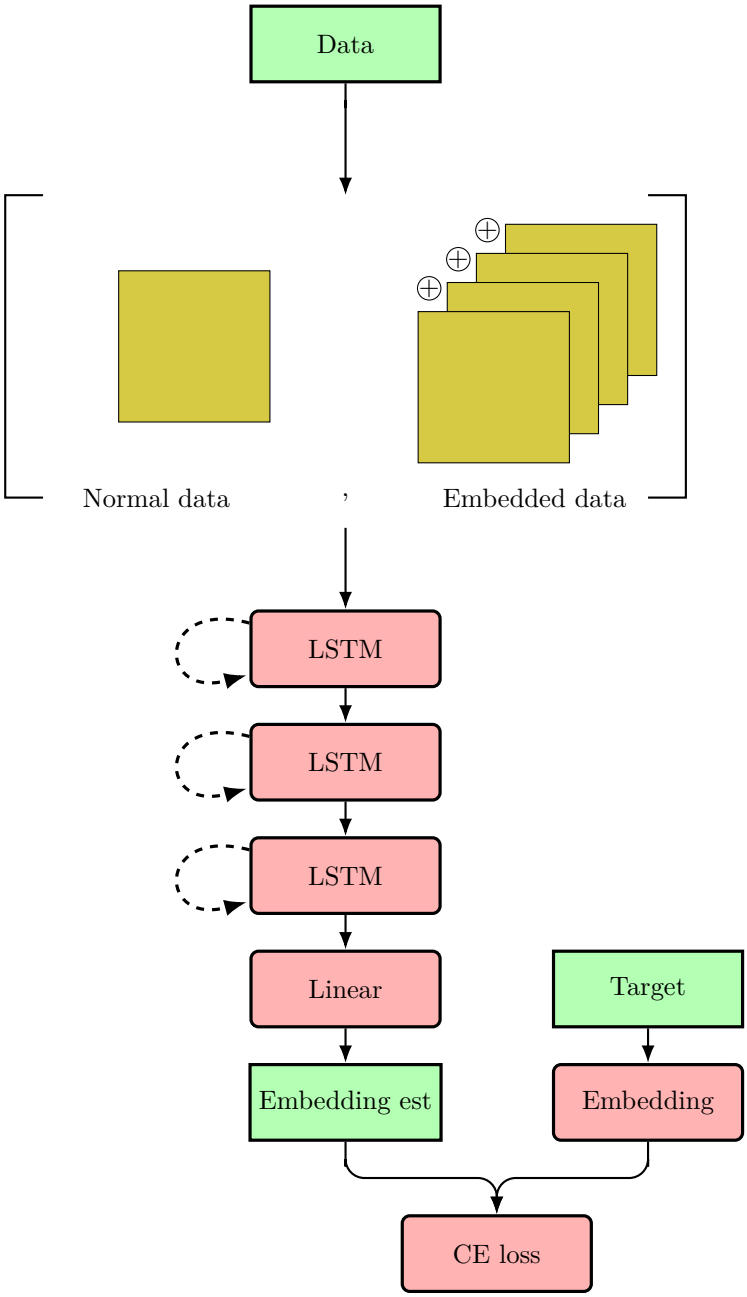


Figure 6.8: Graphical illustration of the Final network. Here the red boxes is some kind of transformation and the green boxes is a clarification of what the data is at the point in the network.

6.2.1 Prediction vs entropy

Looking back at the entropy it would be interesting to know if the prediction rates had any correlation with the upper bound of predictability that was computed from the estimated entropy. Using the prediction rates on the validation data, we plotted the individual prediction rates vs the corresponding estimated upper bound of prediction, see Figure 6.9. A positive correlation is clearly present, with a Pearson correlation coefficient of 0.47. The outliers are likely due to the last 10% of trajectories not always being representative of the whole trajectory, for example if the user was on a vacation, etc. This was also the result we were hoping for since it shows that the network follows the complexity of the data, i.e. the lower the entropy of the data, the more structure should be present.

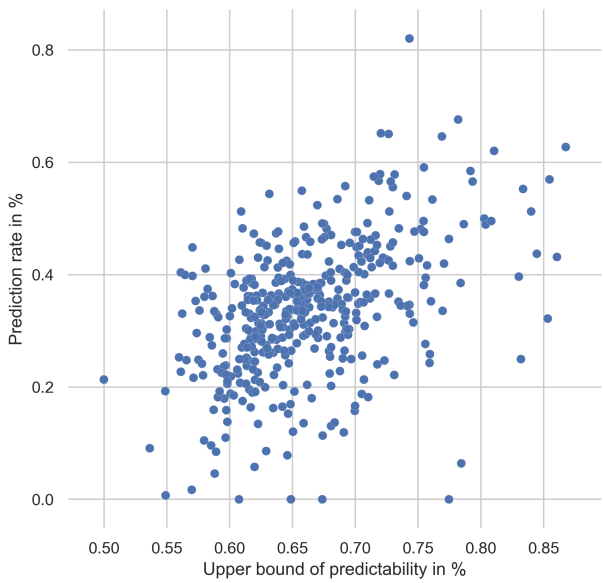


Figure 6.9: A scatter plot of the correlation between the estimated upper bounds of predictability and the prediction rates of the network for each user. We notice how they are positively correlated just as desired. This shows that our network uses more information of the sequence if it is present. We also see some outliers which are due to that the last 10% of the trajectories are not always being represented in the previous 90% due to the high amount of explorations etc. So if a user is on vacation the prediction rate will be 0%.

6.2.2 Exploration prediction

As mentioned several times one of the great challenges of predicting human mobility is the amount of explorations to locations never seen previously. From the data exploration, chapter 5, we know that 21.6% of all locations are explorations which effectively means that we cannot predict those 21.6% of locations. This does not exclude us from trying to predict if someone is doing an exploration; not where they are going, but just that they are in for an adventure. To do this we use the existing network and expand on it. We simply add another decoding linear layer at the end which takes the output from the LSTM layers and transforms it down into a 2 dimensional output to tell us if it is an exploration or not. When computing the loss we use cross entropy on the output and then does a weighted sum of the location prediction loss. This way we try to balance the networks learning so that not one prediction counts more than the other.

Training the network we quickly reached a prediction rate of $\sim$ 79%, which is basically just that the network always guesses on "not exploration". After that the prediction rate slowly increased to $\sim$ 81%, but we quickly decided that predicting actual explorations were not that feasible. Instead looking at the probability of an exploration made more sense. In reality what the network does is exactly outputting two probabilities, one for an exploration and one for not an exploration. The network is then trying to minimize the cross entropy "error" from these probabilities, so looking at the receiver operating characteristic (ROC) and computing the area under the curve (AUC) made more sense. In Figure 6.10 we plotted the ROC curve with the corresponding AUC= 0.71. Depending on the rate of false positives, that we are willing to accept, we can select a cutoff for our model. We will not select a cutoff since it is an problem specific value. Now however the network will spit out a probability of the next location being an exploration, together with the location that the network predicts, given it is not an exploration.

6.2.3 Transfer learning

To see if we could increase the performance of the network, we tried to do some extra training on each individual user, in which we knew from early testing very quickly resulted in overfitting. However letting the network just get the feel of the individual might improve the performance. To test this theory we ran every user through our pre-trained network and saved the performance on their test set every second epoch for 12 epochs in total. The pre-trained network had an average accuracy of 33.6% which was increased with extra training as shown in Figure 6.11. Here it can be seen that training the network after 6-8 epochs seemed to slightly boost the performance. It went up from 33.6%, see Figure 6.3, to 34.2% after 6 epochs. This was tested several times with different initial networks and the result was the same. Here we just accumulated all the predictions for each epoch, but better and more flexible measures of when the training began overfitting could be explored further.

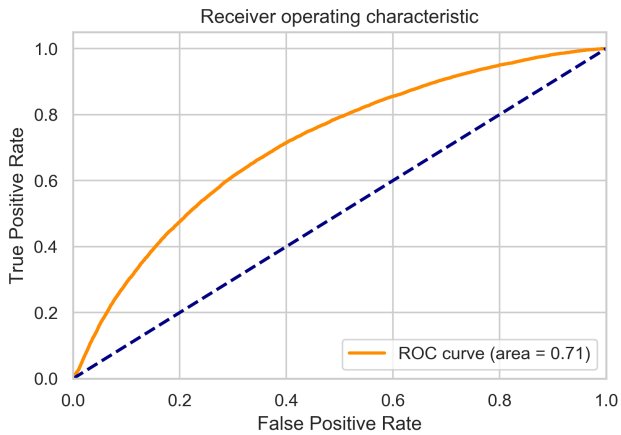


Figure 6.10: ROC curve of the exploration predictions. We notice how the ROC curve (orange) is well above random guessing (blue) but not very confident in its choices. However depending on application a trade off can be chosen between the true positive rate (TPR) and false positive rate (FPR). We also see that the area under the curve (AOC) is 0.71.

6.3 Test set predictions

Now that the process of how we wanted to train the network was complete we could finally see how it performed on the test data. Using the pre-trained network with an average prediction accuracy on the validation set of 33.6% we now got the result shown in Table 6.4.

	Top@1	Top@3	Top@5	Exploration
Final network	34.0%	45.9%	50.1%	81.4%

Table 6.4: Average prediction rate for the network on the test set.

Looking at the prediction rate of the test set it was actually 0.3% higher than on the validation data. This difference could very well be due to the lower amount of test data, but it confirms that it has not overfit. Again we wanted to see if we could boost the performance, and we therefore trained every user individually again for 6 epochs. Since we already had the setup to do it for every second run, we noted the performance for those runs, see Figure 6.12. The full updated result with 6 epochs of extra training is shown in Table 6.5. We still see a boost in performance when training the network on the individual user for a few epochs, without the network overfitting. All of the *top@k* scores increased, see Table 6.5, which are higher than previous suggested models performances as evaluated by Gau et al. [28], see Table 1.1.

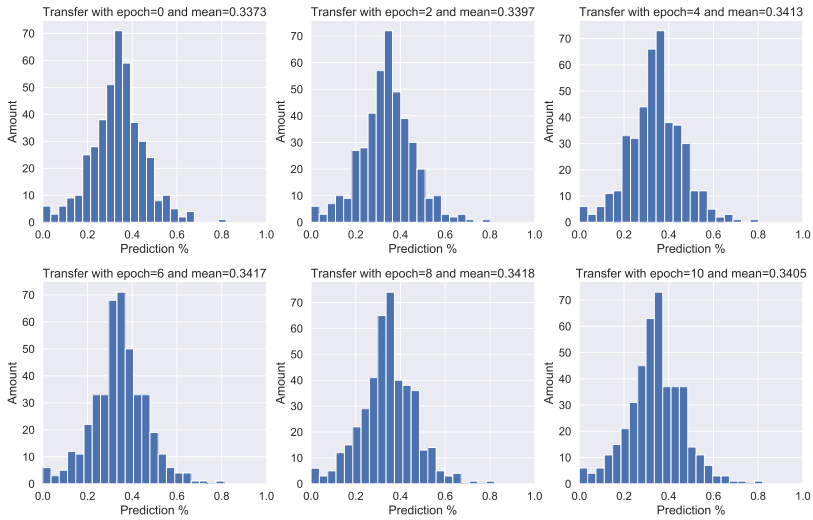


Figure 6.11: Distribution of prediction rates after extra training epochs on every single user. We see that training for 6-8 epochs on each individual might improve performance of predictions.

	Top@1	Top@3	Top@5	Exploration
Final network	34.0%	45.9%	50.1%	81.4%
Final network with transfer	34.6%	46.2%	50.4%	×

Table 6.5: Average prediction rater for the final network on the test set.

To get a better understanding of what the network is predicting, we plotted a trajectory of a random user, see Figure 6.13. To illustrate the process, we selected a starting point, and then showed the true next location and the predicted next location. The previous visited trajectory was colored red, with the current location colored black, the next true location and path colored purple, and lastly the predicted location and path colored blue. In Figure 6.13a we see that both paths are going to DTU - Lyngby campus, but the network was unsure exactly where. In the next step, Figure 6.13b, the network thinks the user is going back to where it previously came from, where in reality he/she headed to central Copenhagen. In the third step, Figure 6.13c, the network and true location is headed for the same approximate location, but not exactly the same, etc. This opened up for another question; even though the network is not always guessing the exact location correct, is it still close to the actual location?

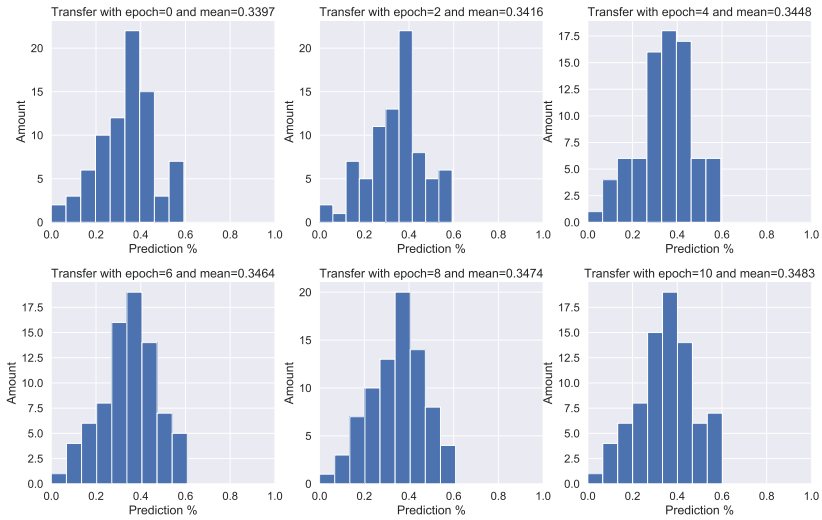
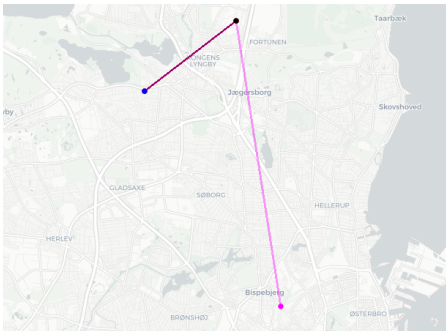


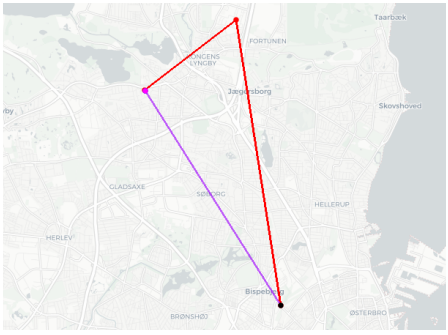
Figure 6.12: Distribution of prediction rates after different extra learning epochs on every single user in the test set. Again we see that training for 6 epochs more improves performance of the network.

Due to the average prediction rates on the test being very similar to the rates on the validation set, we assumed the network had not overfitted, and to get a better approximation of the distance between prediction and true location we used the full data set to calculate the distance between locations and predictions. Using the network with a prediction rate of 33.6% on the validation set, we calculated the distance between the predicted and true location using Vincenty distance. Since some of the distances are insanely large, e.g. if someone went on vacation to the other side of the earth, we plotted the cumulative fraction amount of users vs the logarithmic distance in km, see Figure 6.14a. Due to the huge difference in distances we zoom in and plot the distribution of the first $\sim$ 12km, see Figure 6.14b. It is seen that 50% of all guesses lies within a distance of 1 km of the actual prediction. This could however change depending on the geographical location of the users in which we need other data sets to test. Comparing the next prediction distance to the distance of all location switches, see Figure 6.14c, we see that we are actually doing worse than just guessing on the same location since the cumulative fraction is higher in Figure 6.14c. This suggests that if we could add more information about distances in the network we could maybe achieve better results. Looking back at Figure 6.13, which showed an example of the path vs prediction, it did not get one prediction correct in these six predictions, but several predictions which were very close to the true prediction.

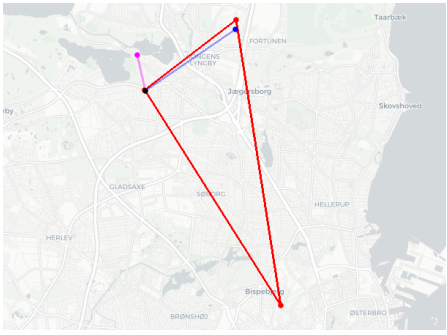
(a) Step 1.



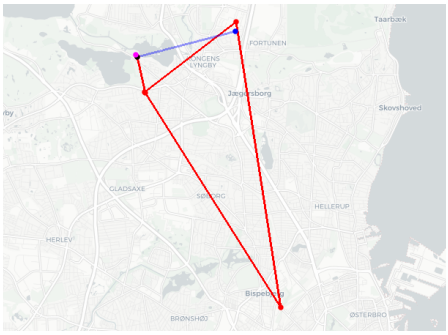
(b) Step 2.



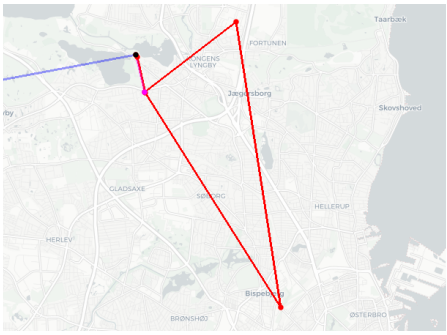
(c) Step 3.



(d) Step 4.



(e) Step 5.



(f) Step 6.

Figure 6.13: (a)-(f) is a timeline showing the trajectory (red) of a path with the predicted (blue) and true (purple) next location. We see how the predicted location was never the actual location but often very close to the real location.

Figure 6.14 shows that this is true for a lot of predictions.

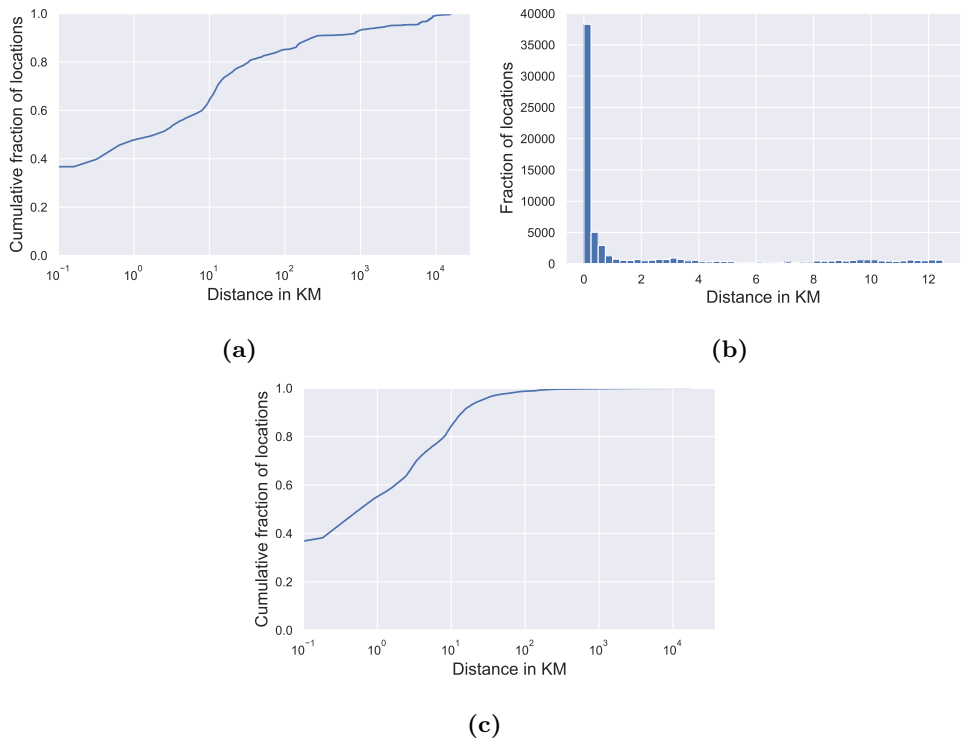


Figure 6.14: Plots showing the Vincenty distance of the predicted location to the actual location. (a) Shows the cumulative fraction of locations vs the distance (in km) between the predicted and true location. (b) Shows the zoomed in distribution of the fraction of locations which has distances less than $\sim$ 12km between the predicted and true location. (c) Shows the cumulative fraction of location switches vs the distance moved for all data points. Overall it is shown that $\sim$ 50% of all locations are within a distance of 1 km of the true location, and that we are not doing better than guessing on the same location all the time distance wise.

CHAPTER 7

Conclusion

In this thesis we have explored the next location predictability of the Sensible DTU data set. Specifically the next location sequences constructed from continuous GPS data by Cuttone et al [32]. To get the best quality of the data, we introduced Shannon entropy and how it could be converted into an upper bound of predictability for time series. The entropy was then estimated for every user to see which of the 851 user sequences that had converged. A convergence criteria together with only looking at location sequences longer than 1000 locations left us with 507 users which together gave us an upper bound of predictability of 65.9%.

We then split our data into a training, validation and test set, and fitted 6 baselines on the training set and tested them on the validation set. This gave us a lower bound of predictability of 26.7% which came from a Markov model order 1 with a small twist.

Our recurrent neural network model, based on long short term memory (LSTM) models, was then introduced with all the choices and details of construction. The first iteration of the model achieved an average prediction rate of 28.0%, which we boosted to 34.0% after introducing a method of combining generalized and individual user information with embeddings. Using transfer learning to continue training on each individual (which quickly overfitted), for just a few epochs, we boosted the average prediction rate up to 34.6%.

Since the data consists of categorical variables we had to embed them in a higher, 13, dimensional space for the network to comprehend. The network also trained the embedding, so we hoped that the network would find some geographical or location based structure (home, work, etc.). This however was not achieved. This could be due to the network learning these connections in the LSTM layers, or that it is something totally different learned. Further research is encouraged.

Lastly due to the high amount of explorations, 21.6% of the locations, we tried to let the network predict if the next location was an exploration. Due to the skewed distribution of explorations vs non-explorations the network mostly learned to never predict exploration. We instead looked at AUROC which had an AUC of 0.71. So depending on the application a trade-off could be chosen between the false positive rate and true positive rate.

In total we achieved to create a LSTM neural network with a prediction rate way above the baselines presented. We also achieved a higher prediction rate than previous presented models, but due to the dependence of the data, see Table 1.1,

it is hard to compare them without trying the models head to head on the same data set. The sensible DTU data mostly consists of students which might have more predictable trajectories or vice versa.

Encouraged further work includes comparing our model against other models on the same data, doing an ablation analysis (removing features of the model to see what contributes the most), include geographical/relative information about distances, and look into a better measure of when transfer learning starts overfitting.

Bibliography

[1] Douglas Adams. *The hitchhiker's guide to the galaxy*. Hitchhiker's Guide to the Galaxy Series. Harmony Books, 1980. ISBN: 9780517542095.

[2] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pages 115–133.

[3] Jeffrey Travers and Stanley Milgram. "An experimental study of the small world problem". In: *Social Networks*. Elsevier, 1977, pages 179–197.

[4] *Number of monthly active Facebook users worldwide as of 3rd quarter 2019*. `https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/`. Accessed: 23-01-2020.

[5] Serdar Çolak, Antonio Lima, and Marta C. González. "Understanding congested travel in urban areas". In: (February 2016).

[6] Xin Lu, Linus Bengtsson, and Petter Holme. "Predictability of population displacement after the 2010 Haiti earthquake". In: *Proceedings of the National Academy of Sciences* 109.29 (2012), pages 11576–11581. DOI: `10.1073/pnas.1203882109`. eprint: `http://www.pnas.org/content/109/29/11576.full.pdf`.

[7] Nicholas Lane et al. "Bewell: A smartphone application to monitor, model and promote wellbeing". In: *5th ICST/IEEE Conference on Pervasive Computing Technologies for Healthcare*. ISBN dukket op på konferencen, men jeg kan stadig ikke finde paperet i IEEEs database; har derfor sat registreringen til "for approval" /haal. IEEE Press, 2011. ISBN: 978-1-61284-767-2.

[8] Chaoming Song et al. "Limits of Predictability in Human Mobility". In: *Science* 327.5968 (2010), pages 1018–1021. ISSN: 0036-8075. DOI: `10.1126/science.1177170`. eprint: `http://science.sciencemag.org/content/327/5968/1018.full.pdf`. URL: `http://science.sciencemag.org/content/327/5968/1018`.

[9] Michael Batty et al. "Smart cities of the future". In: *The European Physical Journal Special Topics* 214.1 (2012), pages 481–518.

[10] A. Cuttone, S. Lehmann, and M. C. González. "Understanding Predictability and Exploration in Human Mobility". In: *ArXiv e-prints* (August 2016). arXiv: `1608.01939 [cs.CY]`.

[11] Dirk Brockmann, Lars Hufnagel, and Theo Geisel. "The scaling laws of human travel". In: *Nature* 439.7075 (2006), pages 462–465.

[12] Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. "Understanding individual human mobility patterns". In: *nature* 453.7196 (2008), pages 779–782.

[13] Injong Rhee et al. "On the levy-walk nature of human mobility". In: *IEEE/ACM transactions on networking* 19.3 (2011), pages 630–643.

[14] Kai Zhao et al. "Explaining the power-law distribution of human mobility through transportationmodality decomposition". In: *Scientific reports* 5.1 (2015), pages 1–7.

[15] Chaoming Song et al. "Modelling the scaling properties of human mobility". In: *Nature Physics* 6.10 (2010), pages 818–823.

[16] Augustin Chaintreau et al. "Impact of human mobility on opportunistic forwarding algorithms". In: *IEEE Transactions on Mobile Computing* 6.6 (2007), pages 606–620.

[17] Jameson L Toole et al. "Coupling human mobility and social ties". In: *Journal of The Royal Society Interface* 12.105 (2015), page 20141128.

[18] Dashun Wang et al. "Human mobility, social ties, and link prediction". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2011, pages 1100–1108.

[19] Miao Lin, Wen-Jing Hsu, and Zhuo Qi Lee. "Predictability of individuals' mobility with high-resolution positioning data". In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM. 2012, pages 381–390.

[20] Gavin Smith et al. "A refined limit on the predictability of human mobility". In: *2014 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE. 2014, pages 88–94.

[21] E. Lind Ikanovic and A. Mollgaard. "An alternative approach to the limits of predictability of human mobility". In: *ArXiv e-prints* (August 2016). arXiv: `1608.06419 [physics.soc-ph]`.

[22] Eunjoon Cho, Seth A Myers, and Jure Leskovec. "Friendship and mobility: user movement in location-based social networks". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, pages 1082–1090.

[23] Adam Sadilek, Henry Kautz, and Jeffrey P Bigham. "Finding your friends and following them to where you are". In: *Proceedings of the fifth ACM international conference on Web search and data mining*. ACM. 2012, pages 723–732.

[24] Salvatore Scellato et al. "NextPlace: a spatio-temporal prediction framework for pervasive systems". In: *International Conference on Pervasive Computing*. Springer. 2011, pages 152–169.

[25] Huiji Gao, Jiliang Tang, and Huan Liu. "Mobile location prediction in spatio-temporal context". In: *Nokia mobile data challenge workshop*. Volume 41. 2012, page 44.

[26] Xin Lu et al. "Approaching the limit of predictability in human mobility". In: *Scientific reports* 3 (2013).

[27] Adam Sadilek and John Krumm. "Far Out: Predicting Long-Term Human Mobility." In: *AAAI*. 2012.

[28] Qiang Gao et al. "Predicting human mobility via variational attention". In: *The World Wide Web Conference*. 2019, pages 2750–2756.

[29] Jie Feng et al. "Deepmove: Predicting human mobility with attentional recurrent networks". In: *Proceedings of the 2018 world wide web conference*. 2018, pages 1459–1468.

[30] Arkadiusz Stopczynski et al. "Measuring large-scale social networks with high resolution". In: *PloS one* 9.4 (2014), e95978.

[31] Oliver P John, Sanjay Srivastava, et al. "The Big Five trait taxonomy: History, measurement, and theoretical perspectives". In: *Handbook of personality: Theory and research* 2.1999 (1999), pages 102–138.

[32] Andrea Cuttone, Sune Lehmann, and Marta C González. "Understanding predictability and exploration in human mobility". In: *EPJ Data Science* 7.1 (2018), pages 1–17.

[33] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Kdd*. Volume 96. 34. 1996, pages 226–231.

[34] Laura Alessandretti, Ulf Aslak, and Sune Lehmann. "The Scales of Human Mobility". unpublished.

[35] Alfred Wehrl. "General properties of entropy". In: *Reviews of Modern Physics* 50.2 (1978), page 221.

[36] L Carnot. *Principes fondamentaux de l'équilibre et du movement [Fundamental Principles of Equilibrium and Movement]*. 1803.

[37] Claude E Shannon. "A mathematical theory of communication". In: *Bell system technical journal* 27.3 (1948), pages 379–423.

[38] Jacob Ziv and Abraham Lempel. "Compression of individual sequences via variable-rate coding". In: *IEEE transactions on Information Theory* 24.5 (1978), pages 530–536.

[39] I. Kontoyiannis et al. "Nonparametric Entropy Estimation for Stationary Processes and Random Fields, with Applications to English Text". In: *IEEE Trans. Inf. Theor.* 44.3 (September 2006), pages 1319–1327. ISSN: 0018-9448. DOI: `10.1109/18.669425`. URL: `http://dx.doi.org/10.1109/18.669425`.

[40] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[41] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: *NIPS-W*. 2017.

[42] Junyoung Chung et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).

[43] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pages 1735–1780.

[44] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[45] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pages 249–256.

[46] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *International conference on machine learning*. 2013, pages 1310–1318.

[47] Bjørn Sand Jensen et al. “Estimating human predictability from mobile sensor data”. In: *2010 IEEE International Workshop on Machine Learning for Signal Processing*. August 2010, pages 196–201. DOI: 10.1109/MLSP.2010.5588997.

[48] Peter Edsberg Møllgaard. “Understanding the Representativeness of Mobile Phone Location Data in Characterizing Human Mobility Indicators”. English. 2017.

[49] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).