Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Erlend Løkken, Endre Waatevik

Location prediction using neural networks

Master's thesis in Computer Science
Supervisor: Heri Ramampiaro
June 2019

NTNU
Norwegian University of
Science and Technology

Erlend Løkken, Endre Waatevik

Location prediction using neural networks

Master's thesis in Computer Science
Supervisor: Heri Ramampiaro
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Preface

The process of this master theses started in late January 2019 with an initial meeting with Heri. Erlend and Endre were introduced to the subject, and the content of the project was discussed. We are grateful for the opportunity to write our master thesis on the subject of text-based location prediction and would like to thank the following persons for their involvement

- Heri Ramampiaro, NTNU. For supervising and guidance.

Date: 11th June 2019
Place: Trondheim

Endre Waatevik Erlend Løkken

Abstract

Identifying the tweet location is crucial in order to utilize the content in studies of regional user behavior. Such information can be used in numerous applications depending on geographical information such as event detection and location-based recommendation. In terms of Twitter, such geographical information is reported to be present in just 1-3 % of all tweets. Thus, the inference of location on non-geotagged tweets is an active research area in geographical information retrieval. In this project we explore the use of deep learning to issue the geolocation problem, and propose a method using recurrent neural networks. The proposed method predicts tweet locations based on information given in a single tweet, where we explore both models based solely on tweet text and models exploiting additional contextual metadata. In this work the geographical area of interest is divided into grid cells, where a neural network is trained to predict the grid cell with highest probability of containing a given tweet. A comparison of uniform and adaptive grid cells is conducted with ambiguous results in terms of the preferable approach for geographical modelling. The evaluation on three different datasets indicates that the proposed method yields a significant improvement compared to state of the art approaches. In terms of exploiting contextual metadata features in combination with text the evaluation yields significantly better accuracy than utilizing text only.

Sammendrag

For å kunne studere brukeres bevegelser og handlinger er det viktig å identifisere lokasjonen en tweet er skrevet fra eller omhandler. Denne typen informasjon kan brukes i en rekke applikasjoner som avhenger av geografisk informasjon, som for eksempel event deteksjon og lokasjons-basert rekommendering. På Twitter rapporteres det at geografisk informasjon bare er tilgjengelig på 1-3 % av alle tweets. Dette har gjort at predikering av lokasjon på tweets uten geografisk merking har blitt et aktivt forskningsområde innen geografisk informasjonsgjenfinning. I dette prosjektet utforsker vi bruken av deep learning for å løse dette problemet, og presenterer en metode som baserer seg på bruk av nevrale nettverk. Metoden som presenteres predikerer lokasjonen basert på innholdet i en enkelt tweet, hvor vi ser på modeller som kun baserer seg på tekst, og modeller som benytter tekst i kombinasjon med annen kontekstuell metadata. I dette arbeidet deles det geografiske området inn i celler, der et nevralt nettverk benyttes for å finne den cellen med høyest sannsynlighet for å inneholde en gitt tweet. For å finne optimal inndeling av det aktuelle geografiske området utforsker vi, og sammenligner både uniforme og adaptive celler. Evalueringen som er foretatt på tre forskjellige datasett indikerer at den presenterte metoden gir en signifikant forbedring sammenlignet med moderne tilnærminger for samme problem. Når det gjelder utnyttelse av metadata viser evalueringen at bruk av ekstra data som *opprettingstidspunkt*, *brukerens språk* og *brukerens profilbeskrivelse* gir mer presise predikeringer.

Contents

List of Figures

List of Tables

1 Introduction

Determining explicit location information of social media users and associated messages provides a valuable resource for a wide range of applications. Such applications includes location-based recommendation, event detection and targeted advertisement. There exists numerous social media platforms, and one of the most popular ones is Twitter. As of the fourth quarter of 2018 this micro-blogging platform averaged 321 million active users monthly. Twitter enables users to post "tweets" of 280 characters, up from 140 the recent years, and share them with their followers. The availability of twitter data through the official Twitter API's makes twitter a great source for academic research. Due to the increasing popularity of GPS-enabled devices, more and more tweets have explicit location information such as latitude-longitude attached. However, such tweets are reported to constitute only 1-3 % percent of all tweets (Li, Eickhoff, & de Vries, 2014). In addition to metadata based locations, is it important to determine the location a tweet relates to.

For this reason it is necessary to infer locations for Twitter users and tweets in order to improve the quality of the location based applications mentioned above. Inferring such information based on tweet text is a research area that has received a considerable amount of attention, with various approaches being used. Among the most researched approaches are statistical methods based on Multinomial Naive Bayes and Kullback-Leibler (KL) divergence, while deep learning methods like neural networks have received increasing attention the recent years.

Crucial for location inference methods is the modelling of the geographical region. A widely used technique for statistical text-based approaches is to model the region of interest into a grid, and apply document classification methods to estimate the grid cell most probable of containing a given tweet. This technique for geographical modelling has not been widely adopted in studies exploiting neural networks, where the geographical modelling mainly focuses on city or country level targets. Considering the recent growth in popularity, and the fairly limited amount of research adopting neural networks in geolocation problems, this definitely deserves further attention. Utilizing deep learning on the geolocation problem enables the task to either be handled as a classification problem, or as a multi-target regression problem.

The goal of this project is to further investigate the use of deep learning for the geolocation problem, and investigate its performance compared to state of the art statistical approaches. We propose a method using recur-

rent neural networks for predicting the geolocation of tweets. Specifically, we are developing neural models able to predict geographical coordinates on a tweet-by-tweet basis utilizing the grid based approach for modelling. Developed models will be based both on tweet text and tweet text combined with contained meta data, and tested on plural different geographical areas.

The main contribution of this thesis are summarized as follows: (1) we investigate the use of neural networks to issue the geolocation problem of tweets using grid cells for geographical modelling, and compare the performance against state of the art statistical methods. (2) We explore the use of contextual features in combination with text and investigate the performance in comparison with neural models utilizing text only. (3) we examine both the uniform and adaptive grid approach for geographical modelling, and compare the performance on multiple geographical regions. (4) we investigate the performance of two methods for coordinate deduction based on a predicted grid cell, where we compare geographical center against the centroid of contained data points.

The remainder of this project is organized by presenting the problem overview in section 2 and the necessary background theory in section 3. In section 4 we provide an overview of related work for Twitter location prediction. In section 5 we describe details about the proposed method. Evaluation of the method and useful comparisons are described in section 6. Finally, we conclude the paper and describe further work in section 7.

2 Problem Overview

In this chapter we give an overview of the problems discussed in this project. This includes an introduction to the Twitter platform and the information available in terms of tweet content, tweet context, twitter network, and an introduction to geolocation problems on Twitter.

2.1 The Twitter platform

Twitter is a micro-blogging platform and one of the most popular online social networks. It was launched in 2006 and has more than 300 million active users monthly around the world. The popularity of Twitter, together with frequent user updates generates a large volume of data at a high velocity. This data is accessible at different grants and levels through The Twitter API delivered on the Twitter Developer Platform. The short and noisy nature of tweets, the availability of data, the rich contextual information and the broad Twitter network makes Twitter a great source for a wide range of studies including geolocation problems.

A tweet is a user-generated piece of text limited by a certain length of characters. The short nature of tweets is related to the max-length originally being set to 140 characters. This was doubled and raised to 280 characters in late 2017, but according to Twitter this has not dramatically changed the length of Twitter posts. When posting tweets a user may include links, images, videos, hashtags which are words starting with #, and mentions, which is another user's or business name by a preceding @. A user may also retweet posts written by other users. These hashtags, mentions, retweets and common Twitter slang vocabulary composes the noisy nature of tweets.

Twitter posts contains rich contextual information, and is more than just a short piece of text. Attached to a tweet are information like posting timestamp, user information, and optionally GPS coordinates supported by the current grow of mobile devices. The attached user information depends on the completeness of the current user profile, which is optional for a user to submit. A profile may contain information like timezone, home cities and country. All this contextual information is crucial in further understanding the content of tweets. An example tweet is shown in listing 1.

Listing 1: Example tweet


```
{
  "created_at": "Fri Feb 15 10:25:56 +0000 2019",
  "id": 1096354919274078209,
  "text": "Stop staring at the clock every day. It's time for a new career",
  "user": {
    "id": 23263441,
    "name": "TMJ– LON Acct. Jobs",
    "screen_name": "tmj_lon_acct",
    "location": "London",
    "url": "http://bit.ly/2I6YXx9",
    "description": "Follow this account for geo-targeted Accounting job tweets in London.",
    "translator_type": "none",
    "followers_count": 372,
    "friends_count": 312,
    "statuses_count": 234,
    "created_at": "Sun Mar 08 02:00:23 +0000 2009",
    "utc_offset": "None",
    "time_zone": "None",
    "geo_enabled": "True",
    "lang": "en",
  },
  "geo": { "type": "Point", "coordinates": [51.5073509, -0.1277583] },
  "coordinates": { "type": "Point", "coordinates": [-0.1277583, 51.5073509] },
  "retweet_count": 0,
  "favorite_count": 0,
  "entities": {
    "hashtags": [],
    "urls": [],
    "user_mentions": [],
    "symbols": []
  },
  "favorited": "False",
  "retweeted": "False",
  "lang": "en",
  "timestamp_ms": "1550226356701"
}
```


Another part of Twitter that makes the foundation for multiple studies is the Twitter community. Twitter user u_a has both a list of followers who are notified by the posts of u_a, and a list of followees who is followed by u_a. Unlike Facebook, where relationships are bidirectional, relationships on Twitter is unidirectional. This means that if user u_a follows user u_b, it does not necessarily mean u_b is following u_a. The Twitter network, context and tweet contents itself are sources of information that can be utilized in inference studies on Twitter.

2.2 Geolocation problems on Twitter

The focus of this project will be on geolocation prediction on Twitter. Locations on Twitter can be divided into three main categories, namely Home location, Mentioned location and Tweet location. All three categories have been widely studied in previous research, and the targeted category for this project will be the Tweet location. This is the task of inferring the geographical origin of where a tweet is posted from.

The ground truths for tweet locations are collected from the optional contextual information of GPS coordinates as mentioned in chapter 2.1. Due to the ground truth being represented by coordinates, this is the most common representations of tweet locations, instead of country, administrative region or city, that are widely adopted representations used in studies for Home and Mentioned location prediction. Inference of tweet location draws a more complete picture of a user's movements, and may extend the available contextual information. Such information may be utilized in a number of location based applications.

A list of typical contextual features that may contain useful information in terms of location inference are provided in table 1

Contextual feature	Type
Tweet text	Text
User description	Text
User name	Text
User profile location	Text
Tweet text language	Categorical
User Language	Categorical
Timezone	Categorical
Posting Time	Timestamp

Table 1: List of contextual features for location inference

2.3 Problem specification

The main objective of this project is to examine the use of deep learning to extract spatial information from text. More specifically, we will explore the use of neural networks to infer location of tweets using Twitter datasets. The main hypothesis we want to investigate is if neural models will perform better than well studied state of the art methods like Multinomial Naive Bayes and Locality-adapted Kernel Densities. In further investigation of geolocation problems we will look into limitations of previous studies. This includes different approaches for geographical modelling and inference of coordinates. Regarding geographical modelling we explore the differences in terms of most accurate predictions between uniform and adaptive grid cells. Due to the rich amount of available metadata on Twitter we also examine the use of various contextual features in combination with text, and explore if such features significantly improves the prediction accuracy. The research questions we want to answer are as follows:

- How to exploit deep learning to further improve the inference of tweet locations?
- How will the geographical modelling effect the performance of the location inference?
- How to derive coordinates from geographical space partitioned into grid cells?
- How to incorporate metadata features into the model, and how will this effect the performance?

3 Background Theory

This chapter introduces literary studies and background theory on deep learning topics and approaches for issuing the geolocation problem. Previous studies on the research area covers a broad range of different methods, but some of them obsolete and therefore unrelated in terms of the intention of this thesis. The topics presented in this chapter will cover only the most relevant deep learning techniques and state of the art statistical approaches regarding the research goals.

3.1 Artificial Neural Networks

Artificial Neural Networks (ANN) is a Machine Learning architecture that has gained a lot of attention in recent years. ANNs have become one of the most efficient and powerful methods to solve complex problems in computer vision, NLP, speech recognition, image, audio, and video generation which have made ANNs the state of the art technology that drives the AI revolution (Goltsman, 2017).

An ANN can be thought of as a computational model that is inspired by biological neural networks in the brain. They commonly consist of hundreds of simple processing units (neuron) wired together in a complex layer structured communication network. Each processing unit is a simplified version of a real neuron which sends off new signals and fires if strong input signals from connected neurons are received.

3.1.1 Architecture

A model of a simple neural network can be seen in figure 1. In this figure neurons are connected by weights (input signals) in a typical layer structure. The input layer passes the information to the next layer without doing any computation. In the output layer an *activation* function is used to map data to the desired output format. In addition, a neural network can consist of none or multiple layers in between the input and output layer. This is where intermediate processing and computation is done. These layers are referred to as *hidden* layers because we don't observe the values computed in these layers.

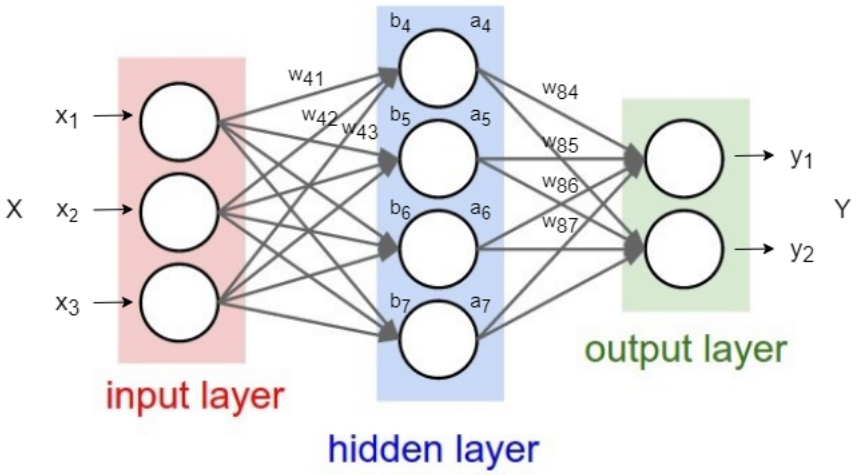


Figure 1: A model of a simple neural network

The activation of a given neuron is given by

$$a_i = g((\sum_{j=1}^{n} w_{ij}a_j) + b)$$

where g denotes the activation function, b denotes the bias, w denotes the weight and a_j denotes the activation of the incoming neuron. To calculate activations in the different layers we calculate the activation functions based on the input values. From figure 1 the activations in the hidden layer is given as

$$\begin{aligned} a_4 &= g(w_{41}x_1 + w_{42}x_2 + w_{43}x_3 + b_4) \\ a_5 &= g(w_{51}x_1 + w_{52}x_2 + w_{53}x_3 + b_5) \\ a_6 &= g(w_{61}x_1 + w_{62}x_2 + w_{63}x_3 + b_6) \\ a_7 &= g(w_{71}x_1 + w_{72}x_2 + w_{73}x_3 + b_7) \end{aligned}$$

An activation function is a function that defines the output of a node given a set of inputs. Examples of activation functions are *ReLU* and *Sigmoid*. This can be seen as a digital network of activation functions in a computer chip circuit that can be *ON* (1) or *OFF* (0). The bias is a value that allows shifting the activation function to the left or right to indicate a neuron's activity.

Similarly as in the hidden layer, the activations for the output layer is calculated as

$$y_1 = g(w_{84}a_4 + w_{85}a_5 + w_{86}a_6 + w_{87}a_7 + b_8)$$
$$y_2 = g(w_{94}a_4 + w_{95}a_5 + w_{96}a_6 + w_{97}a_7 + b_9)$$

3.1.2 Learning

The standard way of training an ANN is to use supervised learning, this requires a set of inputs together with a set of corresponding outputs (targets). The learning process is about finding the correct weights and biases, and consist of three subtasks

- Make a forward pass (Calculate activations for all neurons)
- Calculate the error (Calculate the difference between the given output and the desired output)
- Make a backward pass (Backpropagation)

After each forward pass through the network the output layer contains a set of activations. Based on these output values a "cost" function is used to calculate the difference between the given output and the desired output. This function yield the cost of a single training example. This sum (cost) is small when the network confidently classifies the output correctly and large when the output differs considerably from the desired output.

Gradient decent is the technique of minimizing the cost function. This is done by using the negative gradient of the cost function calculated by using the Backpropagation algorithm (Dabel E. Rumelhart, 1988). Based on the average cost of all training examples the weights and biases are changed to most efficiently decrease the cost (Schmidhuber, 2014).

By using these techniques an ANN is able to learn how to classify input data correctly.

3.2 Convolutional Neural Networks

As mentioned in chapter 3.1, neural networks has become the state of the art technology for solving complex problems in many fields, including computer vision. Convolutional Neural Networks (CNNs) is a class of feed-forward neural networks composed of one or multiple convolutional layers. CNNs have shown superior results in image and speech applications because of its architecture and ability to process visual and other two-dimensional data (Wu, 2017).

3.2.1 Architecture

A CNN usually takes a tensor of order 3 as input, e.g., an image with H row and W columns, and 3 channels of RGB colors (Dimensions D) (Wu, 2017). A tensor can be generalized to higher-order matrices. The input sequentially goes through a series of processing, where each processing step is a layer as described in chapter 3.1.

The layers in a typical CNN can be seen in figure 2. The standard architecture for a CNN is to do a series of Convolution + Pooling operations, followed by a number of Fully-Connected layers.

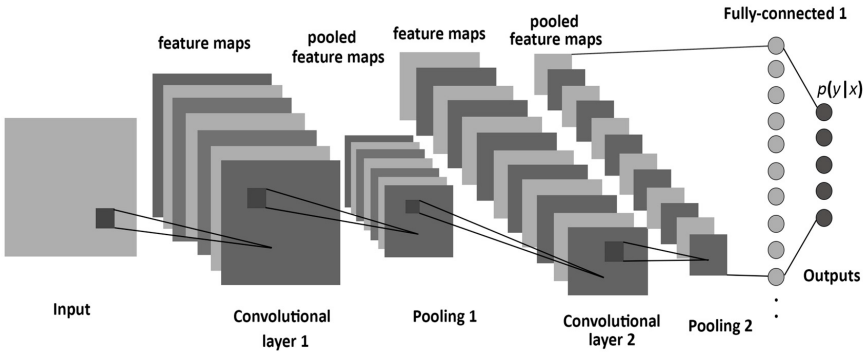


Figure 2: The structure of a simple CNN

3.2.2 Convolution

The Convolution layer is used to extract features from an input image by learning the features using small squares of input data (filters/kernels). This

is a mathematical operation with two inputs such as image matrix and filter matrix.

- Image matrix of dimension $(HxWxD)$
- Filter matrix of dimension $(w_H x w_W x D)$

The operation is given by the following equation

$$Y(j,k) = (X * w)(j,k) = \sum_{\gamma=-c}^{c} \sum_{\delta=-d}^{d} X(j+\gamma, k+\delta) w(\gamma, \delta)$$

where X denotes the upstream layer or image matrix, w denotes the kernel or filter and Y denotes the downstream layer or feature map.

The convolution is performed by a series of filtering operations. Filtering is the operation of lining up the filter with a position of the image matrix (image patch), starting in the upper left corner of X. The result of the mathematical operation yields the upper left entry of Y. The filter is then moved horizontally along X, one *stride* at a time, producing a new entry of Y each time it is applied. After completing a row, the filter is shifted down one *stride* and applied at the start of the row. This operation can be performed using different strides, which is the number of pixels to shift over the filter matrix. If the filter and the chosen stride does not fit perfectly to the image, there are two alternatives

- Pad the image matrix with zeros so that the filter fits
- Drop the part of the image where the filter does not fit, which results in shrinked size.

An example of this mathematical operation is shown with the following matrices

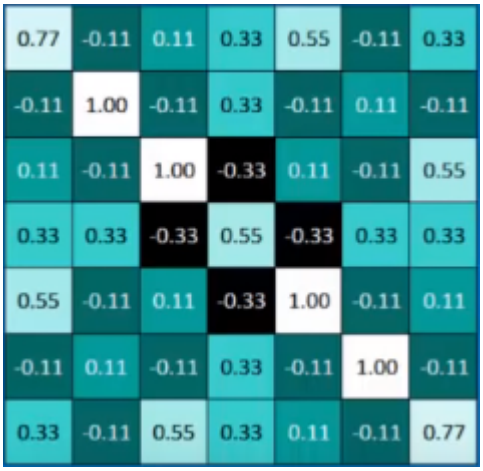
$$X_{HWD} = \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ \hline -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ \hline -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ \hline -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ \hline -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ \hline \end{array} \quad w_{HWD} = \begin{array}{|c|c|c|}\hline 1 & -1 & -1 \\ \hline -1 & 1 & -1 \\ \hline -1 & -1 & 1 \\ \hline \end{array}$$

An element-wise multiplication is applied to the filter and image patch, the results are added up and divided by the total number of pixels (Wu, 2017).

By applying the filter in the top left corner we get

$$= \begin{array}{|c|c|c|}\hline -1 & -1 & -1 \\ \hline -1 & 1 & -1 \\ \hline -1 & -1 & 1 \\ \hline \end{array} \bullet \begin{array}{|c|c|c|}\hline 1 & -1 & -1 \\ \hline -1 & 1 & -1 \\ \hline -1 & -1 & 1 \\ \hline \end{array} = \frac{(-1)+1+1+1+1+1+1+1+1}{9} = 0.77$$

This value is a measure of how well the filter is represented at the given position. Convolution is the repeated process of applying this filter on every possible position of the image matrix. By performing convolution on the given image and filter matrix using a stride of 1 results in the following feature map



A Convolution layer in a CNN consist of performing the described operation for multiple filters, resulting in multiple feature maps. By convolving an image, one image becomes a stack of filtered images.

3.2.3 Pooling

The convolution operation is usually followed by a pooling operation. This is an operation used to shrink the image stack and is performed by

1. Pick a window size
2. Pick a stride
3. Apply the window across the filtered images
4. From each window, pick a value (max, average, sum).

By performing a max pooling (most common) operation on the feature map from the convolution example, using a window size of $2x2$ and a stride of 2, we get the following result

The result from this operation is a shrinked image map where patterns are conserved.

To sum up, a CNN is a class of neural networks where convolution and pooling layers are stacked, together with other techniques used in ANNs to get the desired output.

3.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of feed-forward neural networks with the ability to retain a state that can represent information from an arbitrarily long context window. They are designed to recognize patterns in sequences of data that are related in time or space, such as text, snippets of audio, numerical time series etc. Data is sequentially processed one element at a time, with the ability to selectively pass information across sequence steps (Lipton & Berkowitz, 2015).

Training data for RNNs is typically a set that consists of a ⟨input sequence, target sequence⟩ pair, although commonly either the input or the output may be a single data point. An input sequence is, like the name suggests, the input to an RNN, where the sequence can be denoted as $(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(T)})$. Similarly, the target vector can be denoted as $(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, ..., \boldsymbol{y}^{(T)})$. Sequences may be of either finite or countably infinite length, where the maximum time index of the sequence is called T if finite. While sequences are often referred to as time-sequences, RNNs are applicable to non-temporal as well as to temporal tasks. Several domains have a defined order sequence with no explicit notion of time, like natural language. Consider the following example from Lipton et al.

"John Coltrane plays the saxophone", $\boldsymbol{x}^{(1)}$ = *John*, $\boldsymbol{x}^{(2)}$ = *Coltrane*, etc..

Word sequence with no temporal aspect.

The most fundamental architectural difference between a standard multilayer perceptron (MLP) and a RNN is the recurrent connections in the hidden layer allowing information to persist through multiple states. Figure 3 shows a simple schematic model of such networks with an input node, hidden layer with recurrent connection and output node. These recurrent connections enables the possibility of discovering temporal correlations between data points far away from each other (Pascanu, Mikolov, & Bengio, 2013).

All calculations necessary for a simple RNN (3) to compute the output $\hat{\boldsymbol{y}}^{(t)}$ at each time step on the forward pass, can specified by two equations (Lipton & Berkowitz, 2015)

$$\boldsymbol{h}^{(t)} = \sigma(W^{\mathrm{hx}}\boldsymbol{x}^{(t)} + W^{\mathrm{hh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h) \tag{1}$$

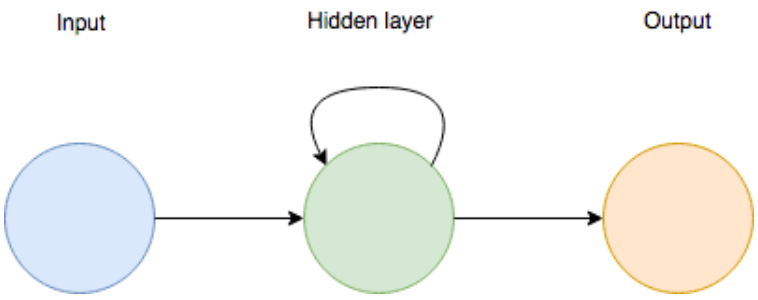


Figure 3: Simple recurrent network with one input unit, one output unit and one hidden layer with a recurrent edge.

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(W^{\text{yh}}\boldsymbol{h}^{(t)} + \boldsymbol{b}_y) \tag{2}$$

where $\boldsymbol{x}^{(t)}$, $\boldsymbol{h}^{(t)}$ and $\hat{\boldsymbol{y}}^{(t)}$ is the input, hidden state and output respectively. W_{hx} is the matrix of conventional weights between the input and the hidden layer and W_{hh} is the matrix of recurrent weights between the hidden layer and itself. Bias parameters which allow each node to learn an offset is represented by the vectors $\boldsymbol{b}_h$ and $\boldsymbol{b}_y$.

The recurrent behaviour across time steps can be visualized by *unfolding* it as in figure 4. Then the network can be interpreted as a deep network with one layer per time step and shared weights across time steps. It is clear that *backpropagation through time* (BPTT) can now be applied to the unfolded network and train it across many time steps. This is the de facto standard how common RNNs apply it (Lipton & Berkowitz, 2015).

3.3.1 Exploding and vanishing gradients

While figure 3 and 4 describe a simple and in principle a very powerful model, there are some common issues with RNNs which makes them hard to train properly. The *vanishing gradient* and *exploding gradient* problems are among the main reasons why this model tend to be so unwieldy, and occurs when errors are backpropagated across many time steps (Pascanu et al., 2013). The *vanishing gradient* problem refers to long term components moving exponentially fast to norm zero, making it hard for the model to update weights significantly enough for further training. Contrary, the *exploding gradient* problem refers to the large increase in the norm of the gradient during training. Large updates from gradients to weights can cause an unstable network, and in extreme cases lead to overflowing weights and

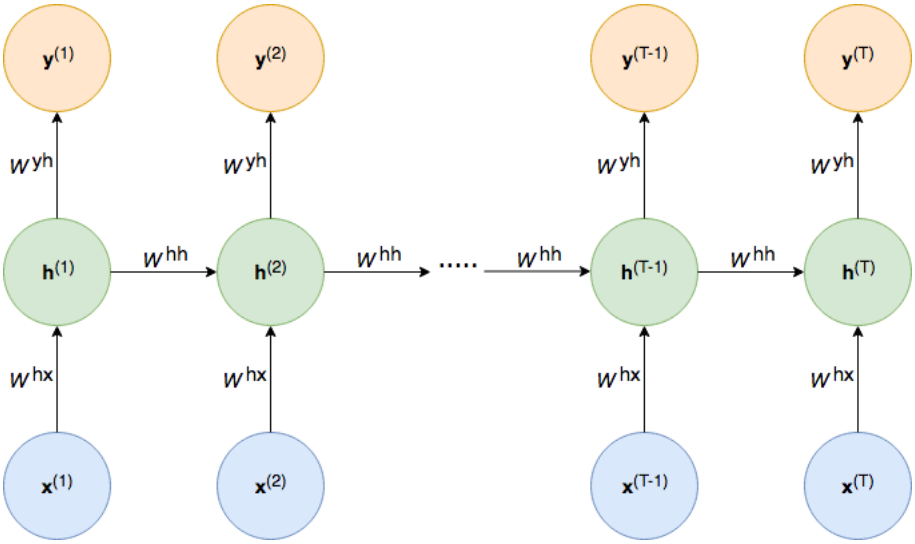


Figure 4: Simple recurrent network unfolded across multiple time steps. Note that the hidden state is passed from one hidden layer to the next one, this is the recurrent step.

result in NaN values.

3.3.2 Long short-term memory

Long short-term memory (LSTM) is a special architecture of RNN, and unlike other feedforward networks, LSTM has feedback connections enabling it to more efficiently memorize longer sequences and it remembers values over an arbitrary time interval. LSTM's are well suited for tasks like classifying, processing and making predictions based on time series.

LSTM was introduced in 1997 by Hochreiter and Schmidhuber as an attempt to solve the exploding and vanishing gradient problem. RNNs have been proven not to be as efficiently trainable by gradient descent when long-term context is required (Bengio, Simard, & Frasconi, 1994). Error signals from backprogation through time tends to either blow up or vanish, the evolution of the temporal backpropagated error exponentially depends on the size of the weights. In theory, researchers could adjust and carefully pick parameters for the network to better learn long-term dependencies, but in practice RNNs seem unable to learn them properly (Hochreiter & Schmidhuber, 1997).

LSTM share a similare structure with the traditional RNN, but each ordi-

nary node in the hidden layer is replaced by a *memory cell*. It consists of three different gates; *input gate*, *forget gate*, and *output gate*. Additionally the memory cell has an *input node* and an *internal state*, all of these elements are illustrated in figure 5 and described below (Lipton & Berkowitz, 2015).

(a) **Input node**: Takes input from layer $\boldsymbol{x}^{(t)}$ at the current time step and from the hidden layer at previous time step $\boldsymbol{h}^{(t-1)}$. An activation function, typically *tanh*, is applied on the summed weighted input.

(b) **Input gate**: Sigmoidal unit which also takes activation from the current data point $\boldsymbol{x}^{(t)}$ as well as from the hidden layer at the previous time step. It is a gate in the sense that if the value is zero, none of the values from other nodes are flowing through. If the value is *one* on the other hand, the values are passed through.

(c) **Internal state**: Heart of the memory cell, with self-connected recurrent edge of fixed unit weight and linear activation. This edge with constant weight allows error to flow across time steps without vanishing nor exploding, and is therefore commonly referred to as the *constant error carousel*.

(d) **Forget gate**: Not part of the original LSTM design described in Hochreiter et al (Hochreiter & Schmidhuber, 1997), but introduced in 2000 by Gers et al (Gers, Schmidhuber, & Cummins, 2000). These gates give the network a possibility to flush the contents of the internal state, and thus *forget* previous learned weights.

(e) **Output gate**: Multiplying internal state with the output gate yields the ultimate produced value by a memory cell. Internal state is often first run through a *tanh* activation function, this gives the output of each cell the same dynamic range as an ordinary *tanh* hidden unit.

The following equations describe formally how the LSTM model works. All calculations are done at each time step.

$$\boldsymbol{g}^{(t)} = \phi(W^{\mathrm{gx}}\boldsymbol{x}^{(t)} + W^{\mathrm{gh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_g$$

$$\boldsymbol{i}^{(t)} = \sigma(W^{\mathrm{ix}}\boldsymbol{x}^{(t)} + W^{\mathrm{ih}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i$$

$$\boldsymbol{f}^{(t)} = \sigma(W^{\mathrm{fx}}\boldsymbol{x}^{(t)} + W^{\mathrm{fh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f$$

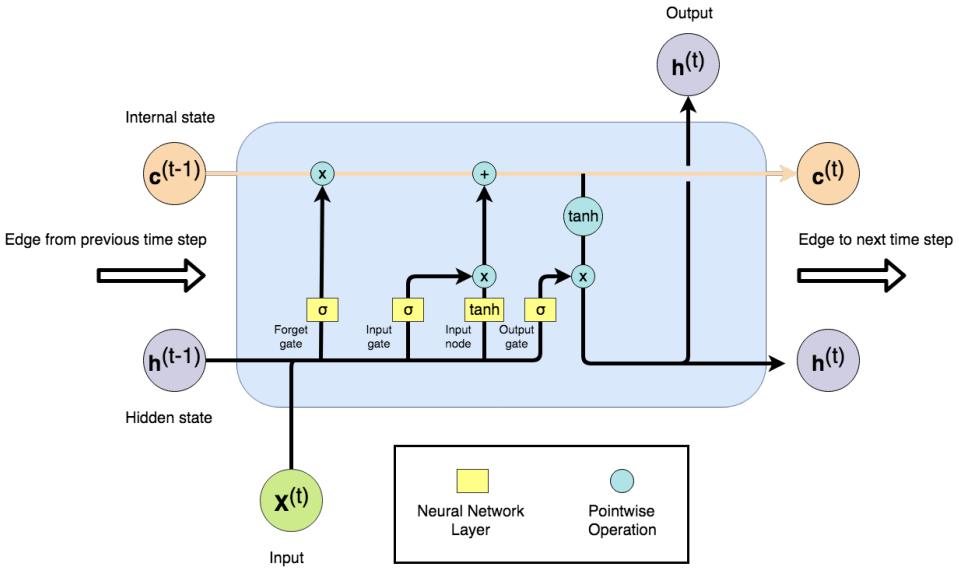


Figure 5: Architectural overview of the LSTM cell.

$$\boldsymbol{o}^{(t)} = \sigma(W^{\text{ox}}\boldsymbol{x}^{(t)} + W^{\text{oh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o$$

$$\boldsymbol{s}^{(t)} = \boldsymbol{g}^{(t)} \odot \boldsymbol{i}^{(t)} + \boldsymbol{s}^{(t-1)} \odot \boldsymbol{f}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \phi(\boldsymbol{s}^{(t)}) \odot \boldsymbol{o}^{(t)}$$

where $\boldsymbol{g}$, $\boldsymbol{i}$, $\boldsymbol{f}$, $\boldsymbol{o}$, $\boldsymbol{s}$, and $\boldsymbol{h}$ is, in the following order, input gate, input node, forget gate, output gate, internal state, and hidden state.

3.4 Kernel Density Estimation

Kernel density estimation (KDE) is a nonparametric estimator that will learn the shape of the density from the data itself. Sometimes referred to as the Parzen's window, the KDE is a widely used approach to estimate the underlying probability density function of a dataset. KDE solves the fundamental data smoothing problem from a set of random variables where the smoothness is depending on a given bandwidth h, as seen in figure 6. Any assumption that the underlying density function is from a parametric family is not required, which makes KDE a very popular approach for complicated data distributions (Chen, 2017).

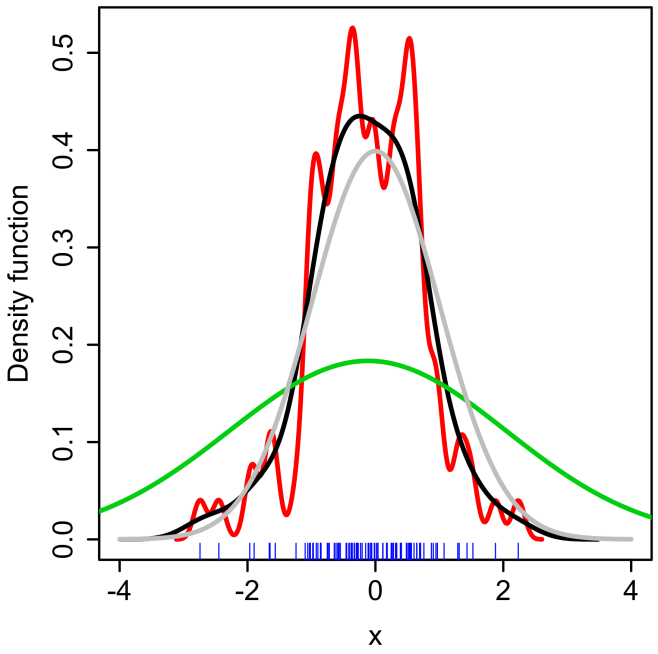


Figure 6: KDE with different bandwidths; grey is the true density, red, black and green has h equal to 0.05, 0.337 and 2 respectively. The sample is 100 random points from a standard normal distribution (from en.wikipedia.org/wiki/Kernel_density_estimation).

Method

Chen states that if we let $X_1, ..., X_n \in \mathbb{R}^d$ be an independent, identically distributed random sample from an unknown distribution P with density function p, KDE can be expressed as

$$\widehat{p}_n(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{x - X_i}{h}), \tag{3}$$

where K is the kernel (smoothing) function and h is the bandwidth which controls the amount of smoothing. Different kernel functions would yield different estimates. Below is two common examples of $K(x)$ outlined

$$\text{(Gaussian kernel) } K(x) = \frac{exp(-||x||^2/2)}{v_{1,d}},\ v_{1,d} = \int exp(-||x||^2/2)dx,$$

$$\text{(Spherical kernel) } K(x) = \frac{I(||x|| \leq 1)}{v_{2,d}},\ v_{2,d} = \int I(||x|| \leq 1)dx.$$

3.5 Naive Bayes classifier

The Naive Bayes classifier is a probabilistic classifier, known for its simplicity and ease of use. Other state of the art classifiers, such as Support Vector Machine, k-nearest neighbor, and Linear Least Squares, significantly outperforms Naive Bayes (Yang & Liu, 1999). Still, Naive Bayes is frequently used for text classification tasks and it often serves as a baseline method because speed and ease of implementation compared to other algorithms, which tend to be slower and more complex.

Naive Bayes assume features are independent given a class and they contributes evidence individually even when features depend on each other (Teevan, 2003). This can be expressed with the following equation

$$P(\mathbf{X}|C) = \prod_{i=1}^{n} P(X_i|C) \tag{4}$$

where $\mathbf{X} = (X_1, ..., X_n)$ is a feature vector and C is a class. Despite the fact that independence is an unrealistic assumption, Naive Bayes has proven to be competitive in many practical applications also beyond text classification, including medical diagnosis and system performance management (Rish, 2001).

From Bayes' theorem we have this equation

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{5}$$

By combining equation 4 and 5 we have the approach used by the Naive Bayes

$$P(C|\mathbf{X}) = \frac{P(C)\prod_{i=1}^{n} P(X_i|C)}{P(\mathbf{X})} \tag{6}$$

where $P(\mathbf{X})$ is identical for all classes, and therefor can be ignored. This yields the *Naive Bayes* classifier

$$P(C|\mathbf{X}) \propto P(C)\prod_{i=1}^{n} P(X_i|C) \tag{7}$$

Multinomial Naive Bayes

A Naive Bayes classifier variant using multinomially distributed data, often encountered in text classification, is called multinomial Naive Bayes. A Multinomial distribution models the probability for n independent trials which have k different categories, opposed to binomial where $k = 2$.

Equation 8 shows the multinomial Naive Bayes model (Su, Sayyad Shirab, & Matwin, 2011)

$$P(C|\mathbf{X}) = \frac{P(C)\prod_{i=1}^{n} P(X_i|C)^{f_i}}{P(\mathbf{X})} \tag{8}$$

where f_i is the number of occurrences of a feature X_i in a feature vector $\mathbf{X}$.

A common problem with both the traditional and multinomial Naive Bayes occurs when a feature has not been associated with a given class in the training phase, which leads to a zero probability for $P(X_i|C)$. This scenario will wipe out all information from this feature vector and return a zero probability, although the rest of the features being strongly classified in one class. Laplace smoothing is a popular way to estimate $P(X_i|C)$ and eliminates the zero probability by adding a small fraction to all probabilities (Dai, Xue, Yang, & Yu, 2007). The Laplace smoothing is given by

$$P(X_i|C) = \frac{1 + n(X_i, C)}{|F| + n(C)} \tag{9}$$

where $n(X_i, C)$ is the number of occurrences of the feature X_i in our training set whose class value is C. $n(C)$ is the total number of training examples whose class value is C, and finally $|F|$ represents the total number of distinct features.

3.6 Kullback-Leibler Divergence

Solomon Kullback and Richard A. Leibler introduced the concept of the Kullback-Leibler (KL) divergence in 1951 (Kullback & Leibler, 1951). KL measures the similarity (or dissimilarity) between two different probability distributions by calculating the cross-entropy minus the entropy given by

$$D_{KL}(P||Q) = H(P,Q) - H(P) \tag{10}$$

which can be expressed in a discrete summation form for distributions P, Q on a finite set χ as follows (Bigi, 2003)

$$D_{KL}(P||Q) = \sum_{x \in \chi} P(x) log \frac{P(x)}{Q(x)} \tag{11}$$

The KL divergence is a non-symmetric measure of distance from P to Q, and therefore does not satisfy the triangle inequality (Lifang, Sijun, & Huan, 2017)

$$D_{KL}(P||Q) \neq D_{KL}(Q||P)$$

KL divergence value is always greater than or equal to zero. If and only if the two probability distributions are exactly the same, the value of KL divergence is zero

$$D_{KL}(P||Q) \geq 0$$

Similar distribution will have smaller relative entropy.

Since KL is an asymmetric measure it can not be denoted as a strictly distance metric, although Bigi describes a symmetric KL divergence variant i.e. the Kullback-Leibler Distance (KLD) metric as

$$D_{KLD}(P||Q) = \sum_{x \in \chi} \left((P(x) - Q(x)) log \frac{P(x)}{Q(x)} \right) \tag{12}$$

3.7 Multidimensional binary search tree

A multidimensional binary search tree, from now on known as a *k-d tree* (k stands for the dimensionality of the search space), is a data structure optimized for the storage of k-dimensional data. Each record in a file is represented by a node, with pointers to other nodes. If both pointers is *null*, the associated node is considered a leaf node of the tree. All nodes also have a discriminator key, these keys are in the range of 0 to $k-1$, inclusive. The discriminator key decides which dimension to split, in a two dimensional space this would alternate between the x and y plane. The root node always has the discriminator 0 and the first two children have 1 as discriminator. This key value continues to increase for each level down the tree, until the kth level whom has discriminator $k-1$. The $(k+1)$th level has discriminator 0 and the cycle repeats. As a rule, the next discriminator denoted as *NEXTDISC* is a function defined as (Bentley, 1975)

$$NEXTDISC(i) = (i+1) \bmod k$$

where i is the level. Figure 7 gives an example of records in 2-d space stored as nodes in a 2-d tree with discrimination key. All nodes on a given level in the tree has the same discriminator.

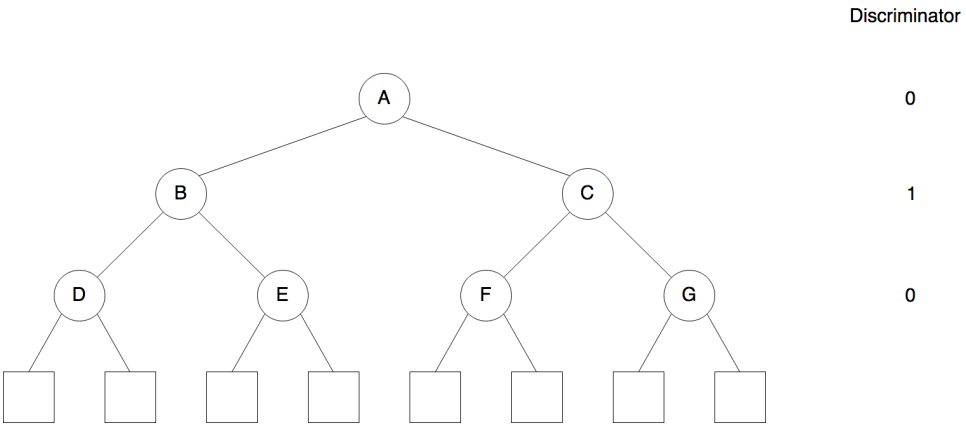


Figure 7: Balanced kd-tree with discriminator keys for each level. Internal nodes are marked as circles and leaf nodes as squares.

The leaf nodes, marked as squares in figure 7, are often called *buckets*. They have a threshold that limits the maximum number of data points allowed in each bucket. This threshold is known as the *bucket size* of the tree. Data points are partitioned to different buckets based on the split value of the internal nodes. Each internal node (a node with pointers) has a split value

where lesser values goes to the left branch and higher or equal values to the right (Egas, Huijsmans, Lew, & Sebe, 1999).

There exists several different methods for choosing the appropriate split value. Two of the most common approaches are the *Friedman* and *Midpoint* method. The first method splits at the median of all the points, creating two equaled sized buckets on both sides. This results in a perfectly balanced kd-tree, which usually leads to a more compact tree and less sparsity. The latter splits on the midpoint between the furthest points, opening up for greater differences in number of points in each bucket. This would likely draw more geographically desirable boundaries than the Friedman method (Roller, Speriosu, Rallapalli, Wing, & Baldridge, 2012).

3.8 Evaluation metrics

Depending on the representations of ground truth and prediction different evaluation metrics are used.

Distance-Based evaluation metrics

Distance-based metrics are used for locations represented by their geographical coordinates. For a tweet t, and a location L, a system is aiming at predicting a location $L(t)$. The predicted location $L(t)$ is expected to be close to the ground truth $L^{'}(t)$. The *Error Distance* (ED) between prediction and ground truth is defined by the *Euclidean* distance between the coordinates. For a single tweet the ED is given by

$$ED(t) = dist(L(t), L^{'}(t))$$

Evaluations are calculated on a collection of tweets, let T be the set of all tweets. For metrics on collection level, we define *Mean Error Distance* ($\overline{ED}$) and *Median Error Distance* ($\widetilde{ED}$). These are given by

$$\overline{ED} = \frac{1}{|T|} \sum_{t \epsilon T} dist(L(t), L^{'}(t))$$

$$\widetilde{ED} = \underset{t \epsilon T}{Median}\ dist(L(t), L^{'}(t))$$

Another widely adopted distance based metric for collection level evaluation is *Distance-based Accuracy* (*Acc@d*). For a predefined distance threshold d, any prediction with error distance not exceeding d is considered a correct prediction. *Acc@d* is given by

$$Acc@d = \frac{|t \epsilon T : ED(t) \leq d|}{|T|}$$

4 Related work

Predicting demographic information of Twitter users is a widely studied research area, which includes studies of inferring users' age, gender, personalities and profession (Xin Chen, 2015).

Another category of inference studies there has been an increasing interest of is the prediction of location of both Twitter users and individual tweets, primarily driven by the lack of sufficient geotagged data. Only 1-3 % of tweets globally are tagged with geographical information such as latitude and longitude (Li et al., 2014). The problem of location prediction associated with objects have been widely studied in several different contexts including Wikipedia articles (Benjamin P. Wing, 2011), web pages (Amitay, Har'El, Sivan, & Soffer, 2004) and general text documents (Allison Gyle Woodruff, 1994). A significant difference of location prediction on Twitter compared to other objects is the new challenges in the noisy and short nature of tweets, but also opportunities in terms of the rich context a tweet contains. Location prediction on Twitter can be divided into three main categories of Twitter related locations, namely *User home location*, *Mentioned location*, *Tweet location* (Zheng, Han, & Sun, 2018). User home location refers to the task of predicting a users' long-term residential address, often divided into different types of location granularities. These location granularities includes administrative regions like countries, states or citites, geographical grid cells and geographical coordinates representing the finest granularity. In tweet contents, users may mention the names of locations. The task of identifying and predicting the locations of those mentions is referred to as Mentioned location prediction. This task may provide a better understanding of tweet contents and benefit applications like location recommendation and disaster detection. The final category of Twitter related locations is tweet location, and will be the focus of this project. This is the task of estimating the geographical origin of where a tweet is posted from. By inferring tweet location we may better understand a user's mobility, and further exploit the tweets content in applications for local event detection (Paul S. Earle, 2011), location-based recommendation and targeted advertisement.

As this has been a widely studied and explored research area in recent years, several approaches and techniques have been examined. Typical differences in previous studies is the focus and representations of geographical areas, and varying information exploited for inferring tweet locations. The majority of previous works focus on limited geographical areas like a country (Berggren, Karlgren, Ostling, & Parkvall, 2016) or city (Özdikis, Ramampiaro, & Nørvåg, 2018b), while only a small amount of previous research focus on location prediction world wide (Huang & Carley, 2017). There are

multiple types of location representation methods used in previous studies. One common method for fine grained location prediction is to divide the geographical area of interest into small grid cells, and use document classification methods to predict which cell a tweet belongs to (Benjamin P. Wing, 2011). A limitation of this approach is that grid cells in rural areas tends to contain very few tweets compared to grid cells in urban areas. To accommodate this limitation Roller et al (Roller et al., 2012) proposed an extension to this method by using an adaptive grid representation where grid cells contain approximately an equal amount of data.

Inference of tweet location can be based on different types of information. A common way suited for real time processing is to base the prediction on content from a single tweet (Özdikis et al., 2018b), e.g, word-centric and location-centric methods. Due to the short and noisy nature of tweets there exists methods to enrich the available information. This includes methods for inference based on twitter network, tweet history or tweet context. Examples of using such enriched information is to use information like a certain number of tweets from a users' timeline (Pengfei Li & Pan, 2018), or the use of social relationships (Jurgens, 2013). In terms of tweet context there has been proposed methods for exploiting posting times or time zones for tweet location prediction (Dredze, Osborne, & Kambadur, 2016).

Various techniques from several areas have been proposed to make accurate predictions, including the areas of information retrieval, machine learning and natural language processing (Zheng et al., 2018). Multinomial Naive Bayes (MNB) and Kullback-Leibler (KL) divergence with varying enhancements are among the most widely applied methods. Ozdikis et al proposed a location prediction method using the grid-based approach for tweets based on the geographical probability distribution of their terms over a region. In this method the probabilities are calculated using Kernel Density Estimation (KDE), with term-specific bandwidth selection (Özdikis, Ramampiaro, & Nørvåg, 2018a). Another grid-based method proposed by Ozdikis et al used term co-occurrences in tweets that exhibit a spatial clustering or dispersion tendency with significant deviation from the underlying single-term patterns. Ripley's K-function is used to analyze the geographical distribution of terms and term pairs, where terms yielding a significant clustering or dispersion tendency are used to extend the feature space in probabilistic language models (Özdikis et al., 2018b).

In addition to the widely proposed range of statistical methods, neural networks is another technique that has gained recent interest to issue the problem of location prediction of tweets (Zheng et al., 2018). The use of neural networks on geolocation problems seems to be a relatively new approach, and results states that this technique deserves further research. Examples of pre-

vious studies exploiting neural networks includes the work of Iso et al (Iso, Wakamiya, & Aramaki, 2017), which use a convolutional mixture density model fed by tweet content, to estimate parameters of the mixture model. The mode value of estimated density is treated as the predicted coordinates for tweets. Other works applying neural network models are the works of Miura et al (Miura, Taniguch, Taniguchi, & Ohkuma, 2017), which uses a complex network that unifies text, metadata and user network representations, and Rahimi et al (Rahimi, Cohn, & Baldwin, 2017) which proposes a simple text-based geolocation model with one hidden layer. Huang et al propose a method of using convolutional neural networks to predict the geolocation based on information in a single tweet (Huang & Carley, 2017). Another method of combining tweet text with metadata (user description, user location, user name, timezone) for training a Long Short-Term Memory based classifier is proposed by (Thomas & Hennig, 2017).

To best of our knowledge the proposed method differs from the latter deep learning studies by exploiting the grid based approach for modelling geographical space, where we examine both the uniform and adaptive distribution. Due to the problem of reproducing AI research (Gundersen & Kjensmo, 2018), the results will mainly be compared to state of the art statistical methods for the geolocation problem.

5 Location prediction using neural networks

In a grid-based approach the geographical region of interest is discretized into smaller grid cells. Our approach for geolocation assigns the grid cell most probable of containing a given tweet. In this section we describe the details of our proposed location prediction method and give a brief overview of baseline methods used for comparison. The proposed method is tested on several different datasets using two approaches for geographical modelling, namely uniform and adaptive grid, both described in this section.

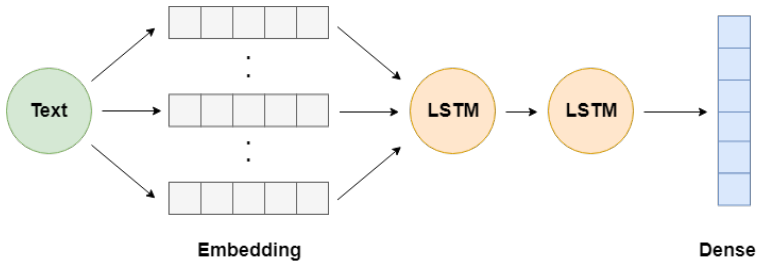
5.1 Approach

To explore the use of deep learning to issue the geolocation problem described in section 2.2, our proposed method is based on neural networks where we train a model to predict the grid cell most probable of containing a given tweet. As mentioned in section 2.2, Twitter data contains a lot of interesting contextual information that may be utilized. Therefore we propose models based solely on tweet text, but also models using additional contextual information. The proposed network architectures benefits mainly from the power of Recurrent Neural Networks 3.3, where we use two different architectures.

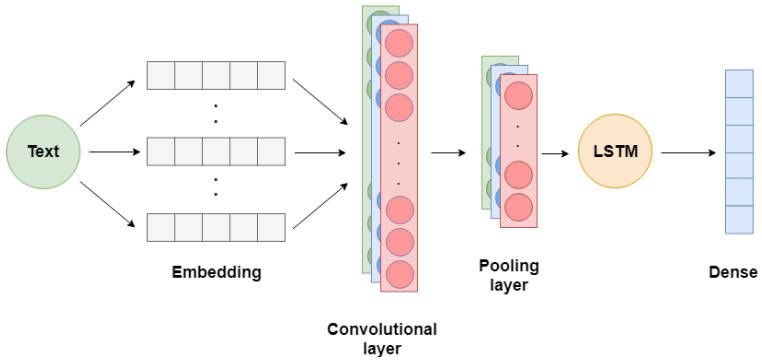
- The first one is based on RNN and utilizes the sequence processing ability of LSTMs (3.3.2), and is referred to as LSTM in the result section.
- The second one is based on a combination of recurrent and convolutional (3.2) neural networks, and is referred to as LSCN in the result section. The idea is that the CNN calculates a higher-level representation of the data, and has shown good results for text classification (Zhou, Sun, Liu, & Lau, 2015).

5.2 Text-based approach

The most general and simple model to infer geolocation is based on tweet text only. This is due to the availability of contextual metadata being different from data source to data source, e.g, the metadata available on Twitter is different from Facebook or Newspaper articles. The development of a model exploiting tweet text only will enable the approach to be tested on other domains. The tweet text is tokenized and converted into word embeddings before being forwarded to LSTM / Convolutional layers. The proposed network architectures for the text-based approach are presented in figure 8.



(a) LSTM

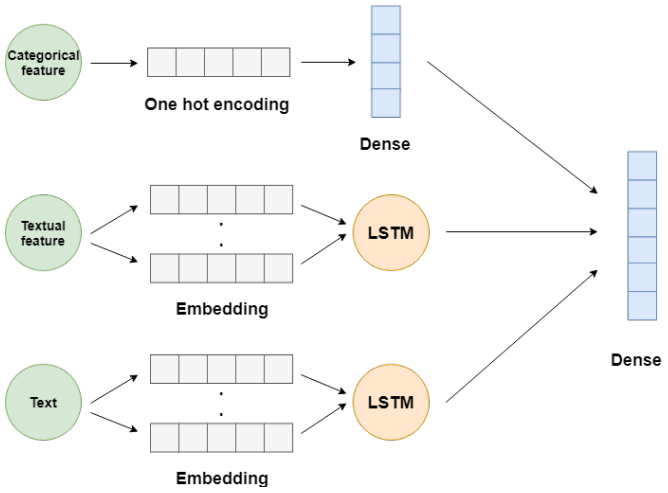


(b) LSCN

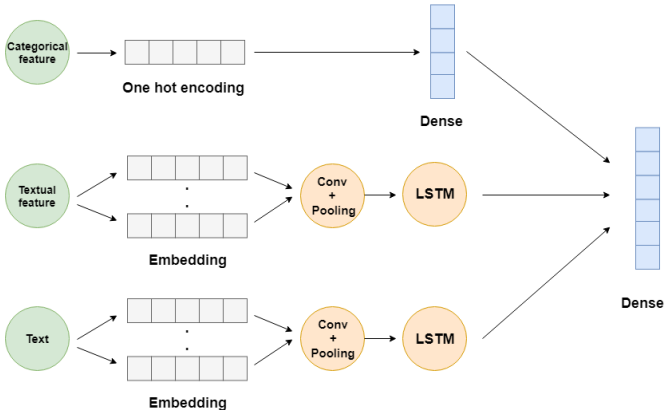
Figure 8: Network architectures exploiting tweet text only

5.3 Text and metadata approach

To investigate the impact of contextual information in terms of more accurate location prediction, we develop a model exploiting Twitter specific metadata. This includes the textual features of *text*, *username*, *user description* and the categorical features of *user language* and *posting time*. Individual models are created for each feature. These individual models are merged after initial encoding and forwarded to the output layer. The textual features are tokenized, converted into word embeddings, and forwarded to LSTM / Convolutional layers, whereas the categorical features are one hot encoded and forwarded to Dense (Fully connected) layers. The proposed network architectures for the metadata approach are presented in figure 9.



(a) LSTM



(b) LSCN

Figure 9: Network architectures exploiting text and metadata

5.4 Baseline methods

As the baseline for comparison we used the state of the art method of Locality-adapted Kernel Densities (Özdikis et al., 2018a). The results presented in the LocKDE paper states that the method significantly outperforms previous work, including methods based on Multinomial Naive Bayes and Kullback-Leibler. Due to these strong results LocKDE is chosen as the baseline method.

The LocKDE method is based on Kernel density estimation 3.4, and is a method for the geolocation problem based on the geographical probability distribution of tweet terms over a region. For this particular method the locality-adapted bandwidths is based on the information gain ratio (IGR), which is an information theoretic metric used to obtain location indicative terms. After estimating the probability density functions for each term, these functions are used to assign probability masses to grid cells. This is done by calculating the probability of observing term t in a specific grid cell. After training, prediction for a new tweet is performed by selecting the grid cell maximising the cumulative probability based on the the probability distributions of the tweet terms.

In our case this method was implemented by using the github repository tweet-localization-LocKDE provided by Özer Özdikis.

5.5 Geographical modelling

Except from exact coordinates, city level is the finest granularity we can obtain directly from the twitter dataset. For large cities with millions of tweets it might be desirable to partition it even further, into sub-areas. One way to accomplish such results is to create a grid-based system overlaying the city, defining nearby tweets as one logical area.

The most trivial solution to this problem would be to construct a grid with equal sized grid-cells within a defined boundary. This approach is called a *uniform grid* and it is a widely used method due to its simplicity. An obvious limitation is the number of data points in each cell. Grid-cells are purely created based on their location and *not* the density of the data points. This results in sparse and overly dense regions, and it is complicated to ensure roughly equal number of points in each cell. *Adaptive grids* tries to solve this problem by dividing a geographical area into grid-cells based on the density of data. Sparse regions get larger cells, and similarly dense regions get divided into smaller cells. Below are both methods outlined in further detail and their implementation explained.

5.5.1 Uniform grid

A uniform grid consists of a $m \times n$ matrix with columns of equal width and rows of equal height. Thus, all grid-cells are equal in size and defined inside a fixed boundary. Figure 10 shows an example of a 5×5 grid over an area with data points.

To create a uniform grid overlaying a city, we have to define the boundaries. Coordinates bounds are collected through the Nominatim Api. A query specifying the city is issued and a list of results are returned in relevant order, the most likely result as the first item. A query for *Manhattan NY* might look like this

$$q{=}manhattan{,}new{+}york{,}us$$

where *manhattan* is the place name, *new+york* is the name of the state, and *us* is the country. The above query would return a result similar to this in listing 2 (the first element is chosen).

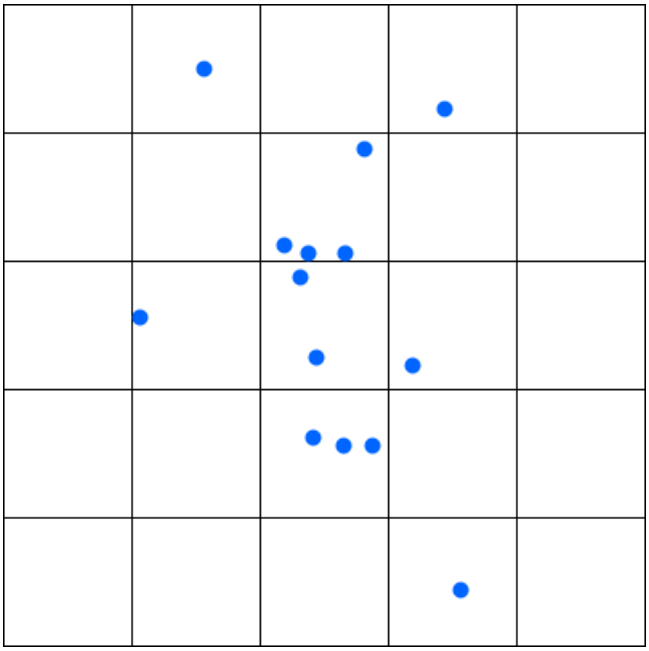


Figure 10: Uniform grid with 5×5 number of cells.

Listing 2: Result of a Nominatim Api search for Manhattan NY


```
{
    "place_id":199324647,
    "licence":"Data OpenStreetMap contributors, ...",
    "osm_type":"relation",
    "osm_id":8398124,
    "boundingbox":[
        "40.6996823",
        "40.8777963",
        "-74.0194416",
        "-73.9101872"
    ],
    "lat":"40.7900869",
    "lon":"-73.9598295",
    "display_name":"Manhattan, New York County, ...",
    "class":"boundary",
    "type":"administrative",
    "importance":1.09014193609354,
    "icon":"https://nominatim.openstreetmap.org/images/..."
}
```


The most interesting part here is the *boundingbox* field. From this the north-east and south-west bound is created, boxing in the region of interest. Number of rows and columns are determined from a density constraint,

given in kilometres. To convert degrees into kilometres a very simple conversion is used. At equator each latitude is 110.567 kilometers apart and at each poles the distance is 111.699 kilometres. This gives us an approximation of 111 kilometres between latitudes, regardless of where the tweet is located. However, longitudes starts out with 111.321 kilometres between each degree at equator, but this gradually shrinks to zero as they meet at the poles. Following function outputs the length of the longitude distance given a latitude

$$long_{km} = L \times cos(\phi) \times d$$

where L is the distance constant (111 kilometres), ϕ is the latitude in radians, and d is the distance in degrees.

The actual grid-cells may differ some in length from the given density. Since the boundary is already fixed, the cell length has to be rounded off to fit a whole number of cells inside the grid.

5.5.2 Adaptive grid

The adaptive is adjusting each cell depending on the density and thus tries to overcome the shortcomings of the uniform grid. An alternative way of splitting the points in figure 10 is shown in figure 11. There exists several methods to partition data points in an adaptive manner, one of them is called k-d tree. When partitioning geolocation data, we consider the surface of the earth to be a 2-dimensional space (k=2), consisting of latitude and longitude pairs. By partitioning geolocated data points with k-d tree, dense regions gets finer granularity opposed to sparse regions which have coarser granularity.

All points starts out in the root node and gets split into two nodes if the total number exceeds the bucket size. The same procedure happens over and over with all leaf nodes until all buckets have points below a given threshold. When splitting an overflowing node, the midpoint method is used, resulting in an unbalanced tree. Additionally there is a tweets limit parameter, effectively filtering away nodes with a very low count of data points.

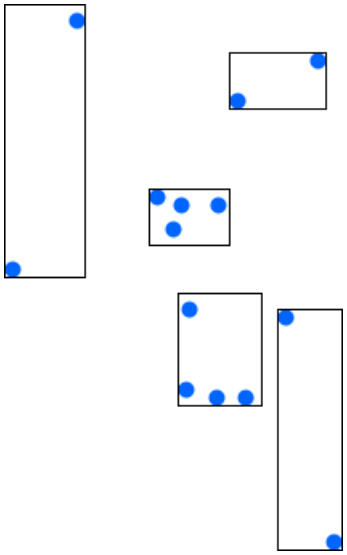


Figure 11: Adaptive grid approach for partitioning data points based on the density.

5.5.3 Partition location method

For each grid cell a point of center has to be chosen. The most trivial solution would be the geographically center, determined by the height and width of the rectangle. Although this approach is just fine for smaller cells, it ignores the fact that a great number of cells have an imbalance in the dispersion of data points. Larger cells may have data points grouped together in one corner and the geographical center does not represent these points sufficiently. As an alternative to the traditional center, we select the *centroid* of the locations of all the data points located in that cell. The *centorid* represents the arithmetic mean position of all points present, and effectively moves the center to the 'center of mass'. As mentioned, this will affect larger cells in a greater extent and small cells will have an insignificant difference in absolute distance between a center or centroid prediction.

It is especially important for the k-d tree to use the centroid method, regions with very low density results in cells spanning over large areas. With centroid it is possible for these large leaves to still be in the mix, predicting locations within them that have the greatest data density.

6 Evaluation

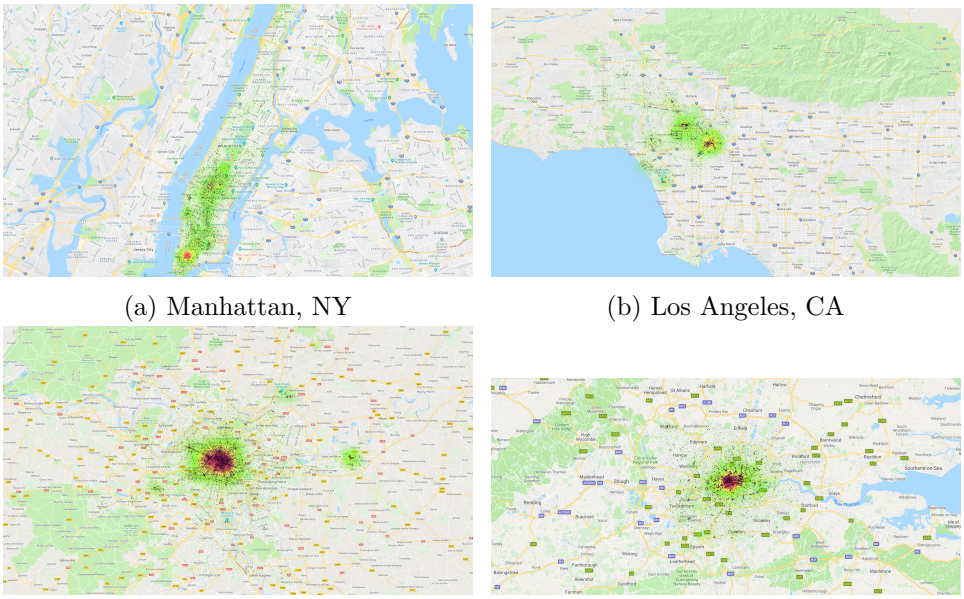
For the evaluation of this project we introduce four different datasets from the geographical regions of Manhattan, Los Angeles, Paris and London. The three former is used for the baseline comparison, and the latter is used for the metadata comparison. The experiment is presented by describing the structure and preprocessing steps of the training data, and how to feed the data to the neural classifier. This includes an outline of the results section and how the results are presented regarding baseline and metadata comparisons, and the evaluation metrics used. To present the accurate comparison of the uniform and adaptive grid approach it is crucial to find the best settings for both approaches respectively. For the uniform grid this involves finding the optimal size of grid cells in terms of prediction accuracy for each dataset, and the optimal bucket size on the same terms for the adaptive grid. These grid settings are developed in a tuning section before the results are presented using the prepared grid values. After presentation, the results are discussed on the basis of the research goals of this project.

6.1 Dataset

In order to train neural network models and compare to results published by other researchers, several different datasets have been acquired. All sets consists of geotagged tweets from different geographical regions. In addition to *latitude* and *longitude* values, it is crucial to have the tweet text available. All other types of metadata provided by twitter is optional, and there is some differences across the datasets regarding how much information they contain beyond tweet text and coordinates. The datasets are collected using the Twitter Streaming API, following common practice for tweet filtering. This includes the exclusion of duplicate tweets, tweets from users with more than 1000 friends or followers, and tweets from users posting more than two times daily.

6.1.1 Manhattan

The **Manhattan** dataset consists of 51 529 tweets from 19 124 unique users. 46 432 tweets are tagged with 'en', denoting tweets written in English. Distribution of tweets are seen in figure 12a.



(a) Manhattan, NY

(b) Los Angeles, CA

(c) Paris

(d) London

Figure 12: Heatmap of tweet locations in four different cities, where green is low density of tweets and red is high.

6.1.2 Los Angeles

The **Los Angeles** dataset is the smallest dataset with 39 870 tweets, including 14 667 unique users and 35 884 English tweets. Distribution of tweets are seen in figure 12b.

6.1.3 Paris

The **Paris** dataset is roughly three times bigger than **Manhattan**, with 152 123 tweets. Limited metadata available, where only *latitude*, *longitude*, and *tweet_text* are included within each tweet.

6.1.4 London

The **London** dataset consists of 69852 tweets and is collected between February and April 2019. In addition to *latitude*, *longitude*, and *tweet_text*, all features listed in table 1 is present in this dataset.

6.2 Experiment

The geographical area of interest is divided into uniform or adaptive grid cells as described in section 5.5, and shown by example in figure 15a. Each grid cell is assigned an *id* (GCID), stored together with bottom left and top right corner in a grid file. A chunk of a grid file is shown in table 2.

Table 2: Example of content in grid file

GCID	Lat-min	Lon-min	Lat-max	Lon-max
0	40.6996823	-74.0194416	40.719472745	-73.992128
1	40.719472745	-74.0194416	40.739263189	-73.992128
2	40.739263189	-74.0194416	40.75905363	-73.992128
3	40.75905363	-74.0194416	40.778844078	-73.992128

Each tweet is assigned to a grid cell based on the associated GPS coordinates resulting in a tweets file containing tweet data and assigned cell id.

Table 3: Example of content in tweets file with dummy text.

GCID	Latitude	Longitude	Tweet text
0	40.7142	-74.0064	Aliquam eleifend est quis mauris.
1	40.7203431	-74.01383173	Quisque tempor consequat risus.
2	40.7395058	-74.008293	Donec eu posuere libero.

The text is tokenized using the *TweetTokenizer* from *nltk*. To reduce data sparsity we also exclude single characters, tokens that appear in less than 5 tweets, and twitter shortened hyperlinks as they carry no meaningful contextual information. The datasets are split into training and validations sets using a randomly selected share of 95% for the training set, and 5% for the validation set. No restriction in tweet language is applied.

To feed the data through a feed forward neural network we convert the textual data into sequences using the *Tokenizer* from the Keras library, and apply zero padding to ensure equal length input vectors. In the training phase of our neural network these sequences are passed as training data, together with one hot encoded vectors of target data built from the associated grid cell ids. A softmax distribution is used to select the grid cell with

highest probability from a series of n different outcomes. The training phase results in a neural model trained to predict probable grid cells for a given tweet. In the prediction phase the validation data is preprocessed equally as in the training phase, and is fed to the pretrained neural model. Based on the output we select the grid cell with highest probability of containing a given tweet, resulting in a predicted grid cell. To derive a set of coordinates from the predicted grid cell we calculate the *centroid* as described in section 5.5.3. This *centroid* represents the predicted tweet location and is used as basis to calculate the error distance, representing the distance between predicted and true tweet location. In the results section we present results for both the LSTM and LSCN model described in section 5.1.

For comparison we implemented the baseline method of *LocKDE* as described in section 5.4. This baseline is selected due to its ability to yield strong results and high accuracy compared to other widely-used techniques in the literature like *MNB* and *KL-divergence*. Based on the results obtained in (Özdikis et al., 2018a), we implement the *LocKDE* under the locality adapted bandwidth enhancement, and the enhanced weighing of probabilities based on IGR.

In additon to the baseline comparison utilizing tweet text only, we present a section comparing models built using tweet text only against models utilizing text and metadata. In terms of twitter, various metadata is available using the Twitter streaming API as described in section 2.2. The London dataset contains such metadata information where we exploit the contextual features of *user language*, *posting time*, *username* and *user description*. The textual features of *username* and *user description* are tokenized equally as *tweet text*. The categorical features of *user language* and *posting time* are one hot encoded, where the former is given as ISO 639-1 codes (two-letter country codes), and the latter is converted from a UTC timestamp to a categorical *hour of day* value (24 hour clock). These features are used to build a neural model as described in section 5.3. The results for the metadata combinations are conducted on the LSTM based model, presenting the results in table 8.

To compare the results of different methods datesets, we are following previous work of tweet geolocation prediction by using the following evaluation metrics:

- *Median*: The median error distance between the predicted location and the true tweet location.
- *Accuracy*: The percentage of tweets assigned to the correct grid cell.
- *Acc@d*: The percentage of tweets assigned coordinates within a distance d of the true coordinates. Using $d = 0.5km$, $d = 1.0km$ and $d = 2.0km$.

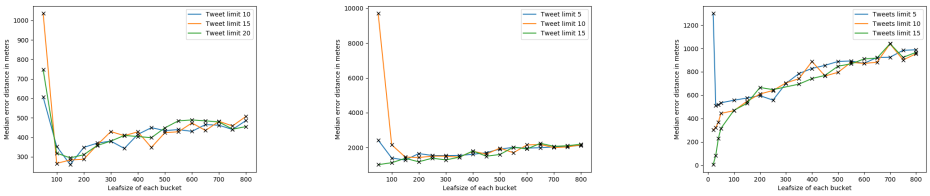
The results are divided into different sections, where we present the geographical modelling and evaluation metrics for Manhattan in section 6.4.1, Los Angeles in section 6.4.2 and Paris in section 6.4.3. These sections contains baseline comparisons under the evaluation metrics presented above, together with uniform against adaptive grid modelling. In section 6.4.4 we present results for our proposed method comparing the text only approach against the text and metadata approach.

6.3 Tuning

Specific parameters were tuned for optimal results; (1) bucket size and tweets limit, (2) uniform grid size of each cell, (3) the partition location method. All results are tuned with respect to median error distance.

Bucket size and tweets limit. For the bucket size there is a golden middle way between large and small buckets. Too large buckets tend to produce vast areas with higher average distance to the center or centroid. This will affect the precision even when the correct leaf is chosen. There is also a greater risk for contextual separated areas to be mashed together with larger bucket sizes.

On the other hand, too small bucket sizes should also be avoided due to fewer documents available for training. Very small leafs could potentially split up areas with the same location keywords attached to them (eg. street name), and hence the model has an almost impossible task distinguishing different areas. One way of enforcing a minimum number of tweets in each leaf is by introducing a threshold variable. Leaves with a tweet count below a specific *tweets limit* should be discarded, leading to fewer coarse areas. Here, there is also a trade off between discarding too few or too many tweets. A lower limit could possibly not solve the beforehand mentioned problem, whereas a high limit would probably take out far too many areas.



(a) Manhattan (b) Los Angeles (c) Paris

Figure 13: Threshold test of bucket size and tweets limit. All datasets favor smaller bucket sizes, but there seems to be no clear choice regarding tweets limit. There are some difference between the test-parameters, Manhattan was tested with slightly higher tweet limit and Paris has additional bucket size tests at lower range.

Figure 13 presents different tests with varying bucket and tweets limit sizes. Three different tweets limits and a span from 50 to 800 with increments of 50 as bucket size were tested. Additionally, bucket size 20, 30 and 40 was tested for the Paris dataset, as initial runs indicated that we needed further investigation of optimal bucket size. For the Manhattan dataset, as seen

in figure 13a, we found optimal bucket size of 150 and tweets limit of 10. Similar results for the Los Angeles dataset in figure 13b, best performing parameters are bucket size of 50 and tweets limit of 15. Finally, we have Paris (13c) with exactly the same parameters as Los Angeles, bucket size of 50 and tweets limit of 15.

Cell size. Threhshold tests of optimal cell size for the uniform grid partitioning. Following tests was conducted; cell sizes of 1.0, 1.5, 2.0 and 3.0 km. Paris also had one test run at 0.707 km. According to the results in figure 14, we choose a cell size of 1 km for the **Manhattan** dataset, 1.5 km for the **Los Angeles** dataset, and 0.707 km for the **Paris** dataset.

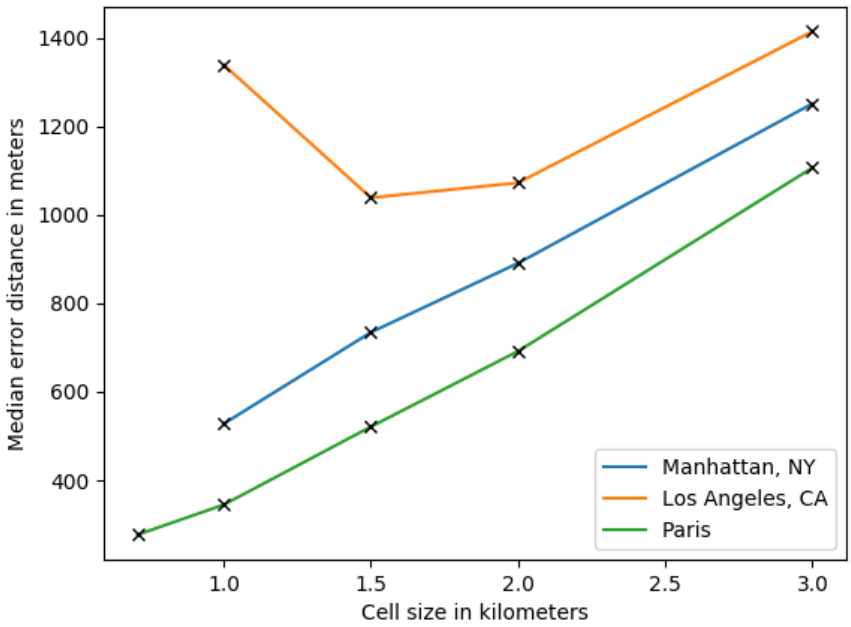


Figure 14: Threshold test of cell size. Clear advantage for smaller cells, especially for Manhattan and Paris. Paris also had one additional cell size test due to large area and number of tweets.

Partition location method. Table 4 shows that the *centroid* method always performs better than *center*, although the difference is quite low. All three datasets have considerably small areas with very high density of tweets. In practice generating a majority of tiny cells which have an almost insignificant difference between *center* and *centroid*. Datasets spanning over greater areas would most certainly have even more superior results with the *centroid* method compared to the *center* method.

Table 4: Comparing centroid vs center for partition location method. Best result in bold.

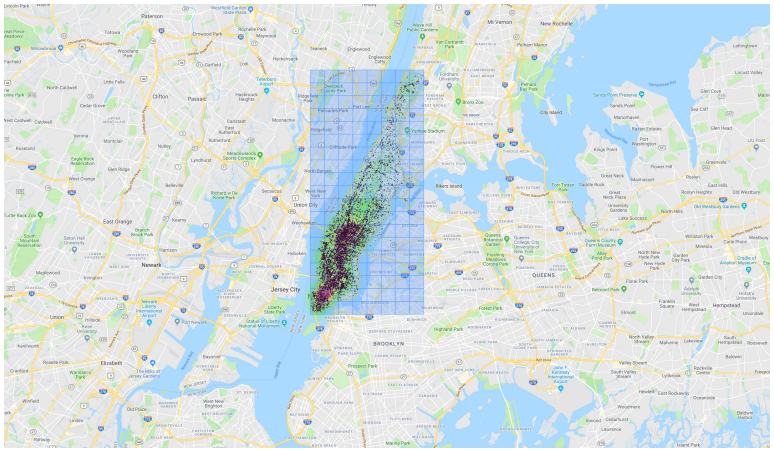
Grid	City	Centroid	Center
Uniform	Manhattan	**512**	565
	Los Angeles	**910**	1052
	Paris	**277**	353
Adaptive	Manhattan	**259**	289
	Los Angeles	**1013**	1039
	Paris	**315**	355

6.4 Evaluation Results

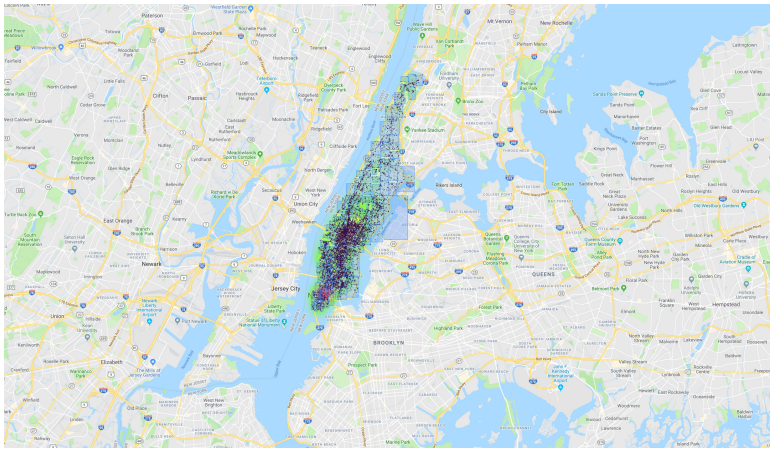
This section contains results for the different methods and datasets. We compare the baseline method *LocKDE* against our two proposed methods; *LSTM* and *LSCN*. The comparison includes both uniform and adpative grid measured against the evaluation metrics described in section 3.8.

6.4.1 The Manhattan dataset

The Manhattan dataset is described in section 6.1.1. The distribution of tweets for both the uniform and adaptive approach is shown below in figure 15.



(a) The uniform grid representation of Manhattan using grid cells of $1x1$ kilometers.



(b) The adaptive grid representation of Manhattan

Figure 15: Geographical modelling of Manhattan

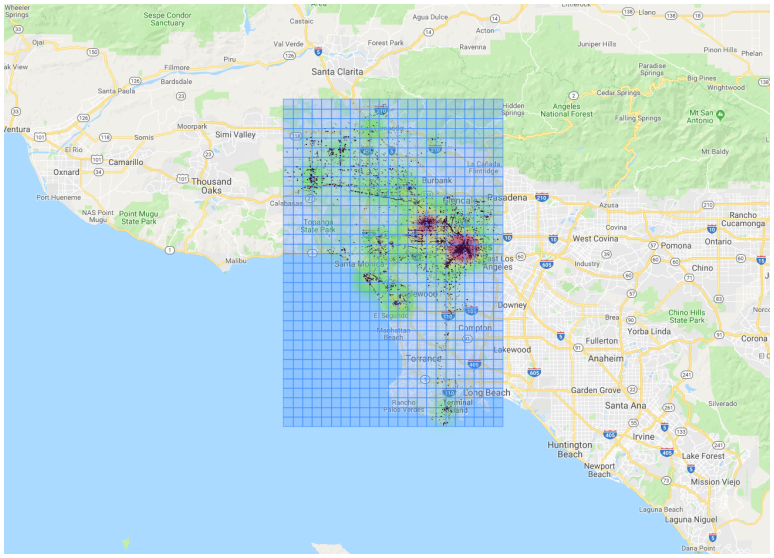
The evaluation metrics for the different methods are given in table 5. For the uniform approach, *LSTM* achieves better results for all metrics, with *LSCN* as a close number two. The adaptive grid produces superior results regarding *median error distance* and *Acc@0.5km*. However, *LSTM* with uniform grid has the best overall performance with highest score in three out of five metrics.

Table 5: Results for the Manhattan dataset

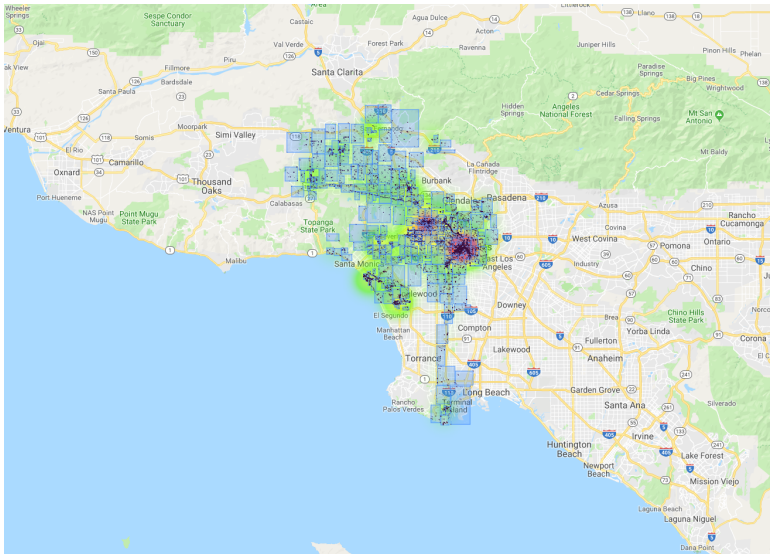
Grid	Evaluation metric	$LSTM$	$LSCN$	$LocKDE$
Uniform	Median (m)	512	514	543
	Accuracy	**0.573**	0.569	0.547
	Acc@0.5km	0.489	0.488	0.472
	Acc@1.0km	**0.619**	0.616	0.610
	Acc@2.0km	**0.745**	0.721	0.732
Adaptive	Median (m)	**271**	280	-
	Accuracy	0.494	0.492	-
	Acc@0.5km	**0.536**	0.534	-
	Acc@1.0km	0.589	0.585	-
	Acc@2.0km	0.699	0.683	-

6.4.2 The Los Angeles dataset

The Los Angeles dataset is described in section 6.1.2. The distribution of tweets for both the uniform and adaptive approach is shown below in figure 16.



(a) The uniform grid representation of Los Angeles using grid cells of $2x2$ kilometers.



(b) The adaptive grid representation of Los Angeles

Figure 16: Geographical modelling of Los Angeles

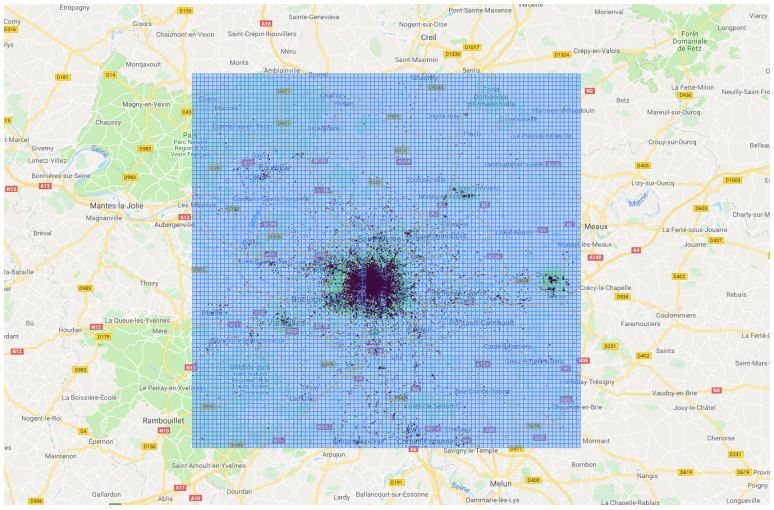
The evaluation metrics for the different methods are given in table 6. This yields fairly similar results for all models in terms of the uniform grid, but again the LSTM model is superior. Comparing the uniform and adaptive grid approach for this dataset shows best overall performance for the uniform grid, however, the adaptive grid yields an increase of approximately 10 % for the *Acc@0.5km* metric.

Table 6: Results for the Los Angeles dataset

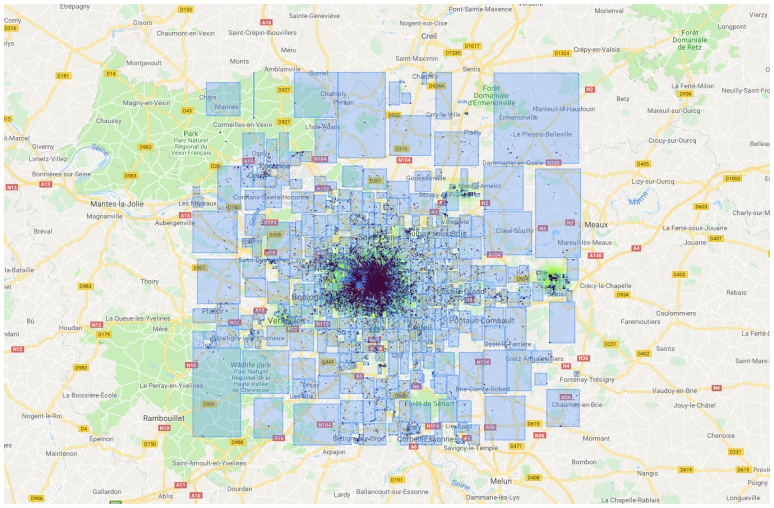
Grid	Evaluation metric	*LSTM*	*LSCN*	*LocKDE*
Uniform	Median (m)	**910**	982	933
	Accuracy	**0.519**	0.510	0.506
	Acc@0.5km	0.357	0.348	0.341
	Acc@1.0km	**0.523**	0.511	0.512
	Acc@2.0km	**0.614**	0.599	0.603
Adaptive	Median (m)	1002	934	-
	Accuracy	0.496	0.502	-
	Acc@0.5km	**0.462**	0.473	-
	Acc@1.0km	0.492	0.506	-
	Acc@2.0km	0.567	0.576	-

6.4.3 The Paris dataset

The Paris dataset is described in section 6.1.3. The distribution of tweets for both the uniform and adaptive approach is shown below in figure 17.



(a) The uniform grid representation of Paris using grid cells of $1x1$ kilometers.



(b) The adaptive grid representation of Paris

Figure 17: Geographical modelling of Paris

The evaluation metrics for the different methods are given in table 7. This yields impressive results for the *LSTM* model, beating *LSCN* and *LocKDE* by a substantial margin. Also noteworthy is the median error distance value

of *LSTM* for adaptive grid.

Table 7: Results for the Paris dataset

Grid	Evaluation metric	*LSTM*	*LSCN*	*LocKDE*
Uniform	Median (m)	277	625	624
	Accuracy	**0.641**	0.471	0.435
	Acc@0.5km	**0.646**	0.487	0.477
	Acc@1.0km	**0.681**	0.523	0.532
	Acc@2.0km	**0.715**	0.584	0.589
Adaptive	Median (m)	**58**	681	-
	Accuracy	0.534	0.428	-
	Acc@0.5km	0.610	0.496	-
	Acc@1.0km	0.634	0.536	-
	Acc@2.0km	0.684	0.587	-

6.4.4 The London dataset

The London dataset is described in section 6.1.4 and is divided into adaptive grid cells as shown in figure 18.

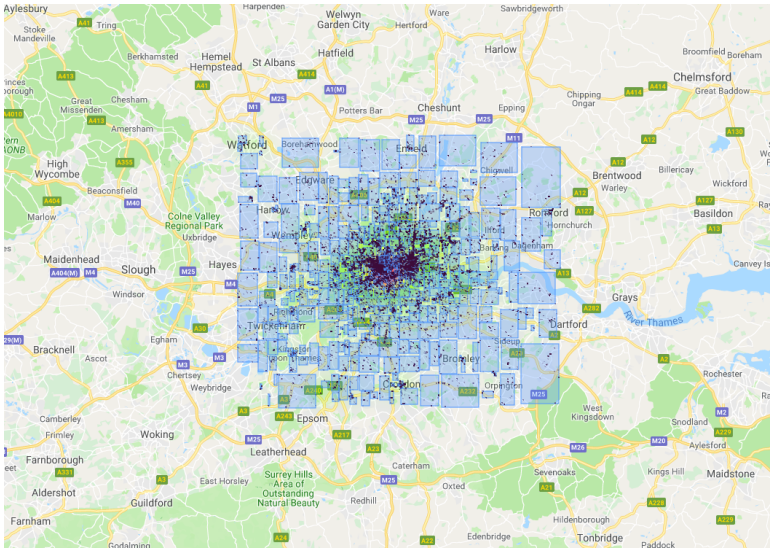


Figure 18: Adaptive grid representation of London with contained tweets

The evaluation metrics for the different metadata combinations are given in table 8. This yields an increasing prediction performance the more metadata features incorporated into the model. Thus, the best results are given by the model exploiting the features of *text*, *posting time*, *username*, *user language* and *user description*.

Table 8: Results for the London dataset

Evaluation metric	*Text*	*Text* *Posting time* *Username* *User language*	*Text* *Posting time* *Username* *User language* *User description*
Median (m)	798	626	**427**
Accuracy	0.466	0.488	**0.517**
Acc@0.5km	0.464	0.486	**0.515**
Acc@1.0km	0.514	0.535	**0.571**
Acc@2.0km	0.572	0.592	**0.638**

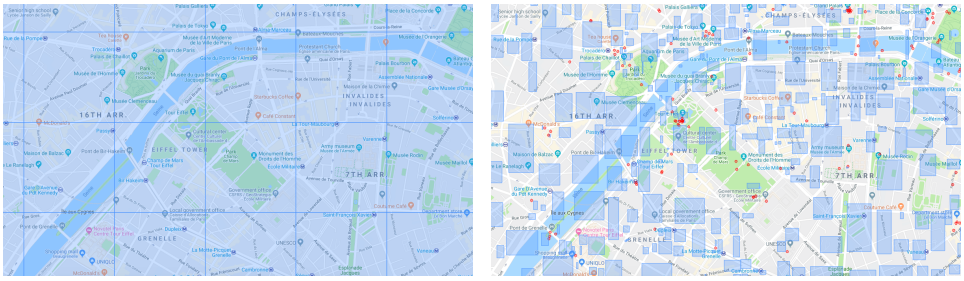
6.5 Evaluation Discussion

For our geographical modelling and baseline comparison given for the different datasets in table 5, table 6 and table 7, the best results are written in **bold**. The *Median* metric has been chosen as our primary comparison basis, and all models are selected based on their median performance. Although, *Accuracy* within a certain distance would indeed serve as a satisfactory baseline and there seem to be a strong correlation between *Median* and *Acc@n*. All further discussion with reference to performance will be in regard to these metrices, if not otherwise stated. This yields that our proposed methods using the *LSTM* based network significantly outperforms the baseline of *LocKDE* and the *LSCN* network for all datasets.

To investigate the differences in tweet prediction we examined each model using a small test set in order to identify which tweets were classified correcly, and which ones were not. This examination indicates that the *LocKDE* method relies heavily on location indicative terms like place and street names, and classifies multiple tweets to the correct area, but to neighbouring cells instead of the correct grid cell. For the neural models it is hard to identify a pattern as both tweets with and without location indicative terms are correctly classified. Overall the accuracy difference comparing the models is relatively limited, but there is a great difference in which tweets each approach classified correctly. The neural models have the ability to learn more complex patterns automatically, resulting in correct prediction of a greater variety of tweets. However, the neural models seems to misclassify a small fraction of tweets by a substantial margin, which is an event occurring less frequently for the *LocKDE* method. In terms of the *LSCN* compared to the *LSTM* model the results indicates that the *LSCN* model does not benefit from the feature extraction in the convolutional layer, and thus resulting in slightly worse performance than the pure *LSTM* model.

6.5.1 Grid partitioning method

There are some differences in terms of best results comparing the uniform and adaptive grid approach. The *Accuracy* metric is not directly comparable due to a various number of grid cells and size. All runs are performed at city level, in small defined areas with high density of data points. This precondition allows small partitioned grids to be designed, as seen in figure 19. Both approaches favor smaller cell size as we can see from figure 14 and 13, producing a preponderance of tiny grid cells. Nonetheless, the adaptive grid has potential to partition even further and create a finer grid layout.



(a) Uniform grid (b) Adaptive grid

Figure 19: Difference between uniform and adaptive grid partitioning for the area surrounding the Eiffel Tower. Blue squares represents grid cells and red 'dot' indicates that the cell is one single point. The high concentration of tweets can not be represented by (a) and it encapsulates the Eiffel Tower area in four big cells compared to (b).

Still, there are no clear advantage to either methods and they perform quite similar on the Paris dataset. However, uniform grid partitioning for Paris at 0.707 km requires substantial computing power to be generated, and this will grow exponentially with respect to the grid size. In case of the adaptive method, increase in grid size would not penalize the performance as much due to grid cells adjusting size based on density and ignoring spaces where no tweets are located, making adaptive grids more scalable and robust to vast areas.

Considering the *Grand Palais* in figure 20, a large historic site, exhibition hall and museum located at the Champs-Élysées. The uniform grid locates the whole palace and some additional nearby area into one grid cell, as opposed to the adaptive method which partition it into multiple cells. A test set of 20 tweets located in the Grand Palais was classified by both models and the results are listed in table 9. As expected, all tweets are classified correct by the uniform model, but with a median error distance of 155 meters. This indicates that the centroid is approximately 150 meters away from the Grand Palais, and all predictions belonging to this place would suffer a penalty regarding the median error distance metric.

The adaptive grid does in practice the opposite, splitting a contextual identical area into multiple smaller grid cells. Effectively initiating a system where distinguishing between the different grid cells are in fact an almost impossible task. The accuracy tells us that only 14 out of 20 tweets got classified correct, but since the cells are all mashed together, the overall performance does match the uniform model. This small example illustrates some of the advantages and shortcomings for both methods. A more optimal

Table 9: Results for the Grand Palais in Paris, running a small test set containing 20 tweets.

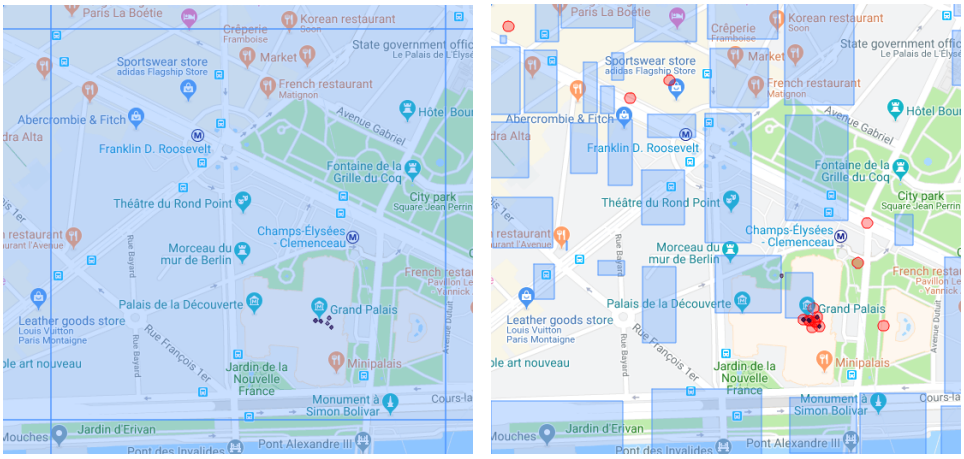
Evaluation metric	Grid	
	Uniform	Adaptive
Median (m)	155	0
Accuracy	1.0	0.7
Acc@0.5km	1.0	0.95
Acc@1.0km	1.0	0.95
Acc@2.0km	1.0	0.95

solution is probably something in between, an assembly where small nearby cells are merged together, creating a single cell representation of smaller areas belonging to the same location. Certainly, there is an option to enlarge the bucket size for the kd-tree, uniting some of the smaller cells, but this would also without question raise the average cell size and potentially create huge cells in less dense areas. An approach where large buckets are allowed and the average cell size is still below an acceptable limit, should be desirable. At this point, the kd-tree implementation is not capable of such requirements, and necessary modifications are required.

6.5.2 Partition location method

Results from table 4 indicates a clear advantage for the *centroid* over the *center* method for partition location. Centroid outperforms the latter for all datasets and grid partition types. Still, there is no major difference in the resulting median error distances among the two methods, nevertheless, all have been beaten by a substantial margin. *Center* predicts on average 18% longer median error distances for uniform grid partition than centroid, and 9% longer for adaptive grid partitioning.

Interestingly, uniform grids seem to benefit greater from the centroid method. The adaptive approach will create grid cells covering just *one single* coordinate in areas with the most extensive number of tweets. Take figure 19 as an example, *The Eiffel Tower* is an tremendously popular tourist attraction located in Paris, and vast amounts of people will tweet from the exact same location and nearby. This phenomena would force adaptive grids to create single point or immensely small cells, as seen sin figure 19b, which implies that the partition location method used is almost irrelevant. Obviously,



(a) Uniform grid (b) Adaptive grid

Figure 20: Two test sets, one for each partition method, containing 20 tweets located inside the *Grand Palais*. Tweets are marked as purple points on the map, darker color indicates multiple points at the same location. (a) partition the palace into one single grid cell, including adjacent locations outside the palace. (b) splits the palace into numerous smaller grid cells, even separating cells that are essentially on top of each other.

there will always be larger cells present, but the chunk of tweets contained in these are modest in comparison. From figure 17 showing tweets distribution in Paris, it is easy to spot an abundance of tweets in the city center compared to the surrounding cells. Uniform grids on the other hand, does not get influenced by tweet density, and all cells will in some sense have an positive effect of the *centroid* method.

6.5.3 Metadata

In terms of our method exploring the effect of incorporating metadata features into the neural models, the results are presented in table 8 for the London dataset. The leading results are presented in bold. This shows that the more contextual features utilized in combination with text the better results. Comparing the pure *text* model against the model exploiting *posting time*, *username* and *user language* yields an improved *median error distance* of 178 meters, and an improvement in accuracy metrics of approximaterly 2 %. Ragarding the *posting time* feature this may indicate that Twitter users are active at different hours of the day for various geographical regions. This feature is similar to the *Timezone* feature, but is used to explore temporal differences within the same timezone. If the test was conducted

for datasets covering a larger geographical area spanning plural timezones, the *Timezone* feature could provide valuable spatial information. The *user language* feature represents the user's preferred language, which may give information about the geographical origin of the user. An improvement in accuracy by exploiting this feature may imply that users with the same preferred language or geographical origin tweets from the same geographical areas. This makes sense by for example the demographics of London where people with the same ethnicity tend to settle down in particular geographical regions. The *username* feature mostly consists of random names which may not provide any spatial information beyond mapping the same user to the same location for different tweets. However, some usernames contains geo-indicative terms which may increase the prediction accuracy.

By expanding the previous metadata model by including the *user description* feature, we obtain an improvement in *median error distance* of 371 meters, and an improvement of 5-6 % for the different accuracy metrics compared to the pure text model. This yields an additional improvement of approximately 200 meters in *median error distance* and 4 % in the accuracy metrics just by adding the *user description*. The user description is a summary describing the user and expressing current interests in maximum 160 characters. By exploiting this feature we provide a considerable amount of contextual information to each tweet, and by reviewing the dataset we observe that most descriptions contain clear location indicative terms. This yields a significant improvement in prediction accuracy.

Twitter data contains an extensive amount of contextual metadata which may not be the case for all relevant domains or platforms considering the geolocation problem. However, if such metadata is available, these results states that it should be incorporated into the models to improve prediction accuracy.

7 Conclusion and further work

In this project we have investigated the use of deep learning to issue the geolocation problem, where we proposed a method using recurrent neural networks in combination with a grid based approach for geographical modelling. Based on information in a single tweet we predict the grid cell most probable of containing the given tweet. The inferred location is derived from the grid cell using the centroid of all data points present in the predicted cell. The evaluations conducted on three datasets representing different geographical regions yields a significant improvement in accuracy compared to state of the art methods. Our experiments comparing pure text models against models exploiting contextual metadata yields a significant improvement in prediction accuracy by extending the feature space with features like *posting time*, *user language* and *user description*. Due to the strong results obtain in this project it is obvious that deep learning deserves further investigation regarding the geolocation problem. The implementation is available at Github.

In future work the proposed method can be applied to infer fine-grained locations on a larger scale than relatively small urban areas like Paris and London. On a global scale the whole world map can be used as the geographical area of interest, which may set higher requirements to the geographical modelling. Other methods for grid partitioning will be explored, and clustering of data seems like a very interesting approach to investigate more closely. In terms of the metadata models we plan to exploit features like images and content behind linked URI's to strive for more accurate predictions.

References

Allison Gyle Woodruff, C. P. (1994). *Automated geographic indexing of text documents.*

Amitay, E., Har'El, N., Sivan, R., & Soffer, A. (2004). Web-a-where: geotagging web content. In *Proceedings of the 27th annual international acm sigir conference on research and development in information retrieval.* SIGIR.

Bengio, Y., Simard, P., & Frasconi, P. (1994, March). Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, *5*(2), 157–166. Retrieved from `http://dx.doi.org/10.1109/72.279181` doi: 10.1109/72.279181

Benjamin P. Wing, J. B. (2011). *Simple supervised document geolocation with geodesic grids.*

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517.

Berggren, M., Karlgren, J., Ostling, R., & Parkvall, M. (2016). *Inferring the location of authors from words in their texts.*

Bigi, B. (2003). Using kullback-leibler distance for text categorization. In *Proceedings of the 25th european conference on ir research* (pp. 305–319). Berlin, Heidelberg: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=1757788.1757818`

Chen, Y.-C. (2017, 04). A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology*, *1*. doi: 10.1080/24709360.2017.1396742

Dabel E. Rumelhart, R. J. W., Geoffrey E. Hinton. (1988). *Learning representations by back-propagating errors.*

Dai, W., Xue, G.-R., Yang, Q., & Yu, Y. (2007). Transferring naive bayes classifiers for text classification. In *Proceedings of the 22nd national conference on artificial intelligence - volume 1* (pp. 540–545). AAAI Press. Retrieved from `http://dl.acm.org/citation.cfm?id=1619645.1619732`

Dredze, M., Osborne, M., & Kambadur, P. (2016). *Geolocation for twitter: Timing matters.*

Egas, R., Huijsmans, N., Lew, M., & Sebe, N. (1999). Adapting kd trees to visual retrieval. In *International conference on advances in visual information systems* (pp. 533–541).

Gers, F. A., Schmidhuber, J. A., & Cummins, F. A. (2000, October). Learning to forget: Continual prediction with lstm. *Neural Comput.*, *12*(10), 2451–2471. Retrieved from `http://dx.doi.org/10.1162/089976600300015015` doi: 10.1162/089976600300015015

Goltsman, K. (2017). *Introduction to artificial neural networks.*

Gundersen, O. E., & Kjensmo, S. (2018). State of the art: Reproducibility in artificial intelligence. In *Thirty-second aaai conference on artificial intelligence.*

Hochreiter, S., & Schmidhuber, J. (1997, November). Long short-term memory. *Neural Comput.*, *9*(8), 1735–1780. Retrieved from `http://dx.doi.org/10.1162/neco.1997.9.8.1735` doi: 10.1162/neco.1997.9.8.1735

Huang, B., & Carley, K. M. (2017). On predicting geolocation of tweets using convolutional neural networks. In *International conference on social computing, behavioral-cultural modeling and prediction and behavior representation in modeling and simulation.* Springer.

Iso, H., Wakamiya, S., & Aramaki, E. (2017). *Density estimation for geolocation via convolutional mixture density network.*

Jurgens, D. (2013). *That's what friends are for: Inferring location in online social media platforms based on social relationships.* AAAI.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *Ann. Math. Statist.*, *22*(1), 79-86.

Li, w., Eickhoff, C., & de Vries, A. (2014). *Geo-spatial domain expertise in microblogs.*

Lifang, Y., Sijun, Q., & Huan, Z. (2017). Feature selection algorithm for hierarchical text classification using kullback-leibler divergence. *2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 421-424.

Lipton, Z. C., & Berkowitz, J. (2015). A critical review of recurrent neural networks for sequence learning. *CoRR*, *abs/1506.00019*.

Miura, Y., Taniguch, M., Taniguchi, T., & Ohkuma, T. (2017). *Unifying text, metadata, and user network representations with a neural network for geolocation prediction.*

Özdikis, O., Ramampiaro, H., & Nørvåg. (2018a). Locality-adapted kernel densities for tweet localization. In *Proceedings of t he 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR 2018)* (pp. 1149–1152). ACM. Retrieved from `http://doi.acm.org/10.1145/3209978.3210109` doi: 10.1145/3209978.3210109

Özdikis, O., Ramampiaro, H., & Nørvåg. (2018b). Spatial statistics of term co-occurrences for location prediction of tweets. In *Proceedings of ecir 2018.* Springer.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318).

Paul S. Earle, M. G., Daniel C. Bowden. (2011). Twitter earthquake detection: earthquake monitoring in a social world..

Pengfei Li, N. K. S. Z., Hua Lu, & Pan, G. (2018). Location inference for non-geotagged tweets in user timelines. In *Ieee transactions on*

knowledge and data engineering. IEEE.

Rahimi, A., Cohn, T., & Baldwin, T. (2017). *A neural model for user geolocation and lexical dialectology.*

Rish, I. (2001). An empirical study of the naive bayes classifier. In *Ijcai 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, pp. 41–46).

Roller, S., Speriosu, M., Rallapalli, S., Wing, B., & Baldridge, J. (2012). Supervised text-based geolocation using language models on an adaptive grid. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning.* EMNLP-CoNLL '12.

Schmidhuber, J. (2014). *Deep learning in neural networks: An overview.*

Su, J., Sayyad Shirab, J., & Matwin, S. (2011, 01). Large scale text classification using semisupervised multinomial naive bayes. In (p. 97-104).

Teevan, J. (2003). Tackling the poor assumptions of naive bayes text classifiers. In (pp. 616–623). Retrieved from `http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.8572`

Thomas, P., & Hennig, L. (2017). Twitter geolocation prediction using neural networks. In *International conference of the german society for computational linguistics and language technology.* Springer.

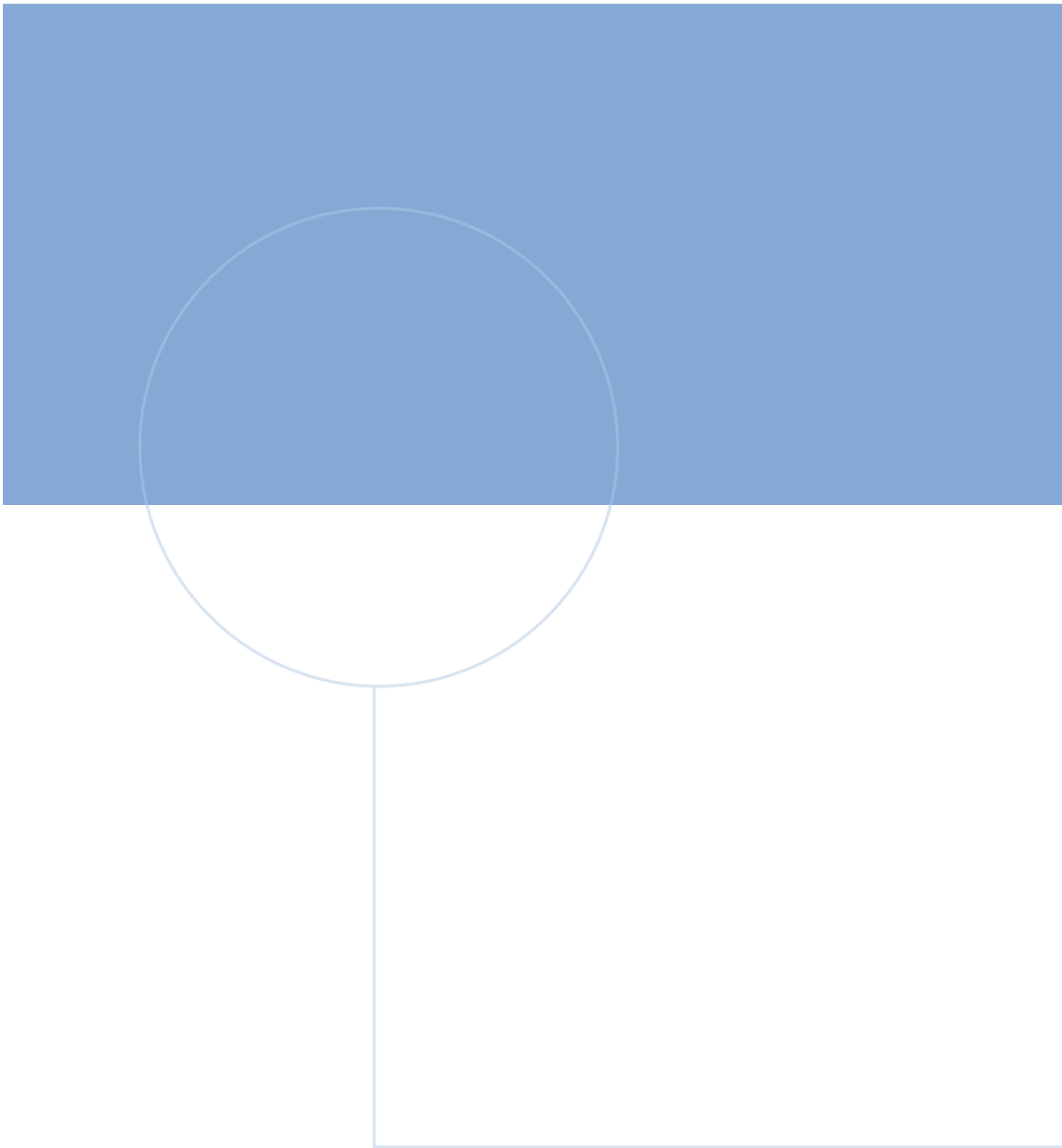
Wu, J. (2017). *Introduction to convolutional neural networks.*

Xin Chen, E. A. F. W., Yu Wang. (2015). A comparative study of demographic attribute inference in twitter. In *Ninth international aaai conference on web and social media.* AAAI.

Yang, Y., & Liu, X. (1999). A re-examination of text categorization methods. In *Proceedings of the 22nd annual international acm sigir conference on research and development in information retrieval* (pp. 42–49). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/312624.312647` doi: 10.1145/312624.312647

Zheng, X., Han, J., & Sun, A. (2018). A survey of location prediction on twitter. In *Ieee transactions on knowledge and data engineering.* IEEE.

Zhou, C., Sun, C., Liu, Z., & Lau, F. (2015). A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*.



NTNU
Norwegian University of
Science and Technology