Synthetic GPS Traces for Privacy Protection

Ross Kelso - 160002045

18th January 2021

MSci Project
School of Computer Science
University of St. Andrews

Abstract

This project explores the possibility of using machine learning techniques to generate synthetic GPS traces that cannot be distinguished from real GPS traces. By doing this we hope to be able to use these synthetic traces in place of real data on GPS enabled devices to protect the user's privacy. Markov model based generation algorithms and generative adversarial neural networks are explored as potential solutions. The synthetic traces are evaluated by human participants and by a classifier designed to identify synthetic traces. The Markov model based approaches perform well at fooling humans whereas the generative adversarial network performs better at fooling a neural network classifier.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 16,781 words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Contents

1 Introduction

Modern devices such as smartphones provide many benefits to their users in their day-to-day lives. These devices are generally GPS-enabled which allows apps and services to use GPS data to provide services. For example, mapping apps use the user's location to help them navigate, fitness apps use speed and distance information to track the user's fitness and dating apps use the user's location to help match them with nearby partners. These and other services use the GPS data provided by a GPS-enabled device to provide benefit to the user.

However, GPS traces contain personal data about their subjects that presents a significant privacy risk. A GPS trace of a person going about their daily life will include data such as which shops they visit, where they work and their home address. When combined with GPS traces from other people it may also indicate who their friends, colleagues and partners are. However, GPS based services require access to this data in order to provide a useful service. Even if users trust the specific apps they are using data breaches are common and so sensitive data may be leaked [37]. Additionally, GPS data that is used for a legitimate purpose by an app may be shared with a number of third party advertising and analytics companies [18].

The accuracy and type of GPS data needed depends on the app or service. For example dating apps don't need precise location data or velocity information in order to match users with other nearby users. However in order for fitness apps to provide accurate metrics they need accurate distance, velocity and altitude readings. Additionally navigation apps need to accurately know the user's position so that they can provide directions to their destination.

This report presents a tool which can synthesise realistic-looking synthetic GPS traces. By injecting realistic looking synthetic GPS data into the data stream of a GPS-enabled device (such as a smart phone) this project aims to prevent GPS-enabled services from collecting personal data and therefore protect the privacy of the user. This project will use machine learning techniques to generate synthetic GPS data and also evaluate how realistic this synthetic data is compared to real GPS data.

In order for a synthetic GPS trace to be useful it must be as indistinguishable as possible from a real GPS trace. Otherwise services or attackers who gain access to the GPS data will be able to identify that the data has been falsified. If a service's business model depends on them selling or otherwise profiting from user data they may ban or restrict service to users who are not providing data that is of value. Therefore users would have no choice but to give up their privacy in order to use these services. Therefore it would be beneficial to create an algorithm that produces GPS traces that appear indistinguishable from real GPS traces. Additionally, if users trust certain apps or services but not others it would be beneficial to allow the user to provide real data to trusted apps but synthetic data to untrusted apps.

1.1 Overview of Objectives

The objectives of the project are as follows (completed objectives are marked with a ✓):

Primary Objectives

- ✓ Create an algorithm for generating realistic looking synthetic GPS data
- ✓ Evaluate whether humans can distinguish between real and synthetic GPS traces
- ✓ Evaluate if computational techniques can be used to determine whether a GPS trace is real or has been generated

Secondary Objectives

- ✓ Generate synthetic GPS data tailored for a specific user
- Insert generated GPS data into the data-stream on a real device
- Have the option of providing different data streams (real/synthetic) to different apps (e.g. providing real data to trusted apps and synthetic data to untrusted apps)

All of the primary objectives were met. Additionally, the 1st secondary objective is also met, as the Markov model can be used to match the speed and distance of an existing user's trace. The objectives relating to injecting generated GPS data into a real GPS-enabled device were not completed due to technical challenges and time constraints. Priority was given to exploring the primary objectives in grater depth rather than focusing on the technical challenges of the secondary objectives.

1.2 Software Engineering Process

This project is implemented as a series of Python [41] Jupyter notebooks [21]. Several libraries are used for data processing and machine learning. The main machine learning library used for this project is TensorFlow [7]. Sci-kit learn [33] is used for various metrics and utility functions. Data pre-processing is done using the pandas [29] and numpy [15] libraries. FLANN [30] (as implemented by pyflann-py3 [38]) is used for finding nearest neighbours. Matplotlib [17] and seaborn [42] are used for plotting and visualisation. Folium is used to visualise traces on maps [40].

This project is implemented using a modular structure. The data loading and pre-processing steps were performed in separate notebooks to the implementation of the various generation algorithms. The processed GPS and map data were exported and reloaded in the Markov model and GAN notebooks. This allowed the pre-processing logic for all models that used the same data structures to be consistent while minimising code

duplication. Each notebook implementing a generation algorithm also generates and exports a data set of synthetic traces. These are imported by the classifier notebook to evaluate the performance of the various algorithms.

This modularity simplifies development as changes to how the data is pre-processed will only result in changes to the pre-processing notebooks and not the generator notebooks. Additionally, the classifier notebook can easily load any synthetic data set. Therefore adding a new generator algorithm doesn't require any modification to the classifier notebook.

1.3 Ethics

The ethical considerations of this project are the user study and the use of secondary data. The secondary data used is completely anonymised and there was no means to de-anonymise it. No attempt was made to de-anonymise the data. The user study is covered by the artefact evaluation form (ethical approval code CS12476) since the user study is a short questionnaire of question collected anonymously and contain no personal information. The artefact evaluation form for this project is included in appendix B.

1.4 Overview of Thesis

To meet the aims of this project several algorithms were developed in order to generate synthetic GPS traces. There are two major sources of data for this project: map data and GPS data. These algorithms are based on two major techniques. The first is using the map data directly by generating traces by traversing the road network (Markov models 4). The second is using neural networks to generate traces directly by learning patterns from the real GPS traces (generative adversarial neural networks 6).

The context survey (section 2) gives an overview of previous work in this area and the machine learning and mathematical basis for the methodologies used by the GPS trace generation algorithms. The data set section (section 3) discusses the structure of the data sets that were used for this project and how they were processed to be in forms that are useful to the machine learning algorithms developed for this project.

The sections 4 and 6 explain the design and implementation of the trace generation algorithms. Additionally, a neural network based classifier was developed to evaluate whether computational techniques can be used to identify synthetic GPS traces. Section 5 discusses the design of the neural network classifier.

In addition to using computational techniques to identify synthetic traces a user study was also conducted. In the user study participants were asked to determine whether traces they were shown were real or fake. In section 7 the algorithms developed for this project are evaluated. The algorithms are assessed based on how likely their generated traces are to be misclassified as real by the classifier or the human participants.

2 Context Survey

In this section the relevant previous works that provide background to this project will be discussed. This includes existing methods for preserving privacy in GPS data sets, as well as existing methodologies for generating synthetic GPS data. Additionally, the foundations of the neural network and Markov model based algorithms developed by this project are discussed.

2.1 Privacy and GPS Anonymisation Techniques

k-anonymity is a measure of the how well a data set preserves the privacy of it's subjects. A data set has k-anonymity if attempting to link identifying information to the data set would result in an ambiguous mapping to at least k entities [36]. In other words, for any entity in the data set there are k-1 other entities that cannot be distinguished from it. Typically this is achieved by using generalisation and suppression techniques.

Generalisation is the process of making the data less specific in order to reduce the number of separate values. For example if a data set contains the age of individuals there may not be many individuals for any specific age making them easier to re-identify (say there is only one person who is 35). By creating age bands (e.g. 30-39) more entries will occur in each band making them more ambiguous (there may be 15 people who are between the ages of 30 and 39).

Suppression is the process of removing entries that are outliers such that they are unique within the data set. If there is one entry that is unique after generalisation has been applied removing this entry from the release will increase the k-anonymity.

GPS traces contain significant amounts of personal information about their subjects. It is therefore beneficial for preserving user privacy to anonymise GPS traces before they are released to the public. However GPS traces are useful for a variety of applications so research has been conducted into how to protect the privacy of individual users while maintaining the utility of the GPS traces.

GPS traces are more complex than many other standard data types so research has been conducted into methods for applying k-anonymity to GPS traces. k-anonymity can be provided in GPS data by generalising the accuracy of GPS samples. If samples are specified to be within an area instead of a point then multiple users will inhabit the area simultaneously and the individual users will not be identifiable [14]. This form of spatial and temporal generalisation can be used to guarantee differential privacy as well as k-anonymity [9].

Other approaches to solving this problem are adding random perturbations to the GPS locations and snapping GPS coordinates to a grid (thereby lowering the granularity and resolution of the data) [39]. This kind of modification to the GPS coordinates preserve users' privacy with the trade-off of reducing the spatial accuracy of the data. The exact requirements of the data's spatial accuracy depends on the application of the data, some applications may require more precision than others.

Therefore research has been conducted into how to maximise anonymity while preserving the spatial accuracy of the data. This involves increasing the amount of obfuscation and measuring how much anonymity is preserved compared to the loss in accuracy. Different methods of obfuscation have different trade-offs. For example snapping GPS positions to a grid has the best performance in terms of protecting privacy but a large snapping grid reduces the spatial accuracy of the data. The level of obfuscation will therefore depend on the spatial accuracy required by the target application. [39]

Some applications require high spatial accuracy in order to function, such as traffic monitoring. A GPS trace will hinder a users privacy if it contains sensitive information (such as stopping at a sensitive location). The longer a trace is, the more likely it is that it will contain sensitive information as those traces are more likely to contain a sensitive location and an identifiable location (e.g. home or work address). Because GPS tracking points are temporally and spatially correlated individual anonymous samples can be reconstructed into identifiable traces by projecting likely next sample locations to join the samples into traces. However in certain situations if the time between samples is great or the number of similar traces in an area is high multiple samples will appear to be potential next sample in the trace. This is called "confusion" as a potential reconstruction algorithm is now confused about which point is next in the trace and therefore can no longer reconstruct it. The "time to confusion" is the minimum amount of time that must pass before any reconstruction algorithm will reach confusion. By removing samples that do not meet a specified confusion level it is possible to release GPS samples with low likelihood of individual traces being reconstructed. [16]

All of these GPS privacy measures decrease the temporal or spatial resolution of the GPS data. Different applications require different levels of accuracy. In America the requirements of the Enhanced 911 Service (E911) specify that mobile providers should be able to identify a users location between 50 and 300m depending on technology and conditions [34]. This information is intended to be used to allow emergency services to find the location of the caller and would be used to identify the building that the caller is in in order to more effectively deal with any emergency. Therefore, accuracy within this range will be sufficient for many applications. However an accuracy of 100m could easily place the user in a different building which may be problematic for uses by the emergency services but would potentially protect the users privacy as their exact position is unknown.

For users particularly concerned about their privacy one option is to disable GPS on their device. This is a feature of many smart phones [11]. However doing this prevents location enabled services from working. Some applications will still function properly with poor location accuracy, therefore iOS 14 allows users to reduce the precision of their GPS data passed to individual apps, doing this allows apps with low accuracy requirements to function while protecting the user's privacy [19]. However GPS inaccuracy will prevent some GPS apps from functioning effectively. It is therefore beneficial to research methods of modifying GPS data in a way that protects the privacy of the user but preserves the utility of the GPS data.

2.2 Methods for Injecting GPS Data into Data-Stream

A common use case for providing synthetic GPS values is in order for developers to test GPS-enabled apps. Android devices can select a mock location app from their developer options. This allows developers to test location based functionality without having to physically move the device.

Several apps exist to provide functionality for selecting fake GPS data [20][24][26]. However, it is possible to detect these apps due to the artificial nature of the data they provide. For example the location reported by a real GPS devices fluctuates as there is an uncertainty with each measurement. GPS mocking apps don't have this limitation and will repeatedly report the same exact position value. This lack of noise can be used to detect that the location is being mocked. However, some mocking apps like GPS Joystick deliberately add random noise to the reported value to counteract this effect. [31]

The Android API allows developers to create drivers that report GPS values to the Android location service [13]. Only one GPS driver can be registered concurrently. This is likely to cause technical issues for applications where the true GPS data is being used to inform parameters of the synthetic GPS data, which would be desirable for some applications.

2.3 GPS Trace Generation

Several research works have been produced on the subject of artificially generating GPS traces. Several of these works have been for the purpose of generating data sets to assess the performance of spatial-temporal database systems. In this case it is beneficial for generated data to be as realistic as possible such that performance metrics are accurate to real world situations.

A network-based generation approach uses the concept that the traversal area is a network (such as a network of paths or roads). The generator can then generate traces by traversing between nodes in this network. In order to accurately simulate real world networks (such as road systems) specific properties of the network, its nodes and edges must be considered. For example the maximum speed and capacity of network edges must be taken into account in order for the data to accurately reflect the real world. Additionally the speed and route of moving objects within the network depend on the other moving objects that share the network. [4]

Not all types of GPS traces follow the edges of a network. Therefore, for these more free-form traces a different form of generator will be required. An example of such an application is fishing ships. In this model ships move freely through the water but there are specific areas that they either approach or avoid. The ships will try to avoid stormy areas but will actively approach areas with shoals of fish. Therefore a free-form GPS trace generator can model the behaviour of these ships by computing repulsion and attraction vectors. Repulsion vectors point directly away from negative areas and the magnitude of the vector is inversely proportional to the distance from that area. Attraction vectors point towards positive areas and have magnitude inversely proportional to the distance

to the area. The sum of these vectors is used to determine the direction the ship will take at each time step. Although this work was originally applied to fishing boats the same principles could be applied to other types of movement by defining suitable positive and negative areas. [35]

These domain-based models for generating GPS traces are good because they are able to generate large amounts of GPS data. However the distribution and properties of the synthetic data may not accurately reflect real world data due to limitations of the designs of the models. In order to address this issue it is possible to augment the generator by accounting for the distribution of traces in real world data. If this is accomplished successfully the generated data set will have the same statistical properties as real world data sets without the privacy issues of releasing real world GPS data. In order to maintain the characteristics of the real data various properties of the real data can be used to inform generation. For example the distribution of origin and destination points can be used to generate traces with believable origins and destinations. Furthermore real world data can inform the speed and timings used on road networks, this allows the simulated traversal of the road network to be modelled accurately to real life. [2]

2.4 Neural Networks

A neural network is a machine learning technique that can be applied to a wide variety of problems. Neural networks can be used to generate realistic synthetic data, this often done by using a generative adversarial network (described in section 2.4.4). Additionally, neural networks can be used as classifiers. If a synthetic GPS trace is sufficiently realistic it will fool a classifier, therefore a classifier network can be used to evaluate the performance of the generation algorithms.

In a dense feed-forward neural network there are several layers of neurons. A neuron is a unit of computation in a neural network. Each neuron takes several inputs and contains a number of internal weights. The output value for a neuron is determined by multiplying each of the neuron's inputs by the corresponding weight. The sum of these weighted inputs is used as the input to the neuron's activation function, the result of this function is the output of the neuron.

The output of a neuron $N(x)$ with $n+1$ inputs $x_0...x_n$, corresponding weights $w_0...w_n$ and activation function f is defined in equation 1.

$$N(x) = f(\sum_{i=0}^{n} w_i x_i) \tag{1}$$

The output of the each neuron is used as the inputs to the neurons in the next layer (or are the network output in the final layer). The flow of inputs from one layer to the next is shown in figure 1.

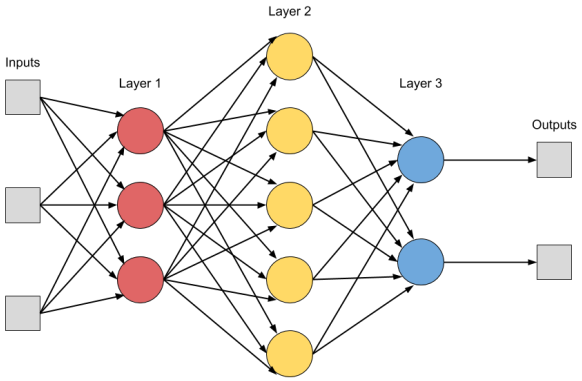

Figure 1: Example dense neural network with three layers. The arrows show how the output of each layer becomes the input to the next layer.

2.4.1 Activation Functions

There are several activation functions that are suitable in different situations.

The sigmoid activation functions are useful for the final layer of a network because they constrain the output into the range (0,1). This is useful because the training data is typically normalised to be in that same range.

The logistic sigmoid activation function is defined as [7]:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

The problem with sigmoid activation functions is that for very large and very small inputs it has a small gradient. The gradient of the activation function is used to update the weights of the neural network during back propagation. This causes training to take longer because the small gradients result in small changes in the weight values causing convergence to take more training step. This is called the vanishing gradient problem.

The ReLU activation function doesn't suffer from the vanishing gradient problem because the gradient has two constant values. The ReLU activation function is defined as [7]:

$$ReLU(x) = max(0, x) \tag{3}$$

The problem with the ReLU activation function is that for value below 0 the gradient is 0. Since the weights of the neurons are updated based on the gradient of the activation function. This means that if the gradient is 0 the weights will become stuck since any update will be multiplied by 0. Leaky ReLU is a modified version of the ReLU function that sets a a constant gradient α when the the input is less than 0. The equation for the Leaky ReLU activation function is [27]:

$$LeakyReLU = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases} \tag{4}$$

The sigmoid activation function tends to be used for the final layer of the network to produce the output that is constrained to be within the range $(0, 1)$. However to mitigate the effects of the vanishing gradient problem the ReLU or LeakyReLU activation functions are typically used for all of the earlier layers.

2.4.2 Loss Functions

Typically a neural network is initialised with random weights. This results in the output of the network being a random weighting of the inputs. For supervised learning problems neural networks are trained by comparing the network output to the known correct value for a specific input. A loss function is used to define a value for how 'wrong' the network is. The loss function typically takes the ground truth and neural network prediction as inputs and outputs a high value when there is a large difference between the true and predicted outputs and a low value when there is a small difference. Different loss functions are used for classification and regression networks.

The gradient of the loss function is used to update the weights of the neurons such that they will produce a lower loss value in future. This process is propagated backwards through the network using the gradients of the activation functions for each neuron in a process called "back propagation". The goal of network training is to reach the global minimum of the loss function such that the loss is as low as possible for all training inputs. Because the gradient of the loss function is used to calculate how to optimise the network the loss function must be differentiable.

Binary cross entropy or log loss is a loss function that can be used for binary classification tasks. When using a sigmoid activation function for the last layer the output of the neural network can be taken as the predicted probability that the network input is a given class. The values 0 and 1 are used to indicate that there is a 100 % chance that the input is of one of the two classes (consider class A is 0 and class B is 1). The value 0.8 would indicate that there is an 80% chance that the input is of class B and a 20% chance that the input is of class A. Therefore a value of 0.5 would indicate that there is equal chance the input is either class.

The binary cross entropy $H(t, p)$ for a given sample where p is the predicted label ($0 \leq p \leq 1$) and t (0 or 1) is the true label is [22]:

$$H(t, p) = -[t\ log(p) + (1 - t)log(1 - p)] \tag{5}$$

Therefore when $t = 1$ the binary cross entropy can be simplified to:

$$H(1, p) = -log(p) \tag{6}$$

Conversely when $t = 0$ the binary cross entropy can be simplified to:

$$H(0, p) = -log(1 - p) \tag{7}$$

The predicted probability that the input is of class 1 is p. Conversely the predicted probability that the input is of class 0 is $1 - p$. For example if $p = 0.2$ this indicates that there is a predicted 20% chance that the input is of class 1 and therefore an 80% chance the input is of class 0. Therefore the cross entropy can be thought of as the log of the predicted probability of the true class.

The log of the p or $1 - p$ term means that the loss will become exponentially larger the closer this term is to 0. This creates a large penalty for extremely inaccurate predictions.

2.4.3 Convolutional Neural Networks

A convolutional neural network works on a similar principle. However instead of using simple neurons in every layer some layers use convolutions. Convolutions are commonly used in image processing. The output of a convolution is the combination of a convolution kernel and the input. A convolution kernel is a matrix of weights. The value of each pixel in the output image is the sum of pixels around the corresponding input pixel multiplied by the corresponding weight in the kernel. The kernel is stepped over the input image to find the value for each pixel in the output.

In figure 2 the output value x is the sum of the highlighted values in the input image multiplied by the corresponding value in the kernel. Colours have been used to show which image values correspond to which kernel value given several output pixels. This is repeated for all pixels in the output. There are several solutions to handling when the kernel exceeds the edge of the input image such as padding with 0s or excluding these pixels which would cause this from the output.

A convolutional neural network learns the weights of the convolution kernels the same way a dense neural network learns the weights of its neurons. This has advantages over treating each pixel as its own input (as it would be in a standard neural network) as the convolution is not location specific. For example if the neural network learns that one pattern of pixels indicates a certain feature, in a standard network this would have to be learned separately for each position in the image but in a convolution neural network the convolution will learn this and that knowledge is automatically transferred to every position in the image. [23]

A deconvolution is the inverse operation of a convolution. A transpose convolutional layer uses deconvolutions instead of convolutions gan-fig. Convolutional layers are typically used in classifier networks when scaling down the input from a large input (such as an image) to a small output. Convolutional transpose layers are typically used in generator networks when taking a small input and scaling it up into a large output (such as an image).

GPS traces can be encoded in a similar way as images. Digital images are commonly represented by a 2D matrix of pixel values. If colour values are included a 3rd axis will

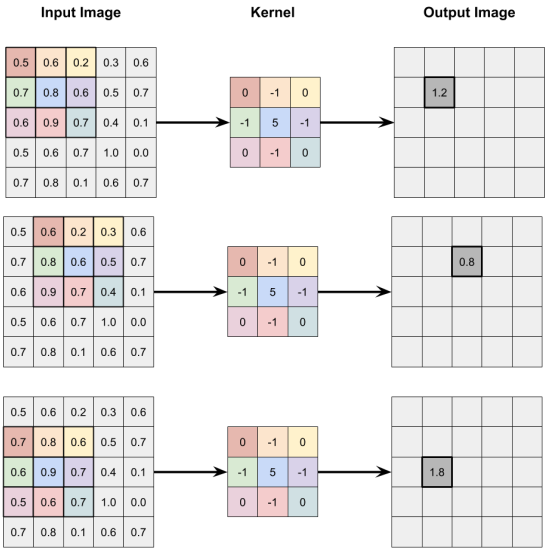

Figure 2: Diagram of an image convolution. Each output value is computed as the sum of multiplying the pixel values in the input image by the corresponding value in the kernel (the corresponding values are highlighted in the same colour).

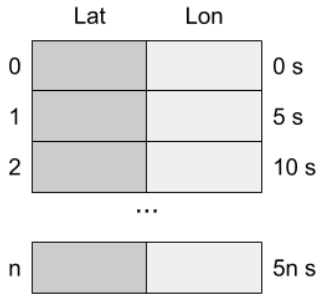

Figure 3: A GPS trace represented as a list of pairs of latitude and longitude values. The trace is a $2 \times n$ 2D matrix where n is the number of samples. Samples are taken at a consistent time interval (5s) therefore the position of the sample in the matrix indicates the sample time.

typically be used to store the colour channels. In this case each pixel can be considered a n-tuple where n is the number of colour channels. The values in this tuple are not spatially related as they are discrete colour channel values.

A GPS trace is a list of pairs containing latitude and longitude. Assuming that samples are taken at a consistent time interval this makes the trace a $2 \times n$ 2D matrix where n is the number of samples, the position of the sample in the matrix indicates the sample time. The values in each tuple are not spatially related, however the matrix of samples is temporally related. A convolution can be applied to a temporal axis the same way it can to a spatial axis.

Convolutional neural networks typically use 2D convolutions because they are used on images which are a 2D data structure. However 1 dimensional convolutions can also be used on 1 dimensional data structures such as GPS traces [7]. 1D convolutional layers are

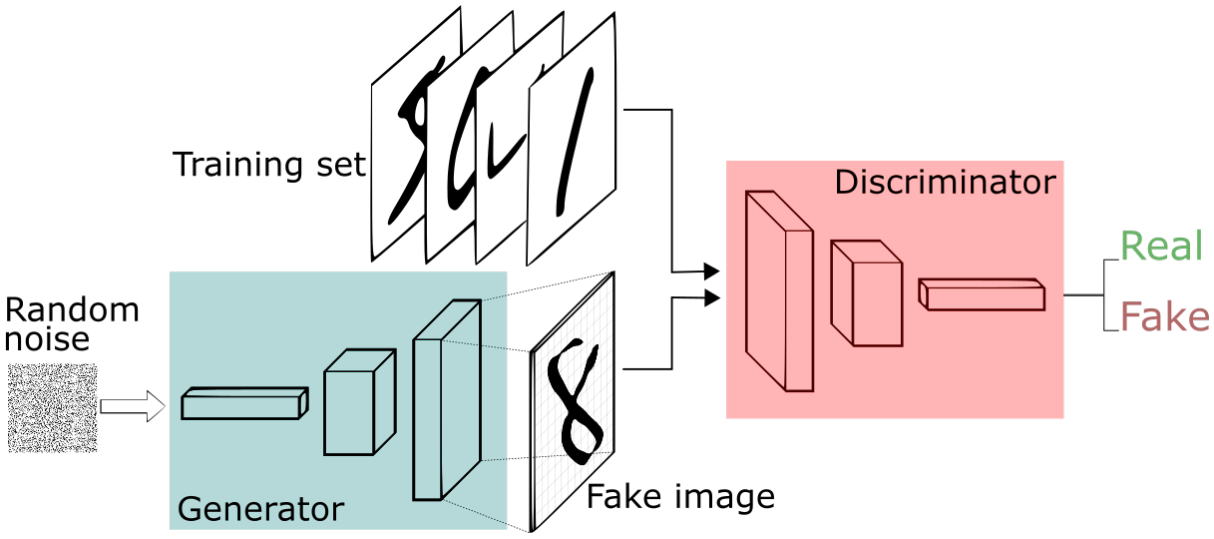

Figure 4: Diagram showing how a GAN is structured [10]. The discriminator classifies its inputs as either real or fake. The generator generates fake images and attempts to fool the discriminator.

used because the trace is a series of data points with 2 channels (latitude and longitude) sampled over a single dimension (time). The spatial invariance of convolutional neural nets is an advantage in image processing because the relevant features of the classification may be in any part of the image. Similarly useful features of GPS traces won't always occur at the same time point because important events (e.g. turning a corner) will occur at different times due to the route taken, traffic and other factors. However, because the GPS data has been encoded with the time series as a spatial axis convolutional neural networks provide temporal invariance.

Convolutional networks require their inputs to be a specific size so GPS traces that are too short can be padded with zeros and those that are to long can be truncated. Instead of using the direct GPS values as input to the neural network a pre-processing step can be used to calculate the properties of the movement at that time point. For example calculating speed, acceleration etc. These can then form the input to the convolutional neural network [8]. However, the network should be able to learn to calculate the useful aspects of the trace directly if it is sufficiently complex and sufficient training is completed.

2.4.4 Generative Adversarial Neural Networks

In a Generative Adversarial Neural Network (GAN) architecture two neural networks work adversarially against each other. One neural network is called the discriminator, this is a classifier network that takes a trace as input and outputs whether that trace is really from the data-set or is synthetic. The loss function for the discriminator is typically a suitable loss function for a binary classification task such as binary cross entropy . The other network is called the generator, this network generates GPS traces from its input. The generators loss function is set to be the "confusion" of the discriminator, that is the generator has low loss when the discriminator mistakes a generated trace for a real one. This is typically implemented as the binary cross entropy between the discriminators

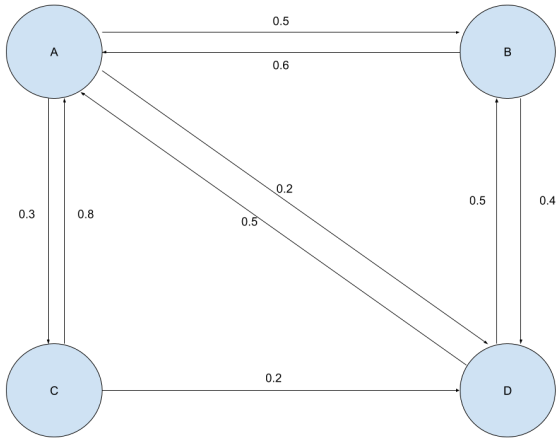

Figure 5: Example Markov model showing transitions and probabilities between 4 states. Transitions with a probability of 0 are excluded.

output for generated traces and the 'real' label. As the two networks train they will improve each other resulting in high quality generated GPS traces.[12]

This approach has previously been used to augment existing data-sets in order to improve performance for transport mode classification tasks [25]. By generating new synthetic data the data set could be augmented so that all transport modes we equally prevalent without reducing the amount of data available. This makes the network less likely to learn a bias towards the most common form of transport.

2.5 Markov Models

A Markov model is a set of states and transitions. Each transition has probabilities such that the sum of the transition probabilities for all transitions leaving a given state sum to one. An example Markov model is shown in figure 5. You can model a GPS trace as a series of transitions between the nodes on a road network. Research has been conducted into the possibility of using Markov models to predict the destination of user from their current location and a section of their GPS trace [1] [28].

In figure 5 the probability of transitioning between state 'A' and state 'B' is 50%. At each time step the model randomly performs a transition from its current state through one of the available transitions weighted by the probabilities. For example if the current state was 'A' then at the next time step there would be a 50% probability of transitioning to 'B', a 30% probability of transitioning to 'C' and a 20% probability of transitioning to 'D'. Only the current state of the model has any effect on the probability of transitioning to the next state. This is called the Markov property. [3]

It is possible to randomly walk through the states in a Markov model in order to simulate the system it represents. For example for the Markov model shown in figure 5 might transition through the following states (after 5 time steps): A, B, A, C, D, A. The states and probabilities of the Markov model can be setup to emulate a real world process. Using this model will then produce results similar to those found in the real world process.

The Markov chain can be represented as a series of states and transitions. The probability of transitions can be represented by a square matrix of transition probabilities where the probability of the transition $i \to j$ can be found by looking up the row i and column j in the matrix.

3 Data Set

This project uses a data set of real GPS traces. The data set used for this is the Microsoft GeoLife data set [43]. Additionally, some of the algorithms use map data for the region covered by the GPS traces as input. Map data was downloaded from OpenStreetMap [6] and converted into representations that could be used by the algorithms more easily.

In order to fulfill the aims of this project a number of GPS trace generating algorithms were implemented. These algorithms required the raw data to be standardised and normalised so a number of pre-processing steps were used (these are described in section 3.2.2. Furthermore, in order to process the GPS data in terms of the map model the traces had to be converted into a series of map nodes instead of GPS coordinates (this is described in section 3.1.2).

3.1 Map Data

In order to generate GPS traces that follow roads and paths public available map data was used. The map data describes a graph that represents the real world road network. The nodes and edges of this graph are used to construct the Markov model used by the Markov model based algorithms (as discussed in section 4).

3.1.1 Data Format

The map data as downloaded from OpenStreetMap [6] is provided in XML format. The data is structured as a series of 'nodes' and 'ways'. Each node has a corresponding location and each way is a series of nodes that form a map feature. The 'highway' label is used to indicate that a way describes a road or path. Not all nodes correspond to roads and paths so any node that is not part of a 'highway' way is excluded. Additionally, each 'highway' way contains metadata indicating the type of road it is. This data is recorded in order to classify which type of road each node is a part of. An example XML file from the OpenStreetMap wiki showing the structure of the map data as provided by OpenStreetMap is provided in appendix A.

The majority of the data from the GPS data set occurs in Beijing (as discussed in section 3.2). For this reason the section of the OpenStreetMap data that was exported covers Beijing. Specifically the data exported was the map elements with latitude in the range $[39.438362, 40.303177]$ and longitude in the range $[115.791885, 116.87722]$.

3.1.2 Pre-processing

The Open Street map data is provided as an XML file. The built-in python XML parser is used to parse the XML document. The aim of the map loading notebook is to convert this map data into a graph. In order to do this the edges between all nodes are represented

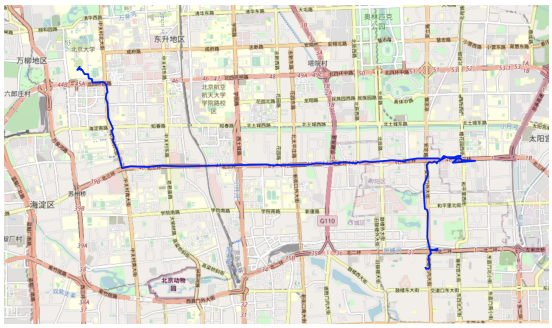

Figure 6: Example original real trace

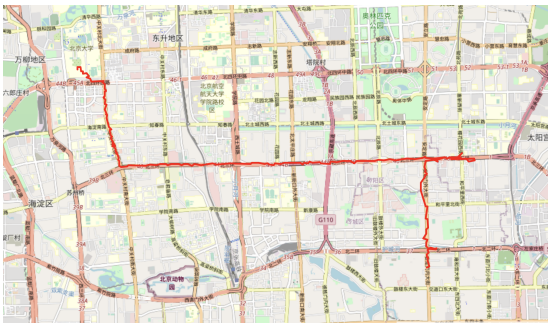

Figure 7: Real trace snapped to the map

as pairs of origin and destination nodes. Additionally the position of each node must be recorded so that node IDs can be converted back to real locations later.

In the XML data each node has a unique numerical ID however to simplify indexing the data structures used in this project, the nodes are given new IDs based on the order they are found in the XML data. This allows the node ID to be used as an index into an array to look up its position. While loading the data a python dictionary is used to keep track of how original Open Street Map node IDs map to the node indices used by this project. A 2D array is created of shape where each entry is the latitude and longitude of the node for each index.

Each node can have any arbitrary number of edges between it and neighbouring nodes. Therefore a ragged array is required to store these edges efficiently. Ragged arrays are not supported by numpy natively so the edges are stored as an array with , this allows the value for each index in the array to be another numpy array. Because each nested array is its own object it can have its own length.

In order to analyse the probability of each possible transition the real traces from the GPS data set have to be converted to a series of map nodes instead of GPS coordinates. In order to do this the index of the nearest node to the GPS sample must be found. In order to do this quickly FLANN is used. This library automatically selects the most appropriate nearest neighbours algorithm (and parameters) for the provided data set. It uses Euclidean distance between the points to compute the distance. Since our coordinates are GPS points which are not Euclidean this is not strictly accurate. However Euclidean distance is a reasonable approximation of the true distance over a small area (such as when only considering traces withing Beijing). The true distance between two GPS coordinates can be calculated using the Haversine formula [32], this is not used to find the nearest node because it is not implemented in the py-flann library. Figure 6 shows an unmodified real GPS trace and figure 7 shows that same trace after it has been snapped to the nodes on the map. This shows that there is little accuracy lost in this process.

3.2 GPS Data

In order to train the machine learning algorithms described in this project a data set of real GPS traces is required. The data set used for this is the Microsoft GeoLife data set [43]. This data set contains GPS traces collected from 182 users over the course of 5

years. The data was collected in a variety of cities in China (and some cities in Europe and America). The majority of the traces were captured in Beijing, China.

3.2.1 Data Format

The Geo Life data set is comprised of 182 folders (each corresponding to a user) containing CSV files containing individual GPS traces. Each trace file contains 6 header lines (which are discarded as they are not relevant to the project). Each subsequent line is a GPS sample in CSV format containing: latitude, longitude, a field that is always 0, altitude (in feet), sample timestamp (as number of days since 12/30/1899), date string and time string. The majority of the traces are logged in a dense format with a sample every 1-5s.

3.2.2 Pre-processing

The pandas libaray was used to load the data set CSV files into data frames. A data frame is a 2D data structure that provides implementations of a variety of data processing functions. The date and time fields from the CSV data where merged into a single 'timestamp' date time field. Then all fields other than the latitude, longitude, altitude and time stamp where dropped since these are the only fields we are interested in (altitude was never used but was kept because it may have been useful). Since the traces are a series of GPS samples at specific timestamps the data frame was re-indexed with the timestamp as the index.

In the raw data set each data point has an associated timestamp. However for this project the absolute time of the sample is not relevant, but the time step between samples is because the combination of the time interval between samples and the positions of those samples encodes the speed of the GPS device. In most of the data the time samples occurred at a frequency of about 5s per sample. Rather than preserving the timestamp for each sample it is simpler to standardise the time interval between each sample. Therefore each trace in the raw data set was re-sampled so that each sample occurred exactly 5s apart.

In convolutional neural networks a convolution is applied to the inputs of each layer by moving the filter across the inputs. Re-sampling the GPS traces to have a consistent interval between samples means the steps made by the convolution have real meaning in the data set. Since moving the kernel one step now always corresponds to moving through time by the same interval. If re-sampling was not performed the step of the convolution would have no real meaning. Having the steps of the convolution kernel correspond to a constant time interval allows meaningful relationships between samples in the trace to be learned by the convolutional neural network. This allows the network to learn to identify features in the trace across both space and time.

In order to ensure that this re-sampling process didn't significantly decrease the quality of the data set a comparison of all interpolation methods provided by the pandas library was conducted. Sections of the raw data were deliberately deleted, the interpolation algorithms were then run over the remaining data. The R^2 score between the re-interpolated

Method	R^2	**Time (s)**
akima	0.991852	1.315225
pchip	0.991648	1.374451
slinear	0.991147	1.144500
time	0.991147	0.701550
index	0.991147	0.743905
piecewise_polynomial	0.991147	5.757356
from_derivatives	0.991147	5.774293
linear	0.990089	0.693300
nearest	0.986766	1.014892
quadratic	0.984276	1.329257
cubicspline	0.977586	1.287085
pad	0.969900	0.332779
zero	0.969900	1.067536

Table 1: Interpolation experiment results. Method is one of the available pandas interpolation methods. R^2 is a measure of how accurate the reconstructed trace is to the ground truth (a perfect reconstruction would have a score of 1). Time is the number of seconds interpolation took to compute (each test was run over the same subset of the GPS data set)

trace and ground truth was calculated. It was found that the 'pchip' and 'akima' interpolation methods were the most accurate. However the 'time' interpolation method was significantly faster and has similar performance. Therefore, the 'time' interpolation method was used for the purpose of re-sampling the data. The full results of the experiment can be seen in table 1. Testing was performed on a small subsection of the data set. When applied in practice interpolation was applied to a much larger number of traces.

In order to create a consistent time interval a time index with a consistent interval was created, this was then merged with the original data frame. This results in a data frame where there are rows with values at whichever times happened to have samples in the original data set and empty rows at consistent time intervals. Pandas is used to interpolate the values in the original data set in order to fill in these missing values. The 'time' interpolation method was used. The resulting data frame has rows at consistent time intervals but with additional rows for the timestamps of the original samples from the raw data set. Finally, the index with a consistent interval is used to select the matching rows in the data frame. This results in a data frame which only contains samples at the specified interval. This process is shown in figure 9.

Figure 8 shows how temporal interpolation is applied. The original trace has a number of samples with measured time stamps (the squares) and a consistent interval index is made (the 'x' marks). Interpolation is used to estimate the position that the GPS device would have been in if a sample had been taken at each 'x' mark.

Originally, the pandas function was used to create a data frame with a consistent time interval. This function discards any entries that do not fall on the specified time interval. Interpolation would then be used to fill in any missing values. This means that some of the data points were deleted before interpolation was performed. Therefore there were less

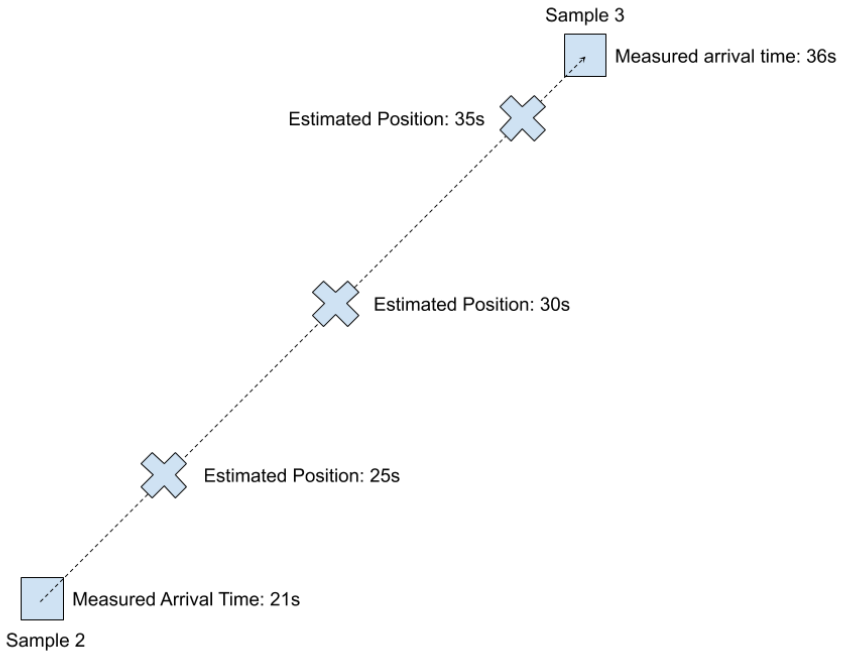

Figure 8: Example of temporal interpolation between two positions. The squares indicate two original GPS samples and their corresponding sample times. Interpolation is used to find the estimated GPS positions for the points marked with 'x', which have a consistent time interval of 5s.

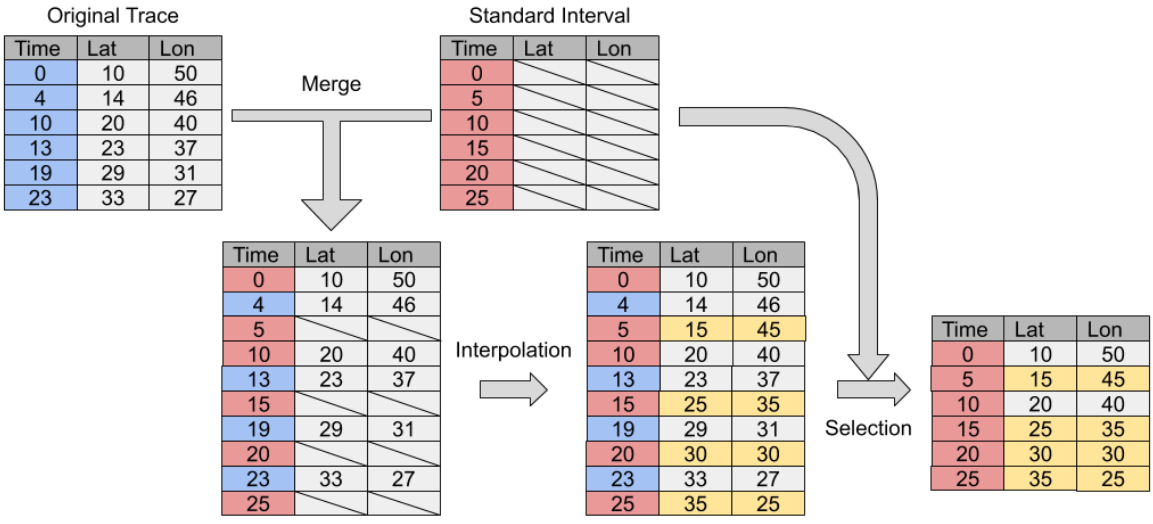

Figure 9: Process of how interpolating to a standard interval is implemented. The original trace (blue) is merged with an empty data frame with a standardised time interval (red). Interpolation is used to fill in the missing values (shown in yellow). Finally the standardised time interval is used to select the relevant coordinates. The latitude and longitude values used in this example are not realistic and are chosen only for illustrative purposes.

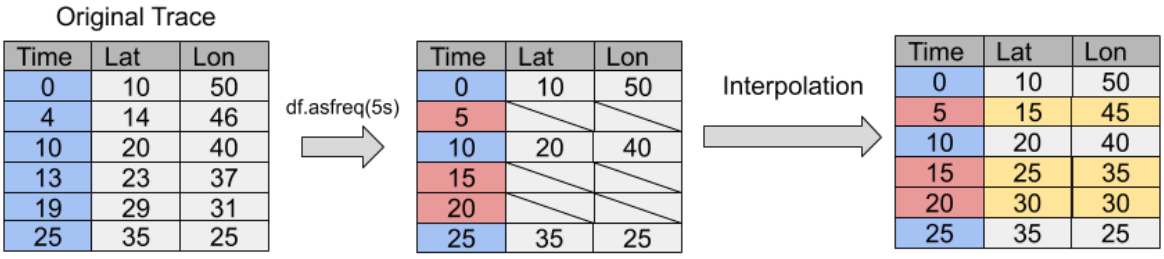

Figure 10: Obsolete implementation of temporal interpolation using the function. Samples that don't fall on the specified interval are discarded. Interpolation is used to fill in missing values.

sample points to interpolate the missing values from. This would decrease the accuracy of the interpolated data. This process is shown in figure 10. The method described above is more complex and takes longer to compute but ensures that none of the real data samples are discarded before interpolation occurs maximising the accuracy of the interpolated data.

Additionally, most neural networks require that the inputs be of consistent size. This is because the number of inputs to a dense layer of neurons is fixed and therefore the input to the network must be fixed. However, in the raw data set the traces are all of varying lengths. Therefore the traces were split into segments of consistent size. Each trace was split into segments of 200 samples each.

This process was done concurrently with the temporal re-sampling. When traces were re-sampled a limit was placed on the number of points that could be created by interpolation. This was done so that if there is a large time gap in a trace there is not a large amount of interpolated data which is likely to be inaccurate. When a trace cannot be interpolated because of this limit any effected segments are discarded to ensure that the data is kept clean. Additionally, any remaining samples that are not sufficient to create a whole segment are discarded.

Therefore the larger the segment length the more data gets discarded due to traces not being exactly divisible by the segment length. The length 200 was chosen because it is long enough to show some interesting features in the trace (e.g. going around corners etc.) while minimising the amount of data that is discarded.

Each segment is then exported to a numpy array containing latitude, longitude and altitude (time stamp is implied as each sample is now a consistent interval apart). The segment arrays are then stacked into a single 3 dimensional array of shape .

The majority of the traces in the data set are in Beijing. So the map data that was selected for this project only covers Beijing. Since only traces that are contained within the bounds of the map can be used for training the map based algorithms any trace segments that exceed the bounds of the map were discarded.

Typically the input to neural networks are normalised so that each dimension has the same contribution to the input. In this case latitude has a range of -90°(at the south

pole) to 90°(at the north pole) whereas longitude has a range of -180°to 180°. The output of neural networks is typically confined to a standard range due to the range of the activation function of the output layer. In this project 2 types of neural networks are used: a generator and classifiers. The generator will output a GPS trace and the classifiers take a GPS trace as input. It is therefore reasonable for the inputs and outputs of these networks to specify coordinates in the same format.

Initially, the neural network based algorithms were setup so that the range -1 to 1 spanned all possible GPS coordinates on the planet (i.e. the coordinate (45°, -45°) would be normalised as (0.5, -0.25)). However this meant that there was only training data for a very small area of the coordinate space.

Another normalisation approach was to re-scale each trace based on the distance from the 1st point in the trace. Under this normalisation strategy each coordinate in a trace had the position of the 1st point subtracted from it. This transformed the coordinates to be relative to the starting point of the trace. These traces could then be normalised into the -1 to 1 range by dividing the coordinates by the maximum distance from the start point of the trace. This addressed the issue of traces only taking up a small proportion of the coordinate space as each trace now typically spanned a large proportion of the range. This encoding is also location independent, once a trace has been encoded like this it is not possible to tell where in the world it was actually recorded. However, removing the absolute position information has some downsides. Once this encoding is applied it is impossible to learn the absolute location of obstacles (such as buildings and bodies of water) because this information has been lost. Additionally, traces generated in this format cannot be plotted on a map in a meaningful way because they don't correspond to an absolute position.

This encoding method was not used in the final version because it is impossible to convert these traces back to absolute coordinates without choosing a starting position. It is also impossible for the neural networks to learn the absolute location of obstacles (such as buildings and bodies of water).

Since map data was only exported for Beijing another option for normalisation presented itself. The GPS coordinates were re-scaled so that the points 0 and 1 corresponded to the map bounds (i.e. the coordinate (0.5, 0.5) corresponds to the center of the map). This meant that the data set now covered the entire coordinate range (when only considering traces that occur within the map bounds) but preserved the absolute position information.

4 Markov Models

Since the map data for the relevant region is available it is possible to generate GPS traces by performing a walk over the graph of roads and paths. For this project a number of random walk based algorithms were created with various variations and improvements.

4.1 Design

The simplest version of generating synthetic GPS traces by using the map data is by randomly walking through the graph. A starting position can be chosen randomly, then at each time step the next node can be chosen by randomly selecting a transition to an adjacent node. However, this method of generating a synthetic trace is very limited as the probability of a real person going in any given direction is not uniform. For example if you are on a main road it is more likely that you will stay on that main road than it is you will turn off down a side street.

Therefore in order to generate more realistic synthetic GPS traces it would be beneficial to know the probability of transitioning from one node to another. If these probabilities are known then a Markov chain can be created. Traversing this Markov chain will produce a series of nodes on the map that constitute the synthetic trace. Converting this series of nodes into GPS coordinates is simple as the coordinates of each node can be looked up from a table of node positions.

In order to determine the probability of each transition the real data set is used. For each edge in the network a count is created of how often a transition across that edge is observed in the real data. These counts can then be converted into probabilities by dividing the number of observations for an edge by the total number of observations for each edge departing the same node. This creates a Markov model of how GPS devices move through the road network. This training process is shown in listing 1 in section 4.2. Similarly the number of times each node is the first node in a trace can be counted. Dividing these counts by the total number of traces in the data set produces a series of probabilities that a given node is a starting position.

A more realistic GPS trace can then be generated by selecting transitions weighted by the probabilities found in the real data. The starting position of a trace can be chosen by selecting a node at random weighted by the observed probabilities that each node is a starting position. Then at each transition the next node can be selected based on the observed probability of each available transition. This model of GPS traces is a Markov model. The algorithm for generating traces is shown in listing 1 in section 4.2. A diagram show a small subset of the Markov model generated from the map of Beijing is shown in figure 11.

By weighting the selection of transitions based on how likely it is a that a human would perform the same transition the traces produced should be more realistic. However, since the decisions are random and not based on previous states of the walk, the model is likely to exhibit behaviour that is not common in real traces.

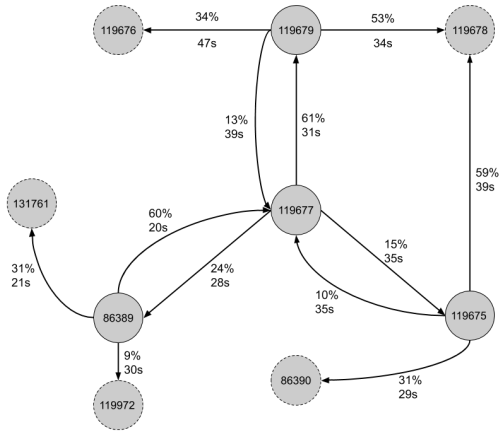

Figure 11: A small subset of the Markov model that was learned from the map data from Beijing. Each transition is labelled with the probability of following that transition (which is used to generate synthetic traces and the average traversal time for that transition (which is used for temporal interpolation).

In the map data set one way streets are labelled as one way. This information could be encoded directly into the graph of the road network by only allowing transitions in the direction of the one way street. This was implemented but not used for the testing of this project. The reason for this is that one way streets are only one way for cars and not pedestrians so it is not true that just because a street is labelled one way that it can only be traversed in one direction. Furthermore, the probabilities of transitions in the network are computed separately for each direction, so if a certain section of the map is truly only traversed in one direction then this information will be learned from the training data.

4.1.1 Backtracking Prevention

In most of the real GPS traces the trace moves in one direction and doesn't go back on itself. This property is not modelled in the Markov chain as only the current state (in this case map node) effects the probability of the transition. In order to prevent this kind of behaviour the previously visited nodes can be considered. To prevent back tracking the probabilities of transitions can be modified so that the probability of visiting a previously visited node is set to 0. Then the other probabilities must be re-scaled to sum to 1. Doing this violates the Markov property but may help to improve the realism of the generated traces.

4.1.2 Temporal Interpolation

The other major inaccuracy in the traces produced by this kind of model is that the trace instantly jumps from the position of one node to the next. In real GPS traces the transition occurs smoothly between the positions of the nodes.

Therefore in order to increase the accuracy of the generated traces it is necessary to smooth the transitions between nodes. This is done by interpolating between the node positions.

During the process of recording how often each transition occurs, the timestamps from the real data set are used to record how long each transition takes in the real world on average. This training process is shown in listing 1 in section 4.2. A diagram showing the average transition times between nodes is shown in figure 11.

In the temporal interpolation version of the algorithm an ongoing time is maintained. At the start node the time is set to 0. Each time a transition occurs from one node to another the average time that transition takes in the real data set is added to the ongoing time. This produces 2 data structures as output, a series of nodes and the time that an individual following this trace is expected to arrive at each node. The algorithm for this is shown in listing 2 in section 4.2. The data can then be re-sampled to have a consistent time interval and interpolation can be used to create a smooth transition between the node positions. Doing this makes the traces appear much more natural. The temporal interpolation is implemented using the same technique used to re-sample the traces to a consistent time interval in the pre-processing step (as described in section 3.2.2).

4.1.3 Transferring Models Between Cities

A limitation of these random walk techniques is that in order to compute the probability of node transitions a data set of real traces in the map area is required. This means if the aim was to generate synthetic traces in St Andrews it would be necessary to capture real world traces from users in St Andrews in order to calibrate the model. It would be beneficial if were possible to calibrate the model in one area and apply it to any area in the world.

In order to achieve this it is not possible to consider the probabilities of the likelihood of a transition from one specific map node to another specific map node because then data for that exact transition is required for training. Instead the training process must be generalised so that the model learns the probability of a transition from one type of map node to another type of map node.

In the map data each "way" is labelled as the class of road that it is. Using this information it is possible to label nodes as what class of road they are a part of. Once each node as been labelled as being a member of a road class it is then possible to use the real trace data to record the number of transitions between classes of nodes instead of between specific nodes. This produces a matrix of probabilities that a node of a given class will transition to a node of another (or the same) class.

There are 147,167 nodes in the Beijing map data. Simplifying the learning to classes of node reduces the number of possible transitions from 21,658,125,889 (for transitions from each node to every other node), to just 31 node classes resulting a transition matrix with 961 values. A heat map of this transition matrix is shown in figure 12.

Now when performing a random walk of the map a starting position can be selected randomly from the map. Then for each transition instead of looking up the probability of the transition for each edge originating at the current node the algorithm can instead look up the class of the neighbouring nodes. The probability of transitioning to each node can then be calculated based on the observed probability of transitioning from the class

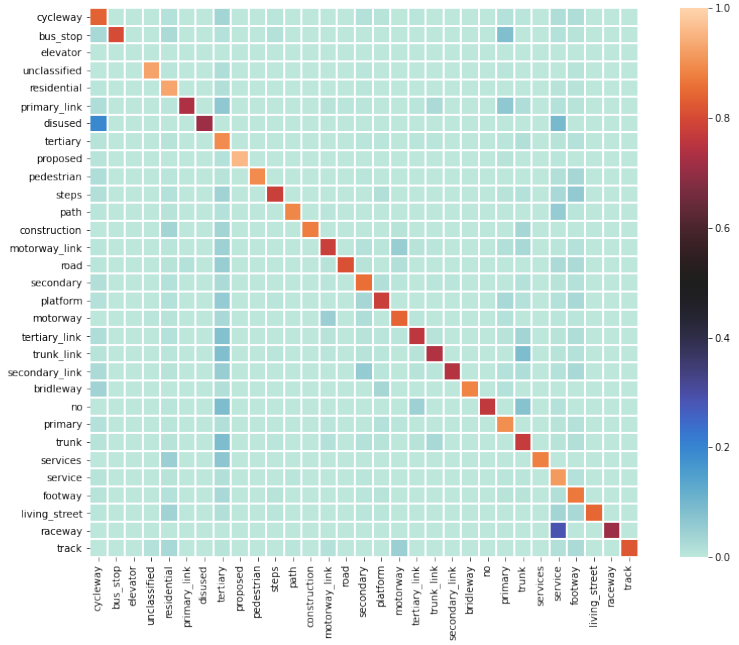

Figure 12: Transition matrix for node classes. The y axis specifies the origin class, the x axis specifies the destination class. For almost all classes the most likely transition is to is the current class.

of the current node to nodes of the classes of neighbouring nodes.

By doing this it is possible to learn the transition probabilities between types of nodes in one area and apply that knowledge to other areas of the world. Since it is simple to download the map data for other regions. However the average traversal time of the specific edge is still used for temporal interpolation. If this methodology was used to generate traces outside the original training region another method would be required to provide speed information for temporal interpolation. Figure 13 shows a trace that was generated by this algorithm by providing it the map data for St Andrews. In order to provide timestamps for temporal interpolation a consistent 10s interval was applied for all node transitions. Transition probabilities for node classes were learned from the original Beijing data set.

This project's implementation of this category-based random walk includes a weaker anti-backtracking measure. Rather than setting the probabilities of all previously visited nodes probabilities to 0, only the previous node is excluded. Doing this will prevent the model from immediately turning around but still allows it to follow the same nodes again if it loops around. This is necessary because when walking along a road the nodes in either direction will always be the same probability because they are of the same class. This means that the algorithm won't be able to use the prevalent direction of traversals along a road to naturally prevent back tracking. So this weaker backtracking prevention is implemented so to prevent large numbers of direction changes while following a road.

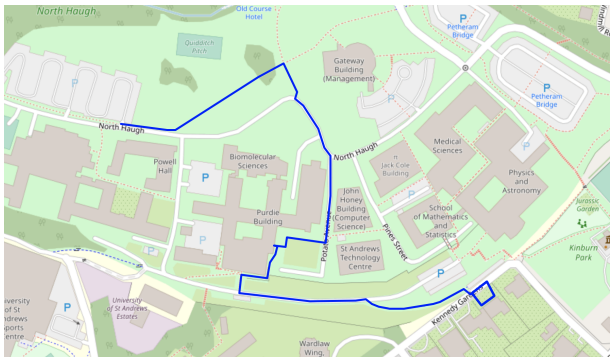

Figure 13: Randomly generated walk around University of St Andrews North Haugh. The model was trained using GPS and map data from Beijing and run using map data from St Andrews.

4.1.4 Preserving Speed and Distance

People use fitness apps to track their fitness activities such as running, walking and cycling. For these apps to be useful they need to be able to accurately report metrics based on the speed and distance of the user's activity. Currently this is done by recording the user's GPS data. This provides the fitness app with the absolute positions of the samples in the trace when this is not strictly needed to provide the service.

These algorithms use temporal interpolation to make the transition speed between nodes look believable. Previously it was set based on the observed transition time for the current edge. However, the transition speed can be set however we like. It is therefore possible to generate a synthetic GPS trace which has the exact same speed and distance travelled as a given target trace.

To do this instead of keeping track of the current time as nodes are traversed the cumulative distance can be tracked instead. Doing this gives a cumulative distance for each node as it is visited. Distance between two GPS coordinates is calculated using the Haversine formula [32]. Similarly the cumulative distance for the target trace can be computed. The distances at each node in the generated trace will not match the target distances. The cumulative distance is used because it is similar to the elapsed time in the trace. Essentially this provides points where it is known that "zm into the trace we'll will be at position (x, y)". In order to find the position the generated trace would be at for each target distance interpolation can be used to find the missing values. This produces a synthetic trace where the distance traversed (and therefore speed) at each interval matches the target trace. 'Index' interpolation is used instead of the 'time' method because the 'time' method can only be used with time based indices.

Fitness apps may also base some metrics on the altitude of the GPS trace. For simplicity this has not been modelled in this project. In a simple implementation the original altitude from the target trace could be passed through to the generated trace. However there is no guarantee that this would result in a reasonable or believable altitude for the generated position. Creating a model that could produce a similar pattern of altitudes to the original trace would be more complex because an area with similar changes in elevation would have to be found in order for the changes in order to be able to preserve

Listing 1: Pseudo-code showing how the observed probabilities and traversal times for each edge are calculated. The and data structures are initialised with 0s for every edge in the network. Logic handling edge cases related to observed transitions not existing in the map data is excluded for brevity.

Listing 2: Pseudo-code showing the algorithm for generating a trace. Where the function takes two arrays as arguments, the first is a set of options, the second is the probability that each option should be chosen. The function returns an element from the first array randomly weighted by the second array. is an array where each value is the probability that the node with that ID is a starting node, the sum of all values is 1. is an array where each element is a dictionary. The keys in each dictionary are the IDs of the nodes for which an edge exists, the value for each key is the probability that the edge will be taken. The sum of the values in each dictionary totals to 1. Finally is an array where the value of each index is the GPS coordinates for the corresponding node.

the altitude changes from the target trace in a believable way.

4.2 Implementation

When performing a random walk the measured probabilities of each transition are needed in order to determine the likelihood of taking a a given path. One way to do this would be to store the transition probabilities in a square matrix where the probability of the transition $i \rightarrow j$ can be found by looking up the row i and column j in the matrix. Any transition that isn't possible would be represented as a 0 in the transition matrix. This approach would work but requires a square matrix with size dependent on the number of nodes in the network. In the Bejing map data there are 147,167 nodes which would result in a 147,167 by 147,167 transition matrix totalling 21,658,125,889 values. However there are only 107,449 non-zero transitions in that matrix. Therefore the vast majority of this matrix would be 0 making it a very inefficient representation of the transition probabilities.

Instead transition probabilities are stored as an array containing dictionaries. Each dictionary contains the probabilities of transitions to neighbouring nodes but transitions that are not possible are not included. This way each dictionary only has to store the few relevant non-zero values. This significantly improves the memory efficiency of the representation. The downside of this representation is that it becomes harder to perform mathematical operations on the transition matrix. However, none of the proposed algorithms require these kinds of operations.

When generating a random walk at each node a new node can be chosen based on the probability of each available transition. The logic used to calulate the transition probabilities is shown in listing 1. The transition probabilities can be looked up from the transition probability data structure. This specifies a number of probability values. In order to select the next node with the correct distribution of probabilities the numpy function is used. This function takes 2 parameters: an array of options and an array containing the probability of each option (which must sum to 1). These 2 arrays can easily be derived

Listing 3: Pseudo-code showing how back tracking prevention is implemented. This code replaces line 12 in listing 3.

for the dictionary entry representing the transitions for the current node. The options are the dictionary's keys (the indices of the possible next nodes) and the probabilities are the values of the dictionary. The pseudo-code for generating random walks is shown in listing 2.

To implement back tracking prevention a copy of the relevant transition dictionary is made (this is done to prevent permanently altering the base dictionary). Any node that has been previously visited as part of this trace has its probability set to 0. This causes the sum of the remaining probabilities to no longer equal 1. In order to rectify this each probability is divided by the sum of the remaining probabilities. This re-scales the remaining probabilities to sum to 1. The psuedo-code for this process is shown in listing 3.

When temporal interpolation is enabled the expected arrival time for each node is computed. This is calculated by doing a cumulative sum of the average traversal time between each edge in the walk so far. This produces a series of nodes and corresponding expected arrival times. The nodes are then converted into GPS coordinates by looking up their position from the node positions array. This produces a trace with time stamps and positions however the interval between these samples is not consistent. To create a consistent sample interval the same temporal interpolation process is applied that was used to standardise the interval in the real traces. A temporal index with the desired consistent time interval is created, this index is merged with the base data frame. Interpolation is then used to fill in the missing values (in this implementation no limit to the maximum amount of interpolation is applied). Finally, the first 200 samples are taken to be the generated trace (the remained are discarded). Initially 200 transitions are generated, this causes a proportion of the transitions to be discarded (as each transition generally takes more than 5s). This is done to ensure that the there are always enough samples in the interpolated trace and it isn't too short since an error would occur if there were fewer than 200 samples in the trace.

When generating a trace with the same distance intervals as a given target trace distance interpolation is used. Distance interpolation is similar to temporal interpolation. For each node transition the cumulative sum of the total distance travelled in the trace so far is computed. The cumulative distance of the target trace can be computed similarly. The distance between two GPS coordinates is calculated using the Haversine formula [32]. This produces a generated trace where each sample has a GPS position and a distance into the trace. A pandas data frame is constructed with the cumulative distance as index and GPS positions as rows. Instead of creating an index with a consistent time interval the cumulative distance from the target trace is used. Merging this with the previous data frame creates a data frame with missing values for the target distances. Interpolation can then be used to fill in the missing values. The cumulative distances from the target trace can then be used to select the relevant values. This produces a generated trace with distance intervals that approximately match the original target trace but with different absolute position.

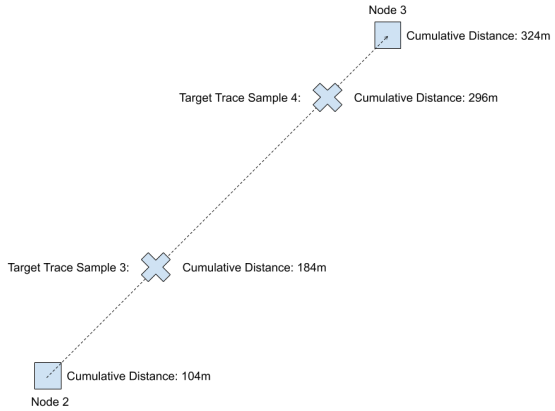

Figure 14: Example of distance interpolation. The squares indicate two nodes from the generated sequence of nodes. The cumulative distance of these positions is calculated using the Haversine formula [32]. Interpolation is used to find the GPS positions for the points marked with 'x', which have cumulative distance matching samples in the target trace.

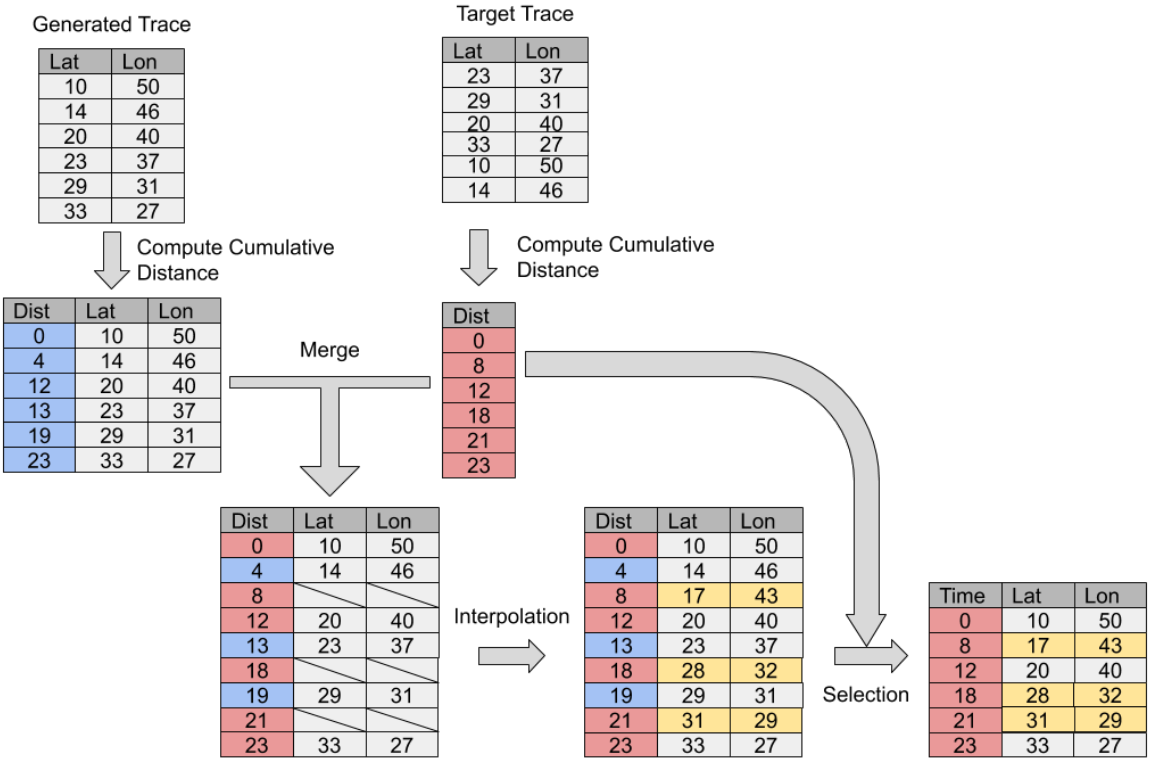

Figure 15: Diagram showing how distance interpolation is implemented. The cumulative distance of these positions is calculated using the Haversine formula [32]. The cumulative distance is then used as the index for interpolation. This produces an output trace where the distance travelled at each time interval approximately matches the target trace.

Figure 14 shows how distance interpolation works. The 'x' points are the cumulative distance of each sample in the target trace. Interpolation can be used to find the position along this trace that the target distance would occur, since the cumulative distances of the nodes that were selected for this trace are known.

5 Neural Network For Detecting Synthetic Traces

In order to evaluate the effectiveness of the trace generation by these algorithms a neural network based classifier was created. The more realistic the synthetic traces are the less accurate a classifier that attempts to identify real traces will be. Therefore a classifier can be used to evaluate the performance of this project.

The classifier is a 1D convolutional neural network. 1D convolutional layers are used because the trace is a series of data points with two channels (latitude and longitude) sampled over a single dimension (time). The 1D convolutional layers in this network allow it to learn to identify features in the traces in a way that is not specific to the position of the feature in the trace. Figure 16 shows the architecture of this network. The stride for all convolutional layers is 1, this causes the convolutions to have the same output size as their input.

The output of this network needs to be a single value indicating whether the input trace is a real trace from the data set or a synthetic trace generated by one of the algorithms developed for this project. Neural networks have numeric outputs so 0 was chosen to indicate that the neural network predicts the trace is synthetic and 1 was chosen to indicate that the trace is real. A sigmoid activation function is used for the final layer as this function has a range between 0 and 1. The network outputs a floating point number somewhere in that range, values closer to 0 or 1 indicate higher confidence and values close to 0.5 indicate low confidence.

Only a single output value is required so this neural network ends in a dense layer with a single neuron. The input to the network is a GPS trace with 200 samples with 2 values each (latitude and longitude). The first 3 layers of this network use 1D convolutional layers to find features of those traces in a temporally invariant way. Because the convolution kernels are stepped over their input they can learn patterns in the input in way that is not linked to a specific time. These layers use the 'same' padding method which pads the input with 0s so that the output size is not changed. The output of the convolutional layers is then flattened so that it can be input into the dense layers. These dense layers do

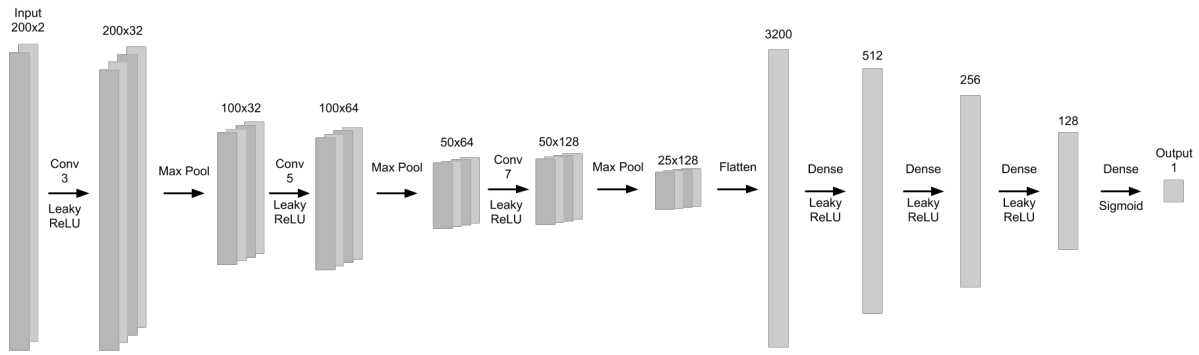

Figure 16: Classifier network architecture. Above each arrow the operation being performed is specified. Activation functions are shown below the arrow. The input and output shape of each layer is specified above each intermediary data structures.

not have this temporal in variance property. This means that they have specific weights for inputs at specific time steps of their input. Therefore if a certain feature is detected by the convolutions at one time step it may be treated differently than if detected at another. The combination of these layers allows the network to learn to identify important features in general throughout the trace and then separately combine them based on the time they occurred in order to produce a result.

The neural network has 3 convolutional layers and 4 dense layers. Each subsequent convolutional layer gets wider (there are more filters) and the kernel size gets larger. As the input goes through the network more types and more complex features are detected. So the layers get wider to allow more features to be based on the previous layer. The kernels get larger so that initially fine detail features can be found in the input but as the input passes through the network those detected features can be combined together to form more complex features that span larger areas of time.

The dense layers take the opposite pattern. Each layer is narrower than its input layer. This is so that the all of the complex features found by the convolutional layers can be gradually aggregated down into the single output value,

The activation of the last dense layer is a sigmoid function, this is done to constrict the output into 0 to 1 range. All of the other layers of the network use Leaky ReLU as their activation function. This is because other activation functions such as sigmoid suffer from the vanishing gradient problem. The vanishing gradient problem causes networks with many layers to be difficult to train because the gradients become small so many iterations are required for the neural network to converge. The ReLU activation function does not suffer from the vanishing gradient problem and therefore this issue is avoided.

The classification of whether a trace is real or synthetic is a binary classification problem (as there are only 2 classes). The loss function of this neural network is the binary cross entropy loss function. The classifier was trained in batches, at each step the classifier prediction for a batch of real traces and a batch of synthetic traces was computed. These predictions where then used to calculate the loss for the network and update the weights.

The network was trained in this way (rather than having 1 labelled data set) for two reasons. The first is that due to how the data is generated the real and synthetic data exists as two separate data sets so this method made the programming simpler. The second is that this training methodology closely mirrors how the discriminator in the GAN is trained so similar training logic could be used simplifying development.

Originally, it was intended that the various generator algorithms would generate traces at each training step. This would have meant that training over multiple epochs of real data wouldn't cause the generated data to repeat, which would help prevent against over-fitting and increase the variety of fake traces the classifier is trained on. However, the complexity of generating GPS traces with any of these algorithms was high enough to have a very significant impact on training speed. Additionally, generating traces on-the-fly like this would have required each variation of the generation algorithms to be duplicated or imported into the classification notebook. Generating synthetic trace data sets ahead of time increased training speeds and simplified the classifier implementation. When generating data sets of synthetic GPS traces the same number of synthetic traces

where generated as real traces in the GPS data set. This was done so that there would not be an imbalance between the two classes. If there were an imbalance then the neural network may develop a bias towards the class that is more common.

6 Generative Adversarial Network

In the Generative Adversarial Network (GAN) there are two neural networks that are in an adversarial relationship with each other. First is the generator which takes a vector of random noise as input and outputs a synthetic GPS trace. The second is the discriminator, which takes a GPS trace as input and outputs whether that trace is real or synthetic (this is similar to the classifier discussed in section 5). The loss of the discriminator is the binary cross entropy between the predicted output and the true label. The loss of the generator is the 'confusion' of the discriminator, this is calculated as the binary cross entropy between the 'real' label and the prediction of the discriminator for the synthetic traces. Therefore the generator is penalised when the discriminator correctly identifies generated traces as synthetic and rewarded when generated traces are mislabelled as real.

This setup creates a sort of arms race between the two networks. Initially both networks are bad at their respective tasks: the discriminator is essentially picking at random and the generator is producing nonsense. As the discriminator learns to distinguish the nonsense output from real traces it improves. In response the generator will learn to defeat the newly gained knowledge of the discriminator. As the training continues this process of each network getting the upper hand will repeat as both networks improve at their tasks. Once the training is complete the generator will be able to generate GPS traces that are extremely realistic because doing so is necessary to defeat the discriminator.

In order for the training process to work the generator and discriminator must remain evenly matched [7]. If this were not the case, say the discriminator is much better than the generator, then the generator cannot learn because it will never produce traces that start to fool the discriminator to learn from. The same goes for the reverse. It is therefore necessary when designing these networks to make sure that one network is not significantly more powerful than the other.

The discriminator used as part of the GAN has a similar design to the classifier used to classify random walks (as described in section 5). The stride for all convolutional layers is 1, this causes the convolutions to have the same output size as their input. These layers use the 'same' padding method which pads the input with 0s so that the output size is not changed. In order for the discriminator to be more matched with the generator is has been simplified so that it can train at a similar rate. The architecture of the discriminator is shown in figure 17.

The architecture of the generator is shown in figure 18. The generator uses transpose convolution layers instead of conventional convolution layers. The stride for these layers is set to 1 which causes them to behave the same as convolutional layers. Transpose layers were used because if the stride is increased they cause the input to be scaled up (rather than scaled down as in a standard convolutional layer), this is useful since the generator has a small input and creates a large output. This ability was not used in the final implementation but was useful during development. These layers use the 'same' padding method which pads the input with 0s so that the output size is not changed.

The GAN uses much larger convolution kernels than the classifier (section 5), this is because the wider the kernels are the more scope there is for the generator to learn rela-

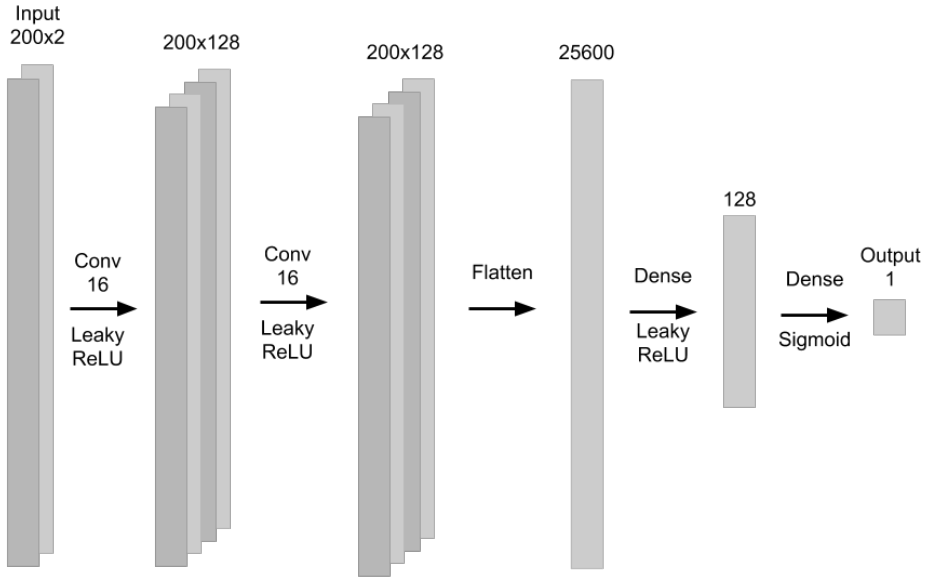

Figure 17: Discriminator network architecture. Above each arrow the operation being performed is specified. Activation functions are shown below the arrow. The input and output shape of each layer is specified above each intermediary data structures.

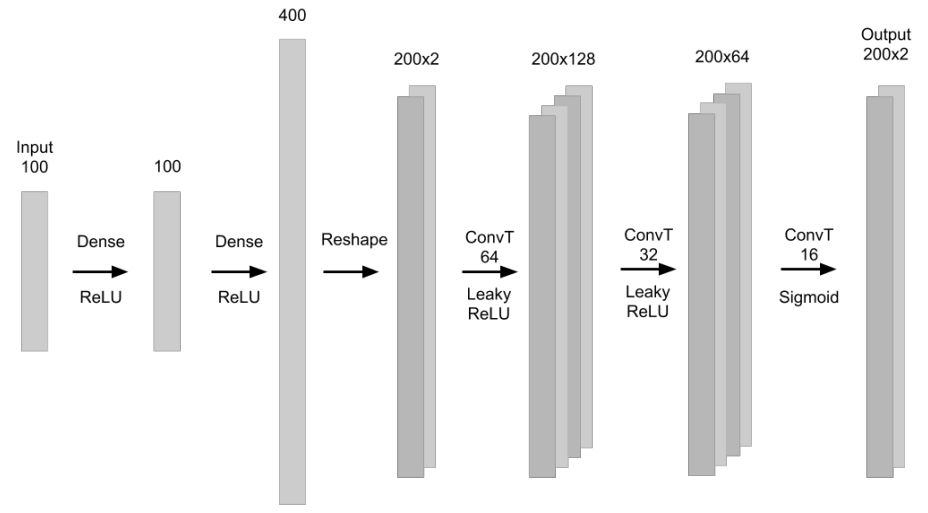

Figure 18: Generator network architecture. Above each arrow the operation being performed is specified. Activation functions are shown below the arrow. The input and output shape of each layer is specified above each intermediary data structures.

tionships between points that are distant in time. This increases the time needed to train the network (which is why it was not done for the classifier) but should improve the generators ability to make the trace realistic over longer time periods. The discriminator also has these wider kernels so that it is similar to the generator, otherwise the discriminator would be at a disadvantage.

6.1 Encoding Map Data

The main problem that the generator was having was that it does not know where the roads and obstacles on the map are. Therefore when the traces are plotted on a map it becomes very apparent that the trace is synthetic because the trace will not stick to roads and will pass through obstacles.

Ideally the network would learn to stay on the streets on its own but due to the large number of streets a large amount of training data and a lot of training time would be required. However, the map data for this area is available so it is possible to try to use that to improve performance. The aim is to have the traces stay in close proximity to the streets (which are the map edges) but it is much easier to design systems that stay close to the maps nodes (which are the junctions) which seems like a reasonable approximation.

In order to try and fix this problem the loss function was modified to try and give the generator more information about the positions of the roads by providing information about the node positions from the map. In order for the loss function to be able to be used it must be differentiable so that the gradient of the loss can be used to update the weights of the neurons in the network. One way of encoding this information into the loss function is to calculate the distance from each in the trace from the nearest node and penalise high distances. This would encourage the generator to stay close to the map nodes. However this loss function is not differentiable because it requires looking up the nearest node in the generated trace using a nearest neighbour algorithm and so cannot be used.

Another option is calculate the aggregate distance of a point from all nodes in the map area. However doing this creates a single point (at the average position of all the nodes) in the map with low loss that all traces will converge around. In order to solve this problem a function was created that rewards trace positions being close to map nodes. This would produce a sort of 'closeness' metric. To calculate this the distance between a generated GPS sample and all nodes in the map is calculated. The reciprocal of the distance is taken and raised to a large power. The larger the distance between the sample and the node the smaller this reciprocal will be. Raising this to a large power causes large distances to be penalised. This results in very low values for distant nodes and comparatively much higher values when close to a node. Summing these values causes distant nodes to barely contribute to the final value and close nodes to contribute more. Therefore, the total closeness will be low if distant from all nodes but high if close to a single node.

Closeness to a node is defined as where p is a large power and d is the distance to the node:

$$closeness = \left(\frac{1}{d}\right)^p \tag{8}$$

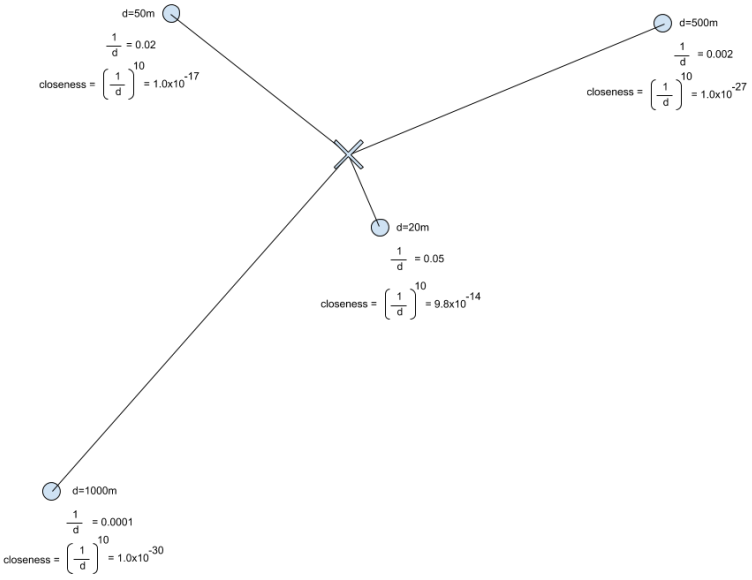

Figure 19: Diagram showing how closeness is orders of magnitude lower for nodes that are distant from the sample point. p is set to 10 for this example.

Taking the reciprocal of the total closeness will then produce a differentiable loss that is high when distant from all nodes, but low when close to any single node. This is shown in figure 20. However, due to the large number of nodes in the map network this loss function cannot be used due to impractical memory and time requirements involved in its computation.

Even if a function like this could be used it is still impractical because the network would still need to learn the positions of the individual nodes in the map. This would increase training time and fix the network to the specific area it was trained on. Additionally, it seems likely that the network would become stuck in a local minima by finding a small number of streets first due to chance and then learning to only output traces on that limited selection of streets.

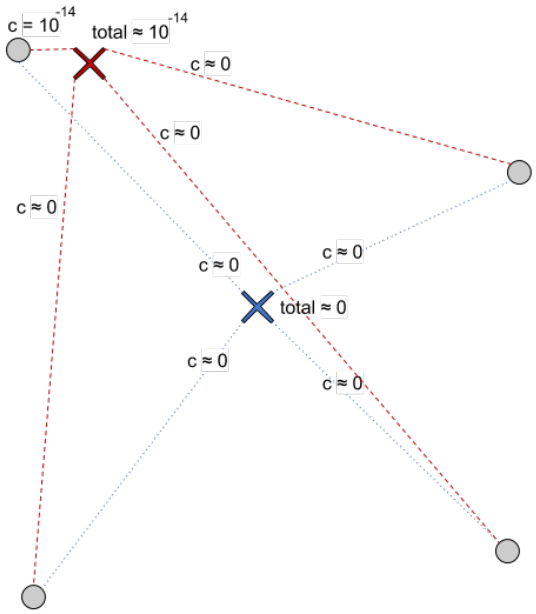

Figure 20: Diagram showing how the total closeness (c) is comparatively much higher if close to a single node. The blue 'x' is distant from all nodes, so for each node $c \approx 0$, therefore for the blue point is $total \approx 0$. Conversely because the red 'x' is close to a single node the closeness for that node is non-zero resulting in a non-zero total closeness.

7 Evaluation and Critical Appraisal

This project will be evaluated in three ways. There will be a qualitative discussion of the synthetic traces produced by the various algorithms in this project. A user study was conducted to determine how realistic the synthetic traces were to human participants. Finally, the neural network classifier described in section 5 will be used to evaluate how good the synthetic traces are at fooling computational techniques.

There are six algorithms and variations that will be discussed in this section (with their respective abbreviations):

- rand - Random walk with no modifications (as described in section 4)
- rand-no-back - Random walk with no back tracking (as described in section 4.1.1)
- rand-interp - Random walk with temporal interpolation (as described in section 4.1.2)
- rand-interp-no-back - Random walk with temporal interpolation (as described in sections 4.1.1 and 4.1.2)
- rand-cat - Node category based random walk (as described in section 4.1.3)
- gan - Generative adversarial network (as described in section 6)

Throughout this section the results will be discussed in relation to a several classification metrics, since the user study and the classifier are both classification tasks where either the neural network or participants were asked to label traces as either 'real' or 'fake'. The metrics used are 'precision', 'recall', 'F1 score' and 'accuracy'.

These metrics are defined in terms of true positives (tp), false positives (fp), true negatives (tn) and false negatives (fn).

Accuracy is the simplest of these metrics. Accuracy is the number of correct classifications over the total number of classifications. It is defined as [33]:

$$accuracy = \frac{tp + tn}{tp + fp + tn + fn} \tag{9}$$

Precision measures for a given label how many of samples given that label are truly members of that class. If the algorithm labelled a sample as class 'A' precision measures how likely it is that the sample is actually a member of class 'A'. Precision is calculated separately per category and is defined as [33]:

$$precision = \frac{tp}{tp + fp} \tag{10}$$

Recall measures what proportion of true members of a given class were correctly labelled as being a member of that class. If recall is 1 that indicates that all true members of the class were labelled as belonging to that class. It is trivial to have perfect recall for a specific class by labelling all samples as that class. For example if an algorithm labelled every sample as being a member of class 'A' there would be 100% recall for class 'A' (but there would likely be low precision). Recall is defined as [33]:

$$recall = \frac{tp}{tp + fn} \tag{11}$$

F1 score is the harmonic mean of the precision and recall metrics. It is defined as [33]:

$$f1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{12}$$

Accuracy is typically calculated for the whole result set. Precision, recall and F1 score are typically calculated for a specific class. When calculating precision and recall for a specific class the 'positive' event is the trace being labelled as the class in question (e.g. if calculating recall for the 'fake' class a 'false positive' is occurs when a real trace is labelled as 'fake'). When calculating accuracy it doesn't matter which class is taken to be the 'positive' case and which is taken to be the 'negative'.

7.1 Qualitative Results

Below a random example trace generated by each algorithm (and a real trace for reference) is shown.

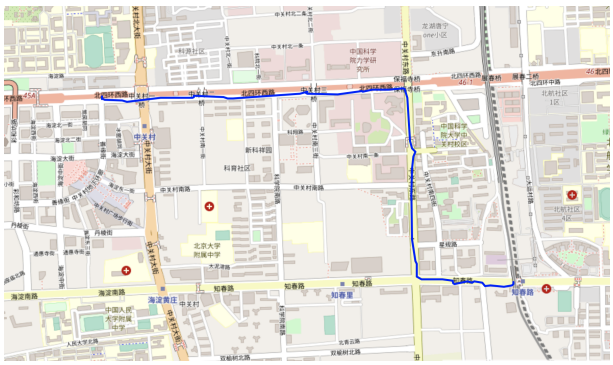

Figure 21: Example real trace

Figure 21 shows a real trace. In this trace the user generally followed the streets as they moved. Additionally, overall they headed in a single direction and did not go back on themselves. This is typical for real traces.

Figure 22: Trace generated by the rand algorithm

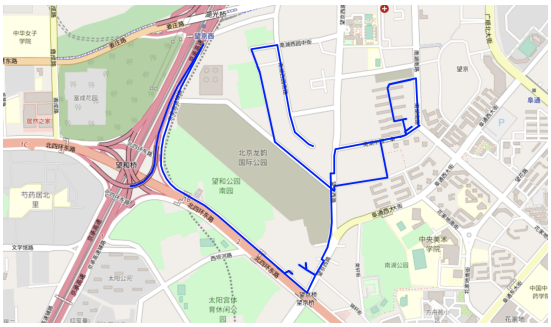

Figure 23: Trace generated by the rand-no-back algorithm

Figures 22 and 23 show generated traces without temporal interpolation. It is difficult to see on a still of the trace, but the trace is traversing entire streets in a single time interval (5s). Because of this these traces are also cover a large distance because they cover 200 node transitions and most node transitions take longer than 5s in the real traces.

Figure 23 shows the back tracking prevention. The trace now only goes directly back on itself when it is stuck at a dead end. This makes the trace look slightly more realistic.

Figure 24: Trace generated by the rand-interp algorithm

Figure 25: Trace generated by the rand-interp-no-back algorithm

Traces 24 and 25 demonstrate how applying temporal interpolation has smoothed out the traces. Additionally, the traversals between nodes are now taking multiple time steps. Figure 25 shows a trace is more believable because it is not going directly back on itself. However, this trace still doubles back on itself (just not on the same streets).

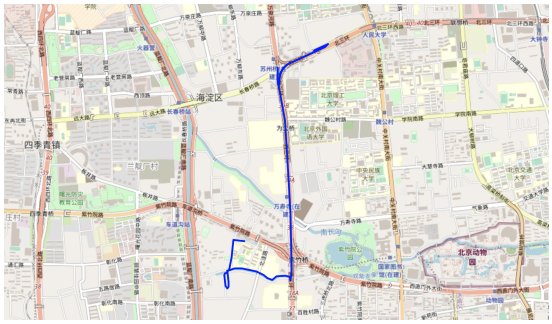

Figure 26: Trace generated by the rand-cat algorithm

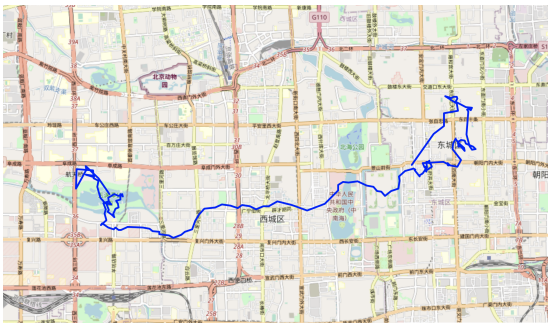

Figure 27: Trace generated by the GAN algorithm

Figure 26 shows how transitioning based on node category performs. This is useful for generalising learning to areas where there is no training data. This trace is comparable in realism to the rand-interp-no-back trace (figure 25).

Finally figure 27 shows an example trace generated by the GAN. The middle portion of the trace seems to have a very believable shape as it is relatively straight but isn't perfectly straight, which is similar to what is seen in the real traces. However the GAN produces traces that completely ignore the road network which makes them very unbelievable when plotted on a map.

7.2 User Study

A study was conducted to determine how convincing the traces generated by the algorithms created for this project are. Participants were shown animated GIFs showing a trace that was either real or fake. The traces were animated to give the participants a sense of the passage of time. Since the participants were not experts and this is not an activity they are likely to be familiar with the participants were shown three real example traces before beginning to give them an indication of what a real GPS trace looks like.

Participants were asked 50 questions. For each question an animated trace was displayed. The trace was either a real trace from the data set or a fake trace generated by one of the algorithms created for this project. For each trace the participants were asked whether they thought the given trace was real or fake. There were an equal number of fake traces as real ones to prevent there being a bias to either the real or fake option.

The fake traces consisted of 5 traces each from 5 variations of the algorithms from this project. There were therefore 25 real traces. The 5 algorithms were: rand, rand-interp, rand-interp-no-back, rand-cat and gan. The gan and rand-cat algorithms were selected because they are significantly different from the base random walk algorithm and are therefore included for comparison. The rand-interp and rand-interp-no-back algorithms are included to demonstrate how these modifications to the base random walk algorithm effect performance. Other possible variations of the algorithms were not included because doing so would result in there being fewer data points per algorithm and these algorithms are representative of each enhancement that was made. The number of questions had to

be limited since the ethical approval for artefact evaluation specifies that surveys will be less than 15 minutes and longer surveys increase the likelihood that a participant will not complete and submit the survey.

The questions were displayed in a random order to prevent traces of the same class from being shown in order.

In total there were 17 responses to the survey. Unfortunately due to human error for the first ten responses two of the questions were displaying the wrong trace. The traces they were showing were duplicated from another question of a different category so these data points were not valid. Therefore all of the responses that were recorded before a correction was made have had these data points removed. This means that for the gan and rand-interp-no-back algorithms there are 75 valid responses instead of the expected 85. Participants were permitted to exclude any question they wished, however no participant excluded any question.

Table 2 shows the precision, recall and F1 Score for all user responses. These results cannot be broken down by algorithm because users were only asked whether the trace was 'real' or 'fake' so these metrics can only be calculated for those classes.

	Precision	Recall	F1 Score
Real	64.13%	67.29%	65.67%
Fake	63.80%	60.49%	62.10%

Table 2: Precision, recall and F1 Score for all user study responses broken down by category

Table 3 summarises the responses from the user study. The 'Correct' row indicates how many responses for questions in that category were correct (i.e. the user selected 'fake' for a trace that is actually fake). The 'Incorrect' row indicates how many questions in that category were incorrect (i.e. the user selected 'real' for a trace that is actually fake). The 'Num Valid' row indicates how many valid responses were recorded for that category. The 'Recall' row indicates the proportion of the responses where the participants correctly identified whether the trace was real or fake. The 'Fooled' row indicates the proportion of responses in that category where the participant gave the incorrect answer, i.e. they were fooled by the algorithm. Because users were only asked to label traces as real or fake precision can only be calculated for the entire result set and cannot be broken down by algorithm.

	real	rand	gan	rand-interp	rand-cat	rand-interp-no-back	Total
Correct	286	52	63	45	42	43	531
Incorrect	139	33	12	40	43	32	299
Num Valid	425	85	75	85	85	75	830
Recall	67.29%	61.18%	84.00%	52.94%	49.41%	57.33%	63.98%
Fooled		38.82%	16.00%	47.06%	50.59%	42.67%	

Table 3: User responses broken down by algorithm

Participants are 67.29% accurate at recalling real traces. This is quite low (randomly

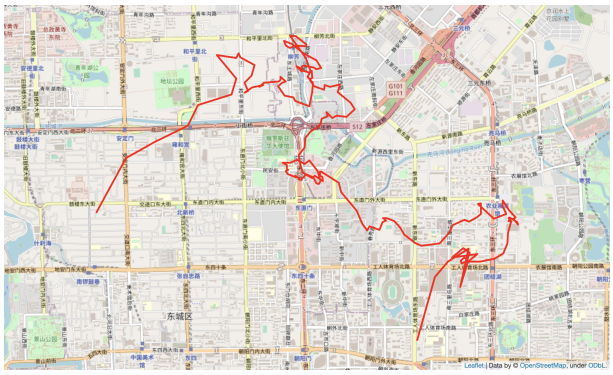

Figure 28: Example GAN trace from Study

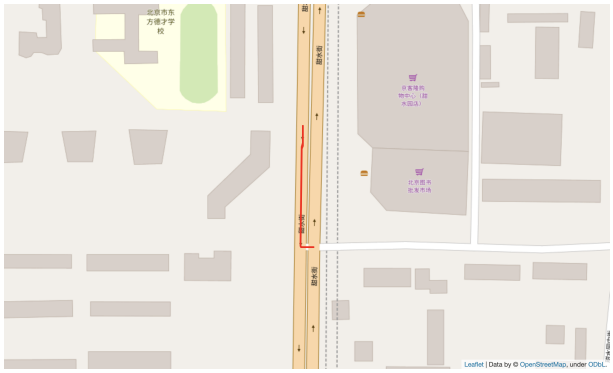

Figure 29: rand-interp trace 0

guessing would result in 50% accuracy) which is likely due to the lack of experience participants had at this task. Therefore in order for an trace generating algorithm to be considered as convincing as real it should have a 'fool rate' of at least 67.29%. None of the algorithms tested in this project reach this goal. However, the rand-interp and rand-cat algorithms have a ∼50% fool rate. This indicates that these algorithms are convincing enough that participants were no better at determining whether these traces were real than random chance.

The algorithm with the worst performance is the GAN. This is likely due to the fact that the GAN is not restricted to following roads so the traces are obviously fake as they traverse through obstacles. If these traces weren't plotted on a map (and therefore participants didn't know where the obstacles were) these traces may be significantly more convincing.

The question that fooled the most participants was Q54 which was rand-interp 0 with a fool rate of 86%. This trace can be seen in figure 29. This trace repeatably goes over the same area and therefore appears to be very short. This is hard to tell in the still image shown in figure 29 and in the animated version shown in the survey. This makes it appear as if the trace only follows roads in a believable way when in fact (due to the lack of backtracking prevention) the trace actually goes the wrong way on a one-way road.

The next 2 traces that fooled the most participants are Q74 (rand-cat 0) and Q90 (rand 3) both with a fool rate of 76%.

Figure 30: rand trace 3

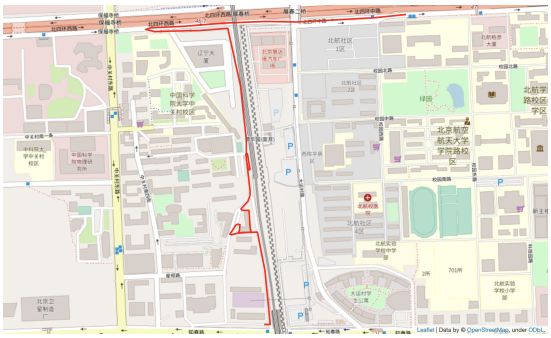

Figure 31: rand-cat trace 0

	gan				rand-cat			
	Precision	**Recall**	**F1 Score**	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Accuracy**
Fake	67%	94%	79%	74%	70%	95%	81%	77%
Real	90%	54%	68%		92%	60%	72%	
	rand				**rand-no-back**			
	Precision	**Recall**	**F1 Score**	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Accuracy**
Fake	74%	97%	84%	82%	68%	94%	78%	74%
Real	95%	66%	78%		90%	55%	68%	
	rand-interp				**rand-interp-no-back**			
	Precision	**Recall**	**F1 Score**	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Accuracy**
Fake	77%	96%	85%	83%	72%	94%	81%	78%
Real	95%	70%	81%		92%	62%	74%	

Table 4: The precision, recall, F1 Score and accuracy of the predictions of the classifier. Each algorithm was trained at tested separately against the data-set of real traces

As shown in figure 30 trace rand 3 follows a major road in a realistic and predicable way. It seems that in this case the random starting point selected was on this major road and at each time step there was no option to divert off of this road. This results in the trace following the road perfectly producing a very realistic trace.

As shown in figure 31 trace rand-cat 0 the trace follows roads and tend to head in a single direction. However, there are a few parts where it goes into a dead end and loop back on itself. This makes the trace seem fairly realistic.

7.3 Quantitative Results

In order to evaluate the algorithms produced for this project computationally the convolutional neural net classifier was used (as described in section 5). For each algorithm the classifier was trained using a balanced data set of real traces and traces generated by the algorithm being tested. The precision, recall, F1 score and accuracy of the classifier was then calculated by testing it against a similar unseen test data set. The results of this are recorded in the table below.

Table 4 shows the classification metrics (precision, recall, F1 score and accuracy) of the classifier. The classifier was separately trained and tested for each algorithm so there is a separate sub-table for each of the algorithms.

An algorithm performing well will get a low accuracy score (as this means that it is successfully fooling the classifier). Therefore the 2 best algorithms when evaluated computationally are the GAN and rand-no-back. The GAN doesn't have the same disadvantage here as it did in the user study because just like the generator in the GAN the classifier doesn't have implicit knowledge of the locations of obstacles. Therefore the classifier would only be able to identify that the GAN traverses through obstacles if it learned where the obstacles are.

Since the classifier doesn't have direct access to the map data it has to base its classification more on the shape of the GPS trace. The algorithms that have back tracking prevention (rand-no-back, rand-interp-no-back) have lower accuracy than their corresponding back tracking counter parts. This suggests that the classifier has learned that back tracking is an uncommon trait in real traces.

Surprisingly, the algorithms with temporal interpolation have higher accuracy than their non-interpolating counter parts since temporal interpolation made an algorithm more likely to fool a human participant. This may be because when temporal interpolation is applied the distances between sample points will be extremely consistent because of the interpolation. The classifier may have learned to pick up on this unnatural level of consistency. This problem could potentially be mitigated by applying a small amount of random perturbation to the traces.

Furthermore the rand-cat algorithm has comparable accuracy to the rand-interp-no-back algorithm. These two algorithms are similar apart from the fact that the rand-cat algorithm only considers the probability of transitions between classes of nodes whereas rand-interp-no-back considers the transition probability between each node separately. This demonstrates that generalising nodes to be within certain classes doesn't significantly hinder the algorithm from generating realistic traces.

In every case there is a similar pattern in the precision and recall values. There is high recall but low precision for fake traces and low recall but high precision for real traces. This indicates that the classifier is over labelling traces as fake which makes it more likely that a fake trace will be labelled correctly but also more likely that real trace will be mislabelled.

Consider a service that is collecting data to sell to advertisers. They know that some of their users are using synthetic GPS traces to protect their privacy. Advertisers are interested in this data because it is an accurate reflection of the users' habits which means it can be used to create targeted ads. Therefore if advertisers were trying to filter out synthetic data (because it is worthless to them) they may decide that they prefer a classifier with this bias because maximising precision on the 'real' class means that if this classifier was used to create a filtered 'real' data set then few synthetic traces would remain. However, due to the low recall this would result in a large proportion of valuable real data being discarded. Therefore they may prefer a classifier with high 'real' recall that only filters out synthetic traces that are very obviously fake. This is especially true since on a real service it is likely only a small subset of privacy concerned users would be providing fake data.

8 Conclusions

In summary this project has produced several algorithms that are capable of generating fake GPS traces. I have shown how a GAN based generator generates traces that are convincing to a neural network based classifier but are not convincing to humans who have access to the map data. In order to address this issue several algorithms that perform walks across the network of roads on the map are proposed. These algorithms produce more convincing traces when plotted on a map. Furthermore, enhancements to this random walk class of algorithm such as temporal interpolation and back tracking prevention increase the quality of the generated traces. Finally, instead of learning the probability of transitioning between specific nodes on the map (which confines the generation to areas where there is data) the transition probabilities can be learned for classes of nodes which allows the trace generation to be generalised to any location with map data.

These algorithms meet the requirements of the first primary objective of this project. Additionally, these algorithms were evaluated by humans in a user study and computationally using a neural network. Therefore this project meets the remainder of the primary objectives. Furthermore distance and speed preserving interpolation allows a synthetic trace that has the same speed and distance characteristics as a given target trace to be generated. This technique could be used to generate synthetic traces that closely match a user's original trace while preserving their privacy. Therefore this project also meets the first secondary aim.

Additionally, by learning the probabilities of transitions between node classes rather than between specific nodes this project has been able to generalise its learning to areas where real GPS data sets are not available. The rand-cat algorithm can generate traces for any area for which map data can be obtained, because OpenStreetMap have maps for the whole world this puts no limit on where traces can be generated for. Furthermore, generalising learning in this way does not significantly reduce the realism of the generated traces. The draw back to this technique is that without real data to inform the transition time between nodes some other source has to be used for temporal interpolation, this may result in less realistic traces. However, this is not an issue if distance interpolation is being used.

This project also produced a GAN that can generate GPS traces. This GAN produces traces that are more convincing to computational techniques but are not convincing to humans. This is because both the GAN and the neural network classifier do not have access to the map data and so the GAN produces traces that traverse through obstacles. The neural network classifier has no way of determining that this has occurred, this means that humans find the traces less convincing but the classifier is not effected.

The major drawback to most of the algorithms produced in this project is that they can only be used in areas of the world where their is training data. However, this drawback is solved by rand-cat algorithm which generalises to any area.

The ability to generate a trace which has the same speed and distance characteristics of a target trace is beneficial to privacy protection. This is because it allows users to get the benefits of apps that use this information (such as fitness apps) without exposing other

personal information. If none of the information from a real trace is preserved then the use of synthetic traces has no benefits over simply disabling GPS altogether.

Future work in this area should therefore focus on producing methods to preserve the useful information needed for other applications. Each type of application will have different requirements for the accuracy and type of data that is extracted from the original data. The algorithms proposed in this work also are not entirely accurate at reproducing traces with the original distance and speed information. Future work in this area should work to improve this accuracy. It would also be beneficial for future works to improve how convincing the synthetic traces produced by these algorithms are.

Additionally, it would be beneficial to encode the map data so that a neural network based generator has access to it. This would make it simpler for a generator network to produce traces that follow the road network without having to learn he positions of every node.

9 Acknowledgments

I dedicate this project to the memory of Gary Brand, who inspired me to pursue computer science.

I would like to thank my supervisor Kasim Terzić for his help and support throughout this project. Additionally, I would like to thank the students that participated in the user study.

The map data used in this project and report is copyright OpenStreetMap contributors.

References

[1] J.A. Alvarez-Garcia et al. "Trip destination prediction based on past GPS log using a Hidden Markov Model". In: *Expert Systems with Applications* 37.12 (2010), pp. 8166–8171. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2010.05.070. URL: http://www.sciencedirect.com/science/article/pii/S0957417410004811.

[2] Konrad Bösche et al. "Scalable generation of synthetic GPS traces with real-life data characteristics". In: *Technology Conference on Performance Evaluation and Benchmarking*. Springer. 2012, pp. 140–155.

[3] Brilliant.org. *Markov Chains*. 2020. URL: https://brilliant.org/wiki/markov-chains/.

[4] Thomas Brinkhoff. "A framework for generating network-based moving objects". In: *GeoInformatica* 6.2 (2002), pp. 153–180.

[5] OpenStreetMap Contributors. *OSM XML*. 2020. URL: https://wiki.openstreetmap.org/wiki/OSM_XML.

[6] OpenStreetMap contributors. *OpenStreetMap*. 2020. URL: https://www.openstreetmap.org/.

[7] TensorFlow Contributors. *TensorFlow Core v2.3.0*. 2020. URL: https://www.tensorflow.org/api_docs.

[8] Sina Dabiri and Kevin Heaslip. "Inferring transportation modes from GPS trajectories using a convolutional neural network". In: *Transportation research part C: emerging technologies* 86 (2018), pp. 360–371.

[9] Ishita Dwivedi. "Privacy-preserving Trajectory Data Publishing Via Differential Privacy". PhD thesis. Boise State University, 2017.

[10] Rohith Gandhi. *Generative Adversarial Networks — Explained*. 2018. URL: https://towardsdatascience.com/generative-adversarial-networks-explained-34472718707a.

[11] Michael R Gardner, Wayne W Ballantyne, and Zaffer S Merchant. *System and method for E911 location privacy protection*. US Patent 7,751,826. July 2010.

[12] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[13] Google. *Location*. 2019. URL: https://developer.android.com/things/sdk/drivers/location.

[14] Marco Gruteser and Dirk Grunwald. "Anonymous usage of location-based services through spatial and temporal cloaking". In: *Proceedings of the 1st international conference on Mobile systems, applications and services*. 2003, pp. 31–42.

[15] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[16] Baik Hoh et al. "Preserving privacy in gps traces via uncertainty-aware path cloaking". In: *Proceedings of the 14th ACM conference on Computer and communications security*. 2007, pp. 161–171.

[17] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[18] Scott Ikeda. *Many of the Major Dating Apps Are Leaking Personal Data to Advertisers*. 2020. URL: https://www.cpomagazine.com/data-privacy/many-of-the-major-dating-apps-are-leaking-personal-data-to-advertisers/.

[19] Apple Inc. *New features available with iOS 14*. 2020. URL: https://www.apple.com/ios/ios-14/features/.

[20] IncorporateApps. *Fake GPS GO Location Spoofer Free*. 2020. URL: https://play.google.com/store/apps/details?id=com.incorporateapps.fakegps.fre.

[21] Thomas Kluyver et al. "Jupyter Notebooks – a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90.

[22] Kiprono Elijah Koech. *Cross-Entropy Loss Function*. 2020. URL: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.

[23] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[24] Lexa. *Fake GPS location*. 2018. URL: https://play.google.com/store/apps/details?id=com.lexa.fakegps.

[25] Linchao Li et al. "Coupled application of generative adversarial networks and conventional neural networks for travel mode detection using GPS data". In: *Transportation Research Part A: Policy and Practice* 136 (2020), pp. 282–292.

[26] Hola VPN Ltd. *Fake GPS location - Hola*. 2020. URL: https://play.google.com/store/apps/details?id=org.hola.gpslocation.

[27] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[28] Wesley Mathew, Ruben Raposo, and Bruno Martins. "Predicting future locations with hidden Markov models". In: *Proceedings of the 2012 ACM conference on ubiquitous computing*. 2012, pp. 911–918.

[29] Wes McKinney et al. "Data structures for statistical computing in python". In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56.

[30] Marius Muja and David G Lowe. "Fast approximate nearest neighbors with automatic algorithm configuration." In: *VISAPP (1)* 2.331-340 (2009), p. 2.

[31] App Ninjas. "What Are GPS Spoofing Apps Actually Doing?" In: *Medium* (2017).

[32] PaulMcG. *Calculate distance between 2 GPS coordinates*. 2015. URL: https://stackoverflow.com/questions/365826/calculate-distance-between-2-gps-coordinates.

[33] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[34] Jeffrey H Reed et al. "An overview of the challenges and progress in meeting the E-911 requirement for location service". In: *IEEE Communications Magazine* 36.4 (1998), pp. 30–37.

[35] Jean-Marc Saglio and José Moreira. "Oporto: A realistic scenario generator for moving objects". In: *GeoInformatica* 5.1 (2001), pp. 71–93.

[36] Pierangela Samarati and Latanya Sweeney. "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression". In: (1998).

[37] Stephen Schroeder. *Top 11 Worst Location Data Privacy Breaches*. 2017. URL: https://turtler.io/news/top-11-worst-location-data-privacy-breaches.

[38] seantur. *pyflann-py3*. 2019. URL: https://pypi.org/project/pyflann-py3/.

[39] Dara E Seidl, Piotr Jankowski, and Ming-Hsiang Tsou. "Privacy and spatial pattern preservation in masked GPS trajectory data". In: *International Journal of Geographical Information Science* 30.4 (2016), pp. 785–800.

[40] Rob Story. *Folium 0.11.0 documentation*. 2013. URL: https://python-visualization.github.io/folium/.

[41] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[42] Michael Waskom et al. *mwaskom/seaborn: v0.8.1 (September 2017)*. Version v0.8.1. Sept. 2017. DOI: 10.5281/zenodo.883859. URL: https://doi.org/10.5281/zenodo.883859.

[43] Yu Zheng et al. *Geolife GPS trajectory dataset - User Guide*. Geolife GPS trajectories 1.1. Geolife GPS trajectories 1.1. July 2011. URL: https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/.

Appendices

A Example OpenStreetMap XML File

[5]

B Ethics

The artefact evaluation form for this project is attached on the next page.