Generative Models for City-Specific Vehicle Trajectories

Major Qualifying Project

Authors:
David Larson
Benjamin Longo
Caleb Ralphs

Advisor:
Dr. Yanhua Li

WPI

May 18, 2020

Contents

Abstract

Modeling vehicle trajectories in cities is an important task that can help transportation planners make smart decisions around traffic policies and transportation infrastructure. Often times there is not enough data available for the planners to use, or the data available contains sensitive personal information which, if used, would violate data privacy policies. Without data, uninformed decisions can lead to traffic congestion and inadequate public transportation infrastructure. With the annual costs of traffic congestion exceeding one trillion U.S. dollars worldwide, this is obviously a problem worth addressing. We propose a method of generating new samples of vehicle trajectory data for a given city, leveraging existing GAN models with novel data representations.

1 Introduction

Generative models such as bayesian inference models, hidden markov models, variational autoencoders, generative adversarial networks, and much more have, shown promise when tackling data problems. These intelligent systems do not only serve the purpose of generating new data, but also have shown promise in translating between data modalities, learning useful representations of data, and filling in missing data [25]. Specifically, deep learning models have proven effective in artificially intelligent applications such as image processing, audio transformation, natural language processing, and much more [10]. Generative adversarial networks (GAN), a type of deep learning model, tackles these same applications, but with a different model framework and approach.

Deep learning models have also shown promise in the realm of urban intelligence. Cities face problems, ranging from air pollution to mobile communications to traffic congestion. In this paper, we focus on addressing problems relating to vehicle movements around cities. Determining the movement of vehicles in cities is an important task that can help urban and transportation planners make decisions around traffic policies and extensions of current transportation infrastructure. The trajectories of vehicles, when combined with computer representations of city road networks, play an important role in guiding transportation policy, but many cities lack an adequate amount of data. These data shortcomings limit the ability for urban and transportation planners to make informed decisions around recommending allocations of resources to guide transportation policy. Uninformed policies can lead to the proliferation of inherent traffic congestion problems and inadequate public transportation infrastructure. With the annual costs of traffic congestion in the United States is nearly $160 billion, exceeding one trillion U.S. dollars worldwide, this is obviously a problem worth addressing []. The use of IoT sensors, crowd source data, and intelligent analysis on urban data enables urban and transportation planners to make more informed decisions in solving these problems [34]. Long-short term memory (LSTM) models, a time-variant type of deep learning model, have been used to make short-term traffic forecasts. Other models have combined personal driving data with other environmental variables, such as weather, to

predict traffic flow at a broader level. Vehicle trajectory data, collected from divers, is necessary to create robust models that accurately capture traffic flow, patterns, and, ultimately, inform transportation policy decisions. Although this data is important, data privacy issues have come to the forefront in recent years, making it difficult to leverage this publicly collected data. These sorts of privacy-policies have stemmed from the 2018 General Data Protection Regulation set in place from the EU which encompasses legislation around sensitive data []. Often times, the data being used in these traffic prediction or path prediction models is sensitive, containing information about where an individual frequently travels to or from.

We propose a method of generating new samples of vehicle trajectory data for a given city, leveraging existing GAN models with novel data representations. The goal of this research is to generate likely, realistic vehicle trajectories for a given city, using a subset of publicly available vehicle trajectory data. This approach solves the data privacy issue, as all data generated from the models is technically fake and does not contain any sensitive personal information. As long as the trajectories are realistic and the various spatial distributions across the city are consistent, then for all intents and purposes the generated vehicle trajectories could be used for subsequent traffic or transportation models. This capability empowers urban planners to make informed policy decisions while complying with data-privacy guidelines.

In this paper, we will:

- survey multiple vehicle trajectory datasets and different types of GANs to determine the best mode of creating realistic vehicle trajectories for a given city;
- develop a novel data representation through the use of gridding and diffusion embedding which allows us to leverage GAN models that would not be practical given the original data;
- identify strengths and weaknesses of various models in their ability to create realistic vehicle trajectories, while also highlighting and addressing complications with the data.

2 Preliminaries

The current state-of-the-art trajectory data generative models leverage GANs and Bayesian inference modeling . Li et. al. proposed a Coordination-Bayesian Conditional Generative Adversarial Network (C-BCGAN) for generating vehicle trajectories for a given network scenario [2]. The C-BCGAN implementation was effective for producing high resolution samples, but at a micro-region scale, such as modeling traffic around a few connected intersections and roadways. The proposed model leveraged Bayesian inference, which did not allow for robust modeling of long-term dependencies, such as car trajectories across larger regions, like a city. Larger, macro-level trajectory generation with more than

just current state dependence is necessary to provide urban and transportation planners with holistic representations of vehicles trajectories for extended durations and distances. In our preliminary research, we introduce two GAN variants, a sequential GAN (SeqGAN) and a deep convolutional GAN (DCGAN), that we implemented for tackling this vehicle trajectory problem, without the macro-level shortcomings of C-BCGAN. We also take a deep-dive into the data representations that we encounter with the datasets we surveyed and how those intricacies affect model development and selection.

2.1 Generative Models

2.1.1 Bayesian Probability Theory

Generative models are a method of modeling how some observed data may have derived from some set of underlying causes [22]. The most simple representations of generative models are built on top of basic Bayesian probability theory, the probability of an event happening given a prior event has already occurred [22]. Bayesian inference models give rise to creating probabilistic density estimations using previously observed data [22]. These models can be represented with Bayes's theorem, in three terms: priors, likelihoods, and evidence

$$P(A|D) = \frac{P(D|A)P(A)}{P(D)}$$

where $P(A)$ is the prior, $P(D|A)$ is the likelihood of observing D given the distribution formed from A, and $P(D|B)$ is the posterior distribution, a combination of the prior distribution and the likelihood function [24]. The posterior distribution can be generalized as a summary of all the observed data. These three terms form a basic generative model where A can be generated given the prior and the likelihood.

Bayesian inference models set the foundation for multivariate generation of distributions in the sense that a set of data, D is explained by a set of underlying causes α, which can be thought of as a combination of the priors and likelihoods[22]. To create the model, M, two problems need to be solved: inferring the optimal set of causes to explain each data point D_i and learning the optimal model M to explain the complete set of data D [22]. To infer each specific data point $D_i \forall\ D_i \in \{D_i, ..., D_n\}$

$$\begin{aligned}\hat{\alpha}_i &= argmax_\alpha P(\alpha|D_i, M) \\ &= argmax_\alpha \frac{P(D_i|\alpha, M)P(\alpha|M)}{P(D_i|M)} \\ &= argmax_\alpha P(D_i|\alpha, M)P(\alpha|M)\end{aligned}$$

where the denominator, $P(D_i|M)$ is equal to one, as D_i has already been selected. Since $\hat{\alpha}_i$ is calculated for every point D_i, all causes for D are defined. This covers the inference part of the model creation. Now, the model M is

created by maximizing the aforementioned posterior distributions, as defined in Bayes's theorem

$$P(M|D_i) \propto P(D_i|M)P(M)$$

where the denominator, $P(D_i)$ is ignored since D is selected and therefore, fixed. To find the model M that maximized the overall likelihood of the data D, the total probability of D is taken

$$P(D|M) = \prod_{i}^{n} P(D_i|M)$$

where $P(D_i|M)$ is the calculated by taking the sum of all the causes α for that point D_i

$$P(D_i|M) = \sum_{\alpha} P(D_i|\alpha, M)P(\alpha|M)$$

and since the goal is to create a generative model, M must be calculated with

$$\begin{aligned} M &= argmax_M P(D|M) \\ &= argmax_M \sum_{i}^{n} P(D_i|M) \\ &= argmax_M \sum_{i}^{n} \sum_{\alpha} P(D_i|\alpha, M)P(\alpha|M) \end{aligned}$$

The model M is now representative of the max likelihood posterior distribution over data D. Data can now be sampled from the generated posterior distribution of model M.

2.1.2 Markov Chain Monte Carlo

While the previously defined Bayesian inference model is a means of interpreting observed data, Markov Chain Monte Carlo (MCMC) is a method of sampling from a defined distribution, like that of model M [27]. These two methods, together, yield a means of efficiently modeling and sampling from a given distribution [27]. The Monte Carlo part of MCMC refers to the random sampling from some distribution, like that posterior distribution model M. A Markov chain is defined as a set of random variables $X_1, ..., X_n$ such that the future and past states are independent of the current state [27]. This means that the future state X_{n+1} is only dependent on the current state X_n [27]. For a continuous state space, the future state is defined as

$$T(i, j) = P(X_{n+1} = i|X_n = j)$$

where T is referred to as the transition operator. It is convenient to represent this transition probability as a matrix of the form T_{dxd} [26]. The transition operator or probability can now be defined as

$$T_{i,j} = P(X_{n+1} = i|X_n = j)$$

This sampling of sequence X, providing X_{n+1}, or X_{new}, from X_n, or X_{prev}, is the Markov chain element of MCMC. This allows for vector probability sampling from T [26].

MCMC does, however, run on the assumption that there is a stationary distribution, meaning that $P(X_{n+1} = i|X_n = j)$ must be the same at each time step n [26]. This can be interpreted as the transitional probability for movement from the current state to another is solely dependent on the current state, not the other sequence of previous states. This assumption constrains the model's probability space to being stationary and for the vector generated from T to be independent from the state history.

2.2 Generative Adversarial Networks

A Generative Adversarial Networks (GAN) is a type of deep learning model consisting of two artificial neural networks which work together in an adversarial manner. The first of these models is trained to generate data realistic with respect to some input dataset. The second model is trained to distinguish the distribution of the generative model from that of real data. As the discriminative model is trained to more accurately identify these data, the generative model learns to produce more accurate data to fool the discriminator. In this way, the two models learn from one another.

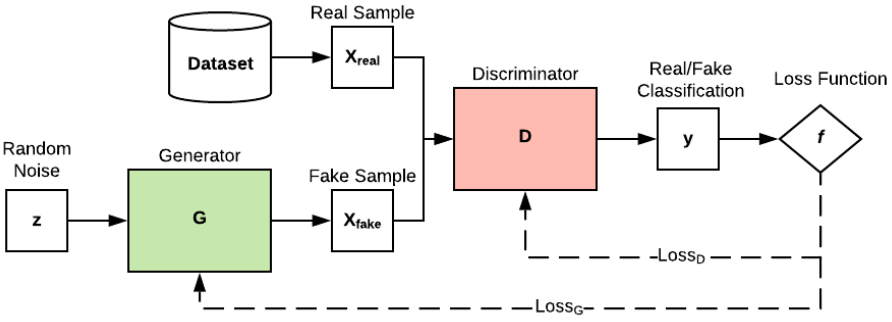

Figure 1: *The high-level structure of a basic GAN.*

More formally, in order to learn the distribution p_g of generated data over the data set x, we need to define a prior distribution for the noise in the data $p_z(z)$ [10]. Then, we can represent the generative distribution as $G(x; \theta_g)$ where θ_g is a an input data point. Then, the second network $D(x; \theta_d)$ will output a single scalar between 0 and 1 representing the probability that x came from the data instead of p_g. From here, we train model D to increase the accuracy of it's discrimination, while we train G to minimize $log(1 - D(G(z))$, resulting in a game of minimax between the two models [10].

2.2.1 Sequence GANs

Although traditional GANs have proven to be effective in generating realistic data, they have a considerably limited ability to generate discrete sequential data [31]. As previously explained, the training of the generator is an iterative process where the model parameters are slowly tuned by the gradient of the loss from the discriminator. This tuning is only a slight change, but when considering a discrete data space, a slight change will likely not result in any other meaningful discrete variable, but rather, a value that lies between two meaningful variables [31]. The other drawback that traditional GANs have with generating discrete sequential data is their inability to evaluate a partial sequences of data [31]. Yu et al. addresses these problems with the proposal of SeqGAN where generator G is treated as an policy in a reinforcement learning approach.

In respect to reinforcement learning, a policy refers to a parameterized method of modeling an optimized behaviour. The policy is changed by means of a policy gradient which models and optimizes the generator's policy G_θ directly. The policy gradient's objective function is defined as

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s)$$

where $d^\pi(x)$ is the stationary Markov chain for a policy state distribution π_θ and $V^\pi(x)$ is the expected value return of the state, referred to as the state-value function [31]. This objective function is modified in respect to the GAN model to make

$$J(\theta) = \sum_{a \in A} G_\theta(a_1|s_0) \cdot Q^{G_\theta}_{D_\phi}(s_0, a_1)$$

where G is the θ-parameterized generator, D is the ϕ-parameterized discriminator, and $Q^{G_\theta}_{D_\phi}(s, a)$ is the expected reward starting from state s, taking action a, and then following the policy G_θ [31]. As previously stated in the drawbacks of the standalone GAN model, the action-value function needs to be evaluated not only in the previous tokens, but also the future tokens. To evaluate the future tokens a Monte Carlo search is used to sample the unknown $(T - t)$ tokens where T is the total number of possible candidate tokens and t is the current timestep. [31]. The execution of the policy G_θ on the current state in respect to the unknown future state is modeled by G_β, referred to as the roll-out policy. The discriminator D_ϕ allows for the reward function to be dynamically updated, iteratively improving the generator G_θ [31]. Now the model can be trained like a traditional GAN model, alternatively training the generator and discriminator.

The generator must also be adapted to effectively generate sequential data where the input training data $x_1, ..., x_n$ is mapped to a sequence of hidden layers in the neural network $h_1, ..., h_n$ [31]. This neural network structure of sequential hidden layers is referred to a a recurrent neural network (RNN).

More specifcally, Yu et al. proposes a long short-term memory (LSTM) RNN to deal with the vanishing gradient problem, since the gradient must

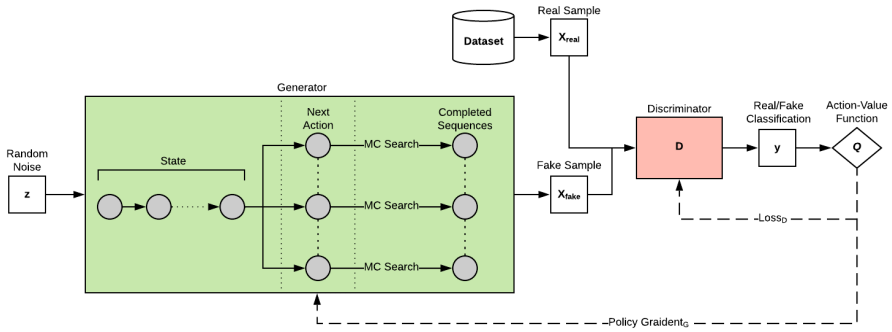

Figure 2: *The high-level structure of SeqGAN where the discriminator D_ϕ provides a reward for a completed sequence and updates the intermediate value for the next action via the Monte Carlo Search [31].*

backpropogate through a sequence of hidden layers [31]. The LSTM allows the RNN to learn long-term dependencies through their inherent structure [7]. Where the RNN can be thought of as a sequence of hidden layers, the LSTM should be envisioned as a cell, containing multiple special layers [7]. The core idea behind the structure of the LSTM is the ability to keep or throw away memory of previous data, regulated by memory gates [7]. Each cell on the model has four layers where, three of which, decide what data is kept, considered, and passes on to the next cell in the sequence [7]. The first of these layers is the forget gate f_t, which is a sigmoid layer that outputs values between 0 and 1, 0 meaning completely forget this data, and 1 being completely keep this data [7]. The next step is to create a vector of new values, to be added to the current state, referred to as candidate values C_t [7]. The current cell state is then updated by multiplying the old state C_{t-1} by f_t, forgetting certain data, then adding $i_t * \tilde{C}_t$ where i_t is the parameter for how much each state value is changed [7].

The current state is now of the form

$$C_t = f_t * C_{t-l} + i_t * \tilde{C}_t$$

where C_t is the updated cell state [7]. The output is then decided through running the data through a sigmoid layer, then a tanh layer to push the values between -1 and 1. Now the output is of the form

$$h_t = o_t * \tanh(C_t)$$

where o_t is the output from the sigmoid function. The SeqGAN model can now effectively generate sequential data.

2.2.2 DCGAN

Deep Convolutional Generative Adversarial Networks (DCGAN) have been proven to be effective unsupervised learning method for generating image data [23]. DC-

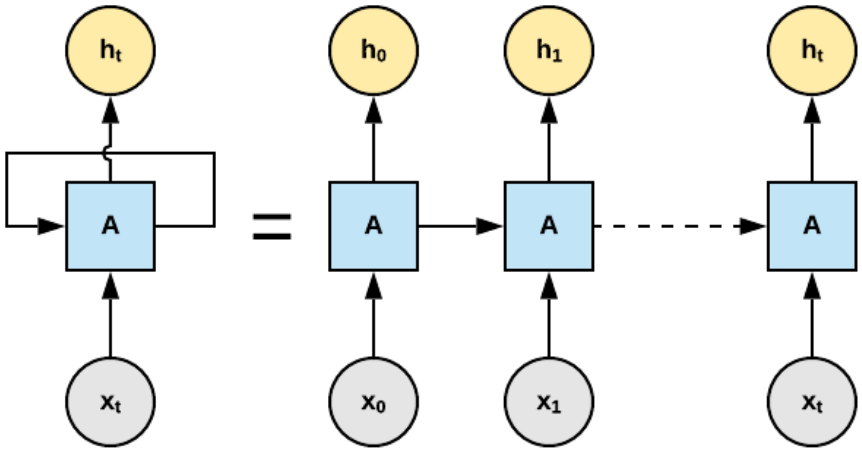

Figure 3: *The high-level structure of a recurrent neural network. A represents formulation of the* $tanh(W_h h_{t-1} + W_x x_t)$ *where* W_h *is the weights to multiply the previous hidden state* h_{t-1} *by, and* W_x *is the weight to multiply the current input* x_t *by. Every state A shares the same weights, i.e., the same learned function[7].*

GAN differs from traditional GANs by its usage of hierarchical representations of the data that the model learning with convolutions and deconvolutions in the generator and discriminator, respectively [23]. DCGAN is also faster than traditional GANs since it uses batch normalization in some of its layers which stabilizes and speeds up the learning by reducing the amount that the hidden layer's unit values shift around through normalizing them between 0 and 1 [23].

Additionally, the DCGAN model structure eliminates fully connected layers and uses ReLU activations in generator and Leaky ReLU activations in the discriminator. The ReLU function, a rectified linear unit, outputs the value of the data if it is greater than 0, and 0 otherwise. Convolutional neural networks, and specifically the DCGAN model, use ReLU units to introducde nonlinearities into the system without creating a vanishing gradient problem something that some activation functions such as signmoid and tanh may create. The discriminator in DCGAN uses a Leaky ReLU function which addresses problems that can arise from the Dying ReLU problem. The dying ReLU problem is where the ReLU neurons in the network "die" and remain inactive regardless of the input that is supplied to the network layer [1]. The "leaky" part of the Leaky ReLU addresses this problem by changing the slope to the left of $x = 0$ to a small slope of $y = .01x$, creating a leak for negative inputs, making them extremely small, but non-zero [1].

DCGAN adopts convolutional neural network architecture with strided convoltuions, allowing the model to learn the spatial downsampling on its own, rather than specifying the the spatial representation with pooling functions such

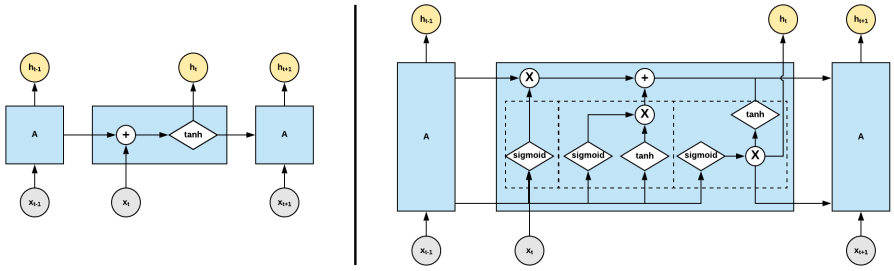

Figure 4: *The RNN state structure (left) versus the LSTM structure (right). In the LSTM, the three dotted boxes represent the forget gate, the input gate, and the output gate, respectively, from left to right. The forget gate determines how previous information will propogate forward to the next timestep, while the input gate handles the new input, and the output gate combines the output from the forget gate and input gate.*

as maxpooling [23]. The strides in the convolutional layers define the step size for the kernel, or how many cells that the kernel filter skips per step through the data, which is the field of view for the convolution, i.e., a kernel size of 3 yields a 3x3 pixel filter. This results in the convolutional layers changing the number of input channels and outputting a smaller number of output channels.

2.3 Data Types

2.3.1 Sequential Data

Sequential data is a vector where the ordering of the data entries has importance, creating a sense of temporal dependencies. Examples of this include speech, music, video, biological sequences (DNA sequences), and many others [5]. Ordering is important in sequential data, and each vector of a given data point is typically at a constant resolution or sampling rate.

Time series data is a subset of sequential data. The key difference here is that one of the features of a given vector must be a timestamp (unless there is a constant sampling rate), whereas such is unnecessary in purely sequential data. If we take the example of a song, the timestamp of any portion of the song is irrelevant. The only important thing is the order in which the discrete portions of the song occur. However, if we are looking at perhaps data representing the ocean's tide patterns, this data is useless unless we know the timestamps associated with each data point.

Furthermore, a subset of time series data is spatio-temporal data. As one could probably guess, spatio-temporal data is the combination of spatial coordinates (perhaps latitude and longitude) and a timestamp. A series of vectors containing these coordinate-timestamp pairs thus constitutes the trajectory of some object over time. We can also have other spatial measures, such as a volume, area, matrix, or graph, with respect to time and it would be considered

spatio-temporal data. For the scope of this paper, we will be focusing on data consisting of coordinate pair with a timestamp.

2.3.2 Vehicle Trajectory Data

An example of spatio-temporal data is the trajectory a vehicle through a road network. This type of data is used extensively to attempt to predict traffic patterns, congestion, and travel times [34]. The ability to generate realistic vehicle trajectories has many appealing applications ranging from autonomous vehicle development, to civil engineering. Due to the high complexity of human driving behavior and interactions, creating expert system simulators that produce realistic results is a hard problem, and a very active area of research [3].

Several authors have applied GANs to the domain of vehicle trajectories [16, 32, 17, 2, 9, 11], however, few of these models successfully tackle the problem of generating macro-scale trajectories.

2.3.3 Sequence Imputation

A related task to sequence generation is sequence imputation. Imputation is the process of filling missing values in an existing, partially observed sequence. Traditionally this is accomplished via some interpolation method, one of the simplest cases being linear interpolation. More advanced approaches such as ARMA, ARIMA, and other smoothing methods have been developed to create more realistic samples; however, the majority of the approaches utilize linear dynamics and typically model sequences as autoregressive. Additionally, any smoothing interpolation based methods discard the relationships between variables in a complex multivariate time series.

Due to these limitations, several researches have recently investigated the application of neural networks to impute sequence data. However, RNNs in general face significant issues when dealing with long term dependencies. Although LSTMs attempt to address this issue, they can still not handle *very long* temporal dependencies well.

Vehicle trajectories generally follow an important property: if a path goes from point a to c through b, it would also follow the same path from a to b even if it's destination wasn't c; this applies recursively. This is certainly not true of all sequence data. For instance, the start and end word of a sentence do not come close to fixing the space of possible sentences - the start and end point of a trajectory however closes down the size of the possible space by quite a lot (assuming we have efficient drivers).

There is an inversion of temporal dependency from autoregressive sequence data. Suppose we have a trajectory $[p_1, p_2, p_3, p_4, p_5]$, then starting at p_3, the path is determined almost completely by p_5 and not significantly by p_1 or p_2.

If we had a model that could impute an entire vehicle trajectory from its endpoints, it is essentially a full generative model. Modelling the distribution of start-end pairs is much more statistically tractable with traditional methods than the actual driving behavior.

This may not be entirely true if the driver decides to take a different path because of bad traffic however. In order words, it is possible that past points influence a drivers decision about a path - but certainly future points are vastly more important in deciding the path. With this in mind, it seems much simpler to generate a trajectory given a source and destination pair than to create one from strictly a source. When choosing to remove this facet from the model, we are making it less general, but perhaps more successful and useful in practice.

3 Methodology

In figure 5, you can see the life cycle of our methodology for our research. For the dataset selection, we focused on datasets that contained meaningful and interpretive information about each trip. As you will see, that is the Beijing Taxi dataset, which contained crucial information on occupancy status at each recorded timestep. After selecting the Beijing dataset, we needed to explore the dataset further in order to motivate some of our model selection decisions and ensure that the data was cleaned and complete. After our pre-analysis, we faced the task of cleaning the data. As our dataset contained occupancy status, segmenting it became extremely simple. We divided the data into trajectories based on occupancy status, and threw out unoccupied trajectories. The rest of the data cleaning included filtering out trips with long time gaps, applying an azimuthal equidistant projection around the center of the city, partitioning the data into grid, and constaining the bounds considered or the city limits. The last step before experimenting with the various GAN models, VanillaGAN, SeqGAN, and DCGAN, was engineering the data to be in the right format for each model. In this section you will see the presentation of the results for each of those models, ending with evaluation of those results.

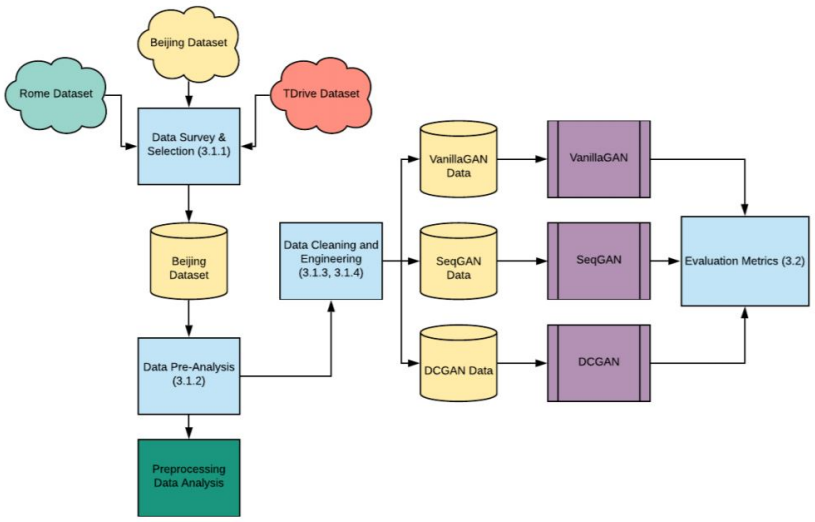

Figure 5: *The progression through which our dataset was chosen, cleaned, manipulated, and evaluated.*

3.1 Data

3.1.1 Data Survey and Selection

Selection of a good dataset is imperative in building a good model. If the model is not being trained on quality data, then the quality of the results will also be poor. In the case of generating vehicle trajectories, if the training trajectories for the generative model are incoherent or inconsistent, then the results will ultimately be skewed, and the model may have trouble learning the "correct" weights. In order to find the publicly available dataset with the most realistic trajectories, a survey was done on various discovered vehicle trajectory datasets, as seen in Table 1.

Name	Area	Timespan	Rate	Points	Drivers
Rio Bus Data [12]	Rio De Janeiro, Brazil	2019-01-25 to 2019-03-21	1m	59,183,745	Unknown
NGSIM[13]	CA & GA, USA	2005-04-20 to 2006-11-09	0.1s	11,850,527	Unknown
DACT [21]	Columbus, OH, USA	2011-06-20 to 2013-08-19	1s	47,846	50
Rome Taxi [4]	Rome, Italy	2013-08-01 to 2014-02-01	7s	21,817,851	320
TDrive [33]	Beijing, China	2008-02-02 to 2008-02-08	177s	17,662,984	10,357
Beijing Taxi	Beijing, China	2009-05-01 to 2009-05-30	variable	129,000,000[14]	8000[14]

Table 1: Considered Datasets

Various aspects of these datasets were compared, including the time frame in which the data was collected, the sampling rate, the total number of sampled

points, the number of drivers, whether or not the data contained occupancy status since most of these datasets were collected from taxi drivers. All of these features seemed to be important, as the data needed to be rich in quality and quantity. The Beijing Taxi dataset was found to be the most desirable due to not only the large size and scope in number of drivers but also because the samples included occupancy status. Occupancy status is integral in creating a robust, realistic training set, since when taxi drivers did not have passengers in their vehicle, their trajectories seemed incoherent and would often times spend much time idle. The occupancy status allows for segmentation of taxi trips, which yields individual, distinct trajectories. The Beijing Taxi data also had ample spatial and temporal coverage in respect to the geography of the city and different times of day and days of the week. There are plenty of samples of data in each and every urban district of Beijing, and the data was collected on a multitude of temporal climates, including weekdays, weekends, and even a public holiday, all with 24-hour data collection [14]. After surveying the identified vehicle trajectory datasets, there were no other dataset which encompassed sufficient temporal and spatial coverage, while also providing occupancy status for segmentation. Therefore, the Beijing Taxi dataset was selected for use in training the generative models.

3.1.2 Data Pre-Analysis

The selected Beijing Taxi dataset was explored and analyzed in order to provide insights into various distributions of the data. Analyzing the distributions of data provides a baseline understanding of the data, and, ultimately, these baseline distributions can be compared with those resulting from the generator to gauge how effective the model was in capturing the unique features of the data. Additionally, visualizing and looking into the distributions of the different features in the data allows for identification of outliers, trends, and can help inform decisions when building the generative model.

The spatial distribution of the data gives an understanding of how dense or sparse various areas are in the city, with respect to where there are high density traffic areas versus sparse ones. Approximately 90% of the trajectories did not deviate further from 20km from the city center in any direction, with the city center being the mean latitude and longitude in Beijing. Deep learning models, and moreover generative adversarial networks, struggle to successfully model sparse data [35]. To this point, the trajectories containing points on the peripheral, low density parts of Beijing were removed, resulting in a, 20km x 20km, 400km^2 bounding box for the data.

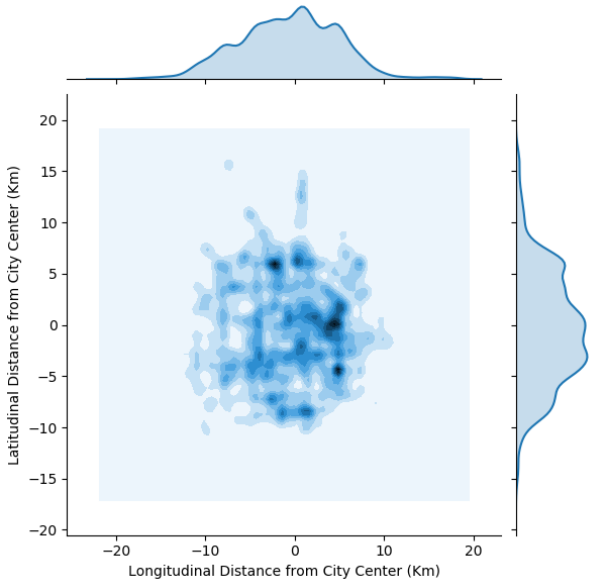

Figure 6: *Spatial distribution from 10,000 sampled trajectories centered around the mean longitude and latitude for the whole dataset.*

In order to assure that there were no problematic outliers in the dataset with respect to the sampling rate, the distribution of the time delta between samples was analyzed. Through this analysis, it was found that there were few durations in time between sampled points on a given taxi trip that were greater than 100 seconds with 75.36% of the consecutive points in the data being sampled within a 1 minute interval [14]. Given that the goal is to generate coherent trajectories, not sparsely sampled ones, all the trajectories containing time deltas greater than 100 seconds were filtered out, as this is approximately where we see our distribution level off.

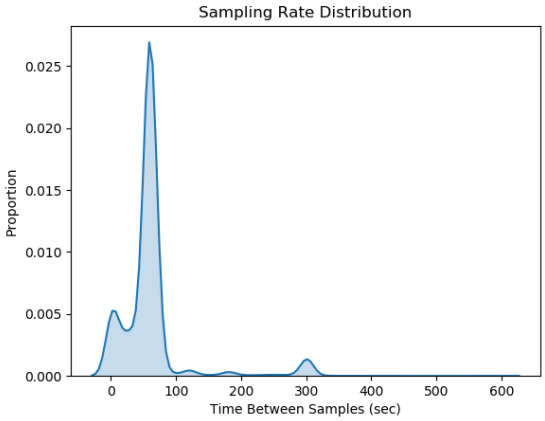

Figure 7: *Distribution in sampling rate between each data point, i.e., time between each recorded longitude, latitude location, from 10,000 sampled trajectories.*

The euclidean distance traveled in each trajectory was relatively low, with a mean trajectory length of 2.31km and 99% of the trajectories being less than 17.06km in length. There were outliers in the trajectory lengths, around 9000km, which may have resulted from numerical miscalculations due to floating point arithmetic. In figure 8, it shows how the distribution is heavily skewed towards the shorted distances, which makes sense given that this is a dataset of taxi trips.

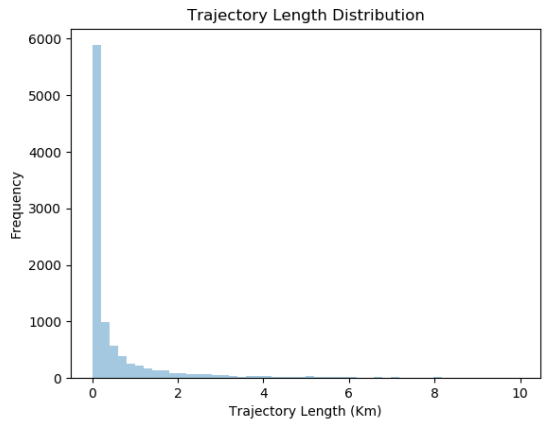

Figure 8: *Distribution of euclidean distance traveled over each trajectory from 10,000 sampled trajectories.*

The velocity was also recorded at each sampled data point in the distribution, and surprisingly, there existed no outliers, with the maximum velocity being

80.8 km/hr and the mean being 7.71 km/hr. These statistics make sense given that these are taxi rides around an, often, traffic dense city. Additionally, the maximum velocity of 80.8 km/hr makes sense, given that the highest speed limit in Beijing is 70 km/hr [6]. With that being said, it was not necessary to remove any samples from the population with the motivation being velocity.

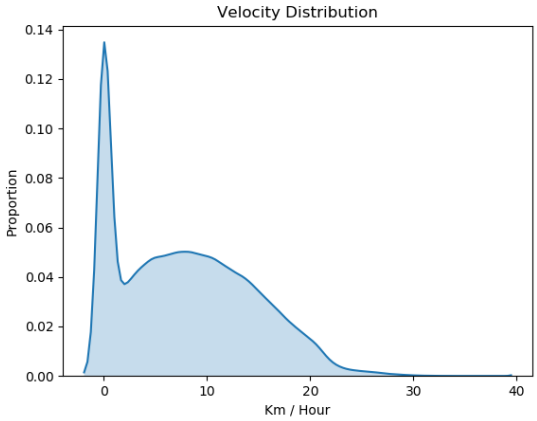

Figure 9: *Distribution of recorded velocities for each data point in 10,000 sampled trajectories.*

3.1.3 Data Cleaning

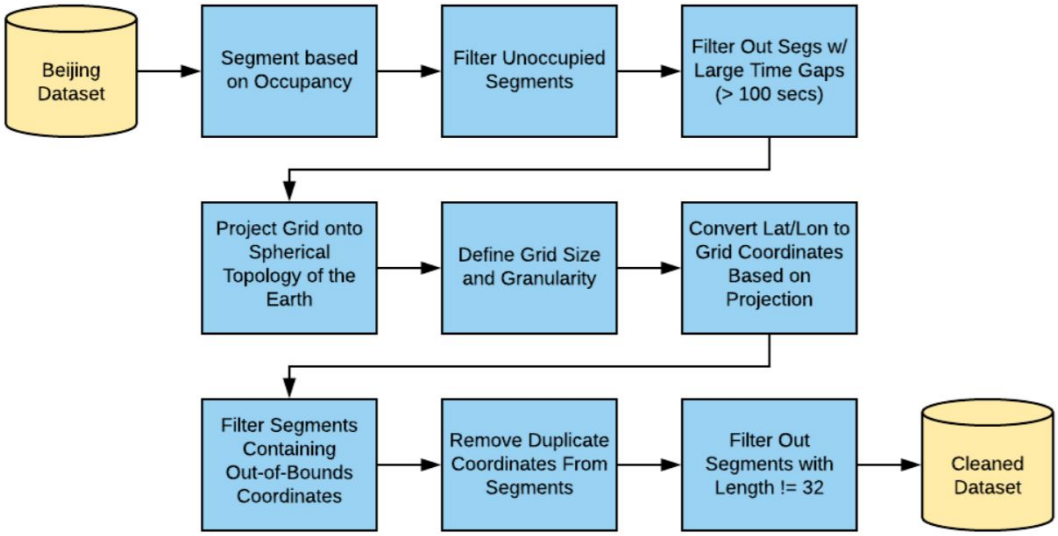

Figure 10: *The process of cleaning the dataset.*

Once a suitable dataset had been chosen, the process of cleaning the dataset began. Several characteristics of the dataset became apparent that needed to be given attention. With other datasets that didn't include the occupancy status of the vehicle at a given time, segmentation yielded trajectories modeling two distinctly different types of behavior: that of a driver bringing a customer to his requested destination, and that of a driver searching or waiting for a new customer. It can be seen that these behaviors produce much different patterns of trajectories. Since the Beijing dataset contains occupancy data, one of these patterns of behaviors can be easily isolated. Thus, the first step in cleaning the data was to segment the data based on occupancy status and filter out segments in which the vehicle was not occupied by a customer.

The next step consisted of removing trajectories that seemed to be missing data. These are identifiable by large gaps in the timestamp of adjacent entries. A maximum delta of 100 seconds was chosen based on the distribution (Fig. 7), and trajectories with deltas larger than 100 seconds were removed.

At this point, the dataset was narrowed down to occupied trajectories with relatively consistent time deltas. In order to grid the data, the gradient of the surface of the earth with respect to latitude and longitude coordinates must be taken into account. The euclidean distance corresponding to 1 degree of longitude, for instance, is dependent upon latitude. That is, 1 degree of longitude is /textitlonger at the equator than it is at the north pole. Because of

this, a projection of the grid onto the surface of the Earth is required. Such a projection creates a grid with squares that are equal in euclidean size when the curvature of the earth is taken into account. In this same process, a grid size and granularity was chosen at 20km x 20km and 64 squares x 64 squares based on the distributions of lat/lon. Then, in the process of converting lat/lon to grid cell coordinates, any trajectories containing coordinates outside of the defined 20km x 20km bounding box were filtered out.

At this point, trajectories were gridded, filtered, and cleaned. However, many had duplicate coordinates in adjacent data entries. For example, if a taxi was in heavy traffic, they may be in the same 3.2km x 3.2km grid cell for several consecutive samples. Duplicate consecutive coordinates were removed, leaving a clear, concise trajectory through the city. Then, for uniformity and to make the data fit in each GAN model, trajectories of length 32 were selected. This set has a cardinality of about 4000 trajectories.

3.1.4 Data Engineering

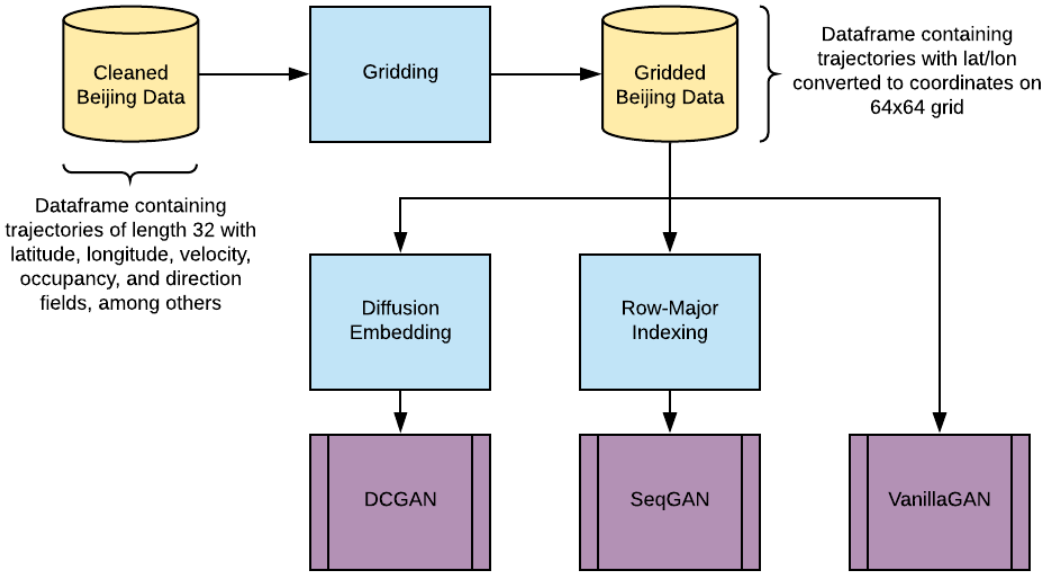

Figure 11: *The process of engineering the dataset to fit in each GAN model.*

Gridding As previously mentioned, the selected region of interest was the 400km^2 bounding box 20km in each direction from the center of the city. After this region was selected and the out of bound segments were removed, the region was divided into a 64 x 64 grid, resulting in grid lengths of approximately

0.32 km. This grid length is reasonable, given that the average block size in the gridded blocks of Beijing is 0.33 km [28]. This grid mapping reduced the overall dimensionality and complexity of the data and allowed for the effective future use of convolutional neural networks for the GAN. Prior to this transformation the data was a sequence of $(latitude, longitude, timestamp)$ tuples, along with other attributes such as direction and velocity. Through this grid transformation, generally, much shorter sequence of cell indices (x, y) were obtained. This grid representation can also be considered as a two-dimensional grid graph, although it is a rather degenerate case of a road network. Most grid structured cities are also aligned with the cardinal directions. Care must be taken with this approximation, especially on cities that have grid structures that are not aligned with the cardinal directions such as Manhattan or Toronto, but this approximation did not pose a problem with the gridded Beijing Taxi data. After the gridding partition, any duplicate visits to the same grid cell were removed, meaning that if a taxi remained idle or only moved within a single grid cell between samples, then the subsequent identical grid entries would be removed.

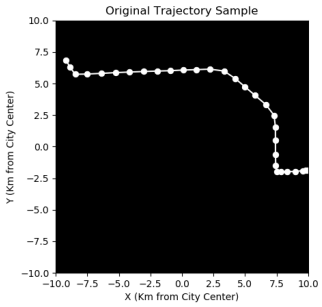

Figure 12: *Sample of vehicle trajectory before gridding*

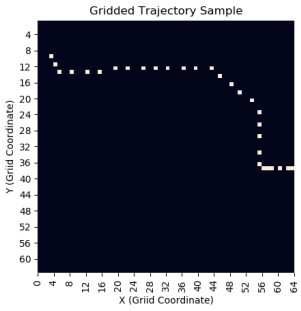

Figure 13: *Same sample of vehicle trajectory after gridding.*

Diffusion Embedding The grid segmentation of the data resulted in a 3-dimensional representation of the data: its x-coordinate in the grid, its y-coordinate in the grid, and the time step in the trajectory where it lies in that grid cell. In order to reduce the dimensionality of the data from 3 dimensions to 2 dimensions, we developed a novel embedding of the trajectory across the grid cells through a linear diffusion function from end to start for the trajectory. This embedding was created through the application a linear diffusion function over the sequence of grids. For example, if the grid was a 4x4 grid and there were 5 time steps for the trajectory, the original gridded representation for the sequence [(0,2), (1,2), (1,3), (2,3), (3,3)], where each entry is of the form (x,y), would be:

```
[[[0 0 1 0]  [[0 0 0 0]  [[0 0 0 0]  [[0 0 0 0]  [[0 0 0 0]
  [0 0 0 0]   [0 0 1 0]   [0 0 0 1]   [0 0 0 0]   [0 0 0 0]
```

```
[0 0 0 0]   [0 0 0 0]   [0 0 0 0]   [0 0 0 1]   [0 0 0 0]
[0 0 0 0]], [0 0 0 0]], [0 0 0 0]], [0 0 0 0]], [0 0 0 1]]]
```

The diffusion embedding for the above gridded sequence would be the following:

```
[[0  0  1  0]
 [0  0 .8 .6]
 [0  0  0 .4]
 [0  0  0 .2]]
```

With this embedding of the trajectory, we can now leverage GAN models, such as DCGAN, that learn 2-dimensional spatial representations of the data. These convolutional GANs are often used on image data, but the trajectory can now be interpreted as an image where the pixel values are all zeros except for those that the trajectory pass through which have the value determined by the linear diffusion embedding of their time step. The models employed were tested with both this diffusion embedding, along with a simple 2-d projection of the grid cells that were visited, removing any notion of time from the complexity.

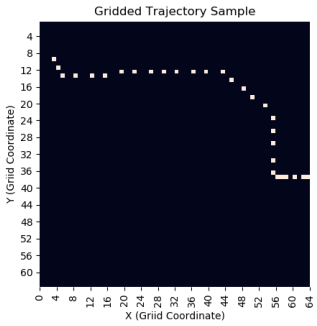

Figure 14: *Sample gridded trajectory.*

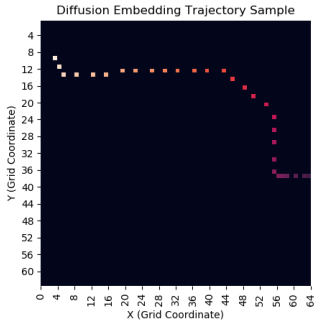

Figure 15: *Same sample after application of diffusion embedding.*

3.2 Evaluation Metrics

KL Divergence Kullback-Leibler Divergence (KL Divergence) is a way of comparing an observed distribution to a known reference distribution. In the context of information theory, this difference can be described as the *relative entropy* of our observed distribution with respect to the reference distribution [15]. This is particularly interesting, as this value can be thought of as the difference between two distributions. If we apply this to the distribution of points in a trajectory about our geographic region, we can do this comparison between a distribution of generated points and our known distribution of real points.

4	4	4	4	4	4	4
4	1	1	1	1	1	4
4	1	0	0	0	1	4
4	1	0	1	0	1	4
4	1	0	0	0	1	4
4	1	1	1	1	1	4
4	4	4	4	4	4	4

Table 2: *The number in each cell represents the baseline penalty if that cell appears after the center cell in a generated trajectory.*

Total Hops Heuristic As mentioned above, we have defined a heuristic to help us quantitatively evaluate our generated data. This heuristic evaluates how well our generator can capture the locality of consecutive points in a trajectory. In our real dataset, ideally there will be no occasions in which two consecutive points jump over a cell in our defined grid. Below, we label the number of cells that would be skipped for a coordinate following the center cell. We have assigned a value of 1 for two consecutive points being in the same cell, as we don't want repeat coordinates either.

Using this concept, we defined a function to accumulate a penalty for a trajectory based on the number of skipped cells between adjacent points. If we let $p =$ the penalty for a trajectory, over a trajectory with length n we can define this heuristic as follows:

$$p = \sum_{i=1}^{n-1} \left[\max((x_{i+1} - x_i), (y_{i+1} - y_i)) - 1\right]^2$$

Probability Distribution of Top-N We can look at the probability distribution for the 'top n' source and destination coordinates, and compare that of the training data with our generated data. Once we have 2 distributions, we can plot them and analyze visually, but we can also then run KL-divergence on the distributions to obtain another quantitative metric for our data. This metric will evaluate the geographic similarities between our generated data and real life data. This metric can also be utilized for grid-based approaches via a 2-dimensional density plot of source and destination coordinates.

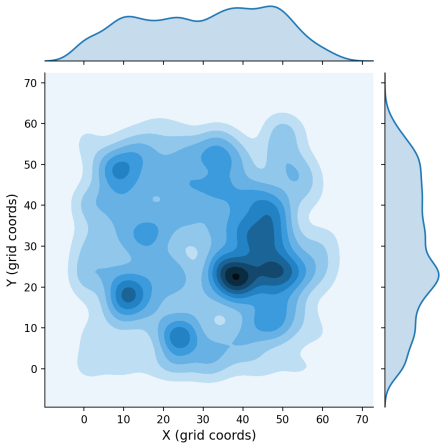

Figure 16: *Distribution of source coordinates of Beijing Dataset*

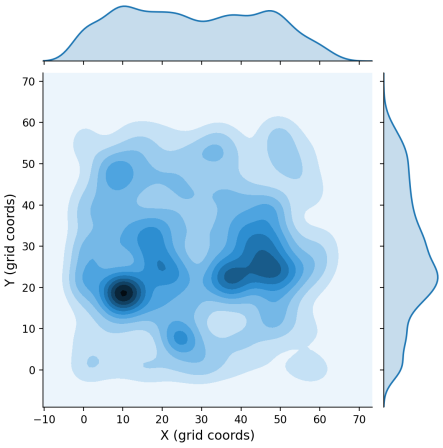

Figure 17: *Distribution of destination coordinates of Beijing Dataset*

3.3 Experimentation

3.3.1 Vanilla GAN

Structure and Implementation For the purpose of getting baseline data, we have found a very simple GAN which uses a few fully-connected layers and leverages pytorch as a backend [18]. We have altered the discriminator of this GAN to have three layers starting with 64 perceptrons, and decreasing down to 1 output. We employed a leaky ReLu activation function on the first two layers and a sigmoid on the last. Sigmoid is appropriate for the final layer of the discriminator because it maps all real domain values to a value between 0 and 1.

We have similarly edited the structure of the generator model. Our generator now has 4 fully connected layers, starting with 4 perceptrons (the length of our starting noise vector z) and increasing to 64 (the size of our input trajectories). We leverage LeakyReLu for the first three layers and then a straight up ReLu for the final layer. ReLU is appropriate for generating our data because our output values always need to be within $[0, 64]$. ReLu functions output a positive value for all real valued domain.

Results and Drawbacks The data we generated from the Vanilla GAN are just what we expected them to be - underwhelming. As it's a baseline model, it points out the fact that our model and data both need to be tailored to one another in order to achieve any profound results. We are able to train the Vanilla GAN in just over a few minutes, since it was a shallow model, and we have generated 65536 "trajectories."

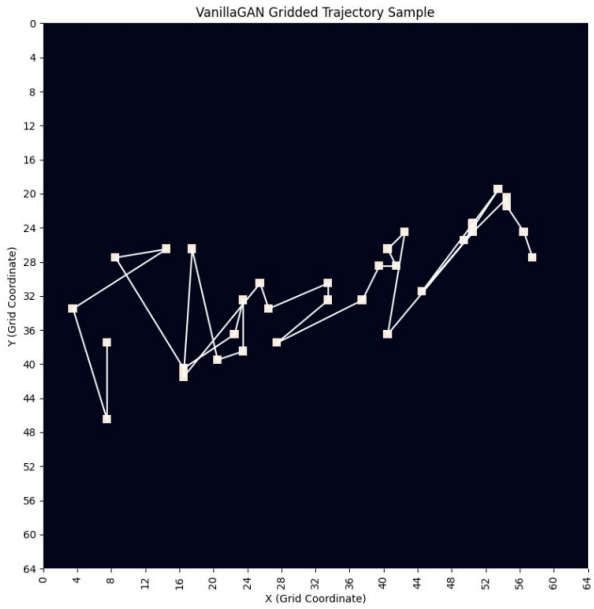

Figure 18: *A sample "trajectory" generated from VanillaGAN.*

Using a random sample of 10,000 points we obtain the KL divergence values in Table 4. This shows us that VanillaGAN learned the overall distribution of points better than SeqGAN which is not surprising as SeqGAN had to re-learn the spatial embedding that was provided directly to VanillaGAN.

Dataset	KL Divergence
Real Data	0
VanillaGAN	1.3485484037563962

Table 3: KL Divergences. Values closer to 0 indicate a more accurate spatial distribution.

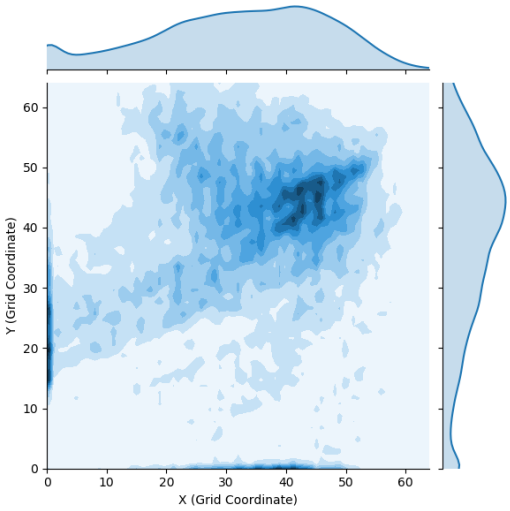

Figure 19: VanillaGAN's distribution of coordinates.

Using our total hops heursistic we have created some illustrations showing the quality of our data. From these graphics, we can conclude several things. The first thing that stands out is that distribution of the values for this heuristic of the generated data is much higher than that of our real data, meaning that the model is quite unsuccessful in capturing this trend in our training data. Secondly, we notice that there are occasions in our training data in which consecutive points skip cells. This tells us that we need to revisit our data cleaning code, because some faulty trajectories are slipping through the cracks.

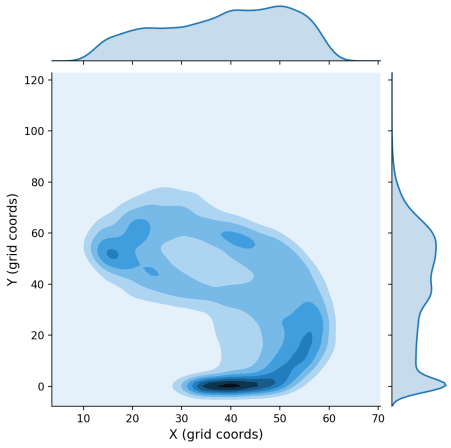

Figure 20: *Distribution of source coordinates generated by VanillaGAN*

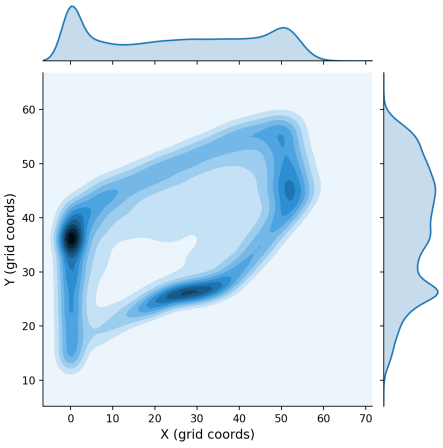

Figure 21: *Distribution of destination coordinates generated by VanillaGAN*

3.3.2 SeqGAN

Structure and Implementation We decided to tackle SeqGAN due to the fact that its intended use is for sequential data. We utilized an existing SeqGAN implementation without structural modification to attempt to generate trajectories.

Data Preprocessing In order for this to work, we had to map our grid cell representation to tokens as SeqGAN uses an embedding and is originally designed for sentences. This is clearly a bad idea as we are throwing out one dimension of our spatial information. Nonetheless, we index the grid via row-major order and hope that SeqGAN will relearn the spatial dependencies.

Results and Drawbacks Compared to DCGAN, the learning was relatively stable. In spite of this, we found that the model was largely unsuccessful in learning spatial distributions of our data. to anything useful and failed to learn that trajectories must go between adjacent or near-adjacent cells. This is unsuprising given the extreme discontinuities in the function from token id to grid cell.

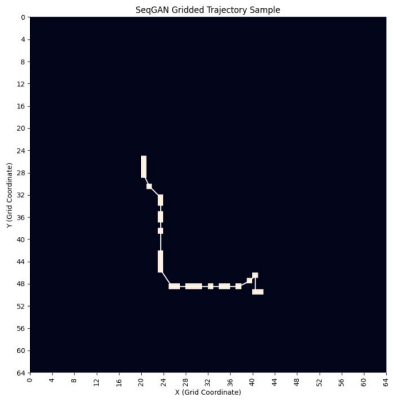

Figure 22: *A sample trajectory generated from VanillaGAN.*

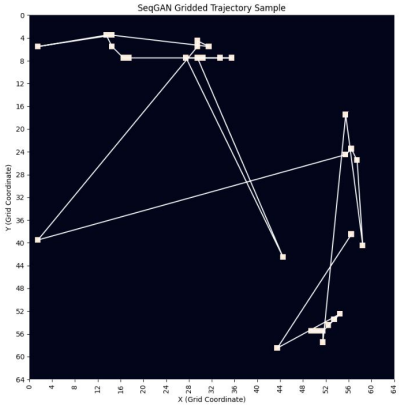

Figure 23: *A sample "trajectory" generated from VanillaGAN.*

Dataset	KL Divergence
VanillaGAN	1.3485484037563962
SeqGAN	1.9052103561695533

Table 4: KL Divergences. Values closer to 0 indicate a more accurate spatial distribution.

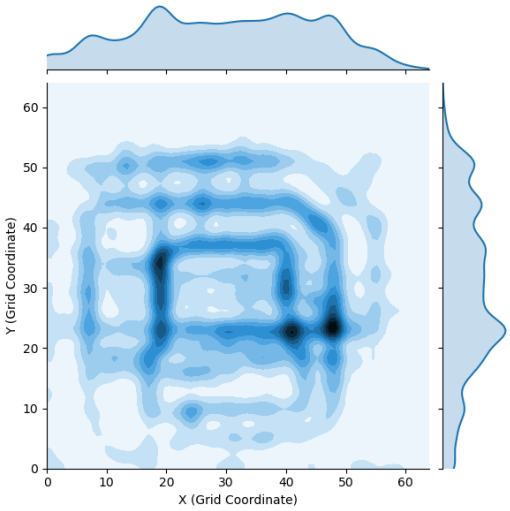

Figure 24: SeqGAN's spatial distribution of coordinates.

As can be seen above, the source and destination coordinate distributions show promise. Some of the hotspots shown can be attributed to similar hot spots in the actual data's distributions. This shows that to some degree, SeqGAN was able to learn patterns in the data. It is worth noting that SeqGAN tends to end sequences in a coordinate of (0, 0), or several of them, so this destination distribution was created by trimming all trailing zeros off of sequences.

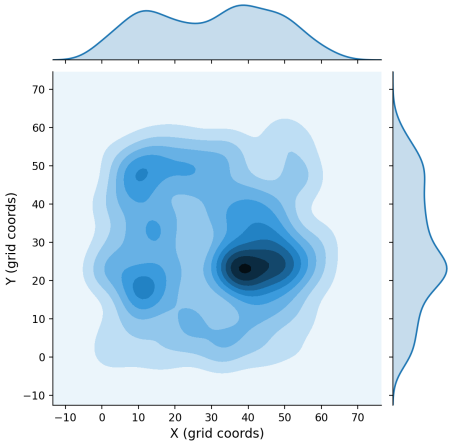

Figure 25: Distribution of source coordinates generated by SeqGAN

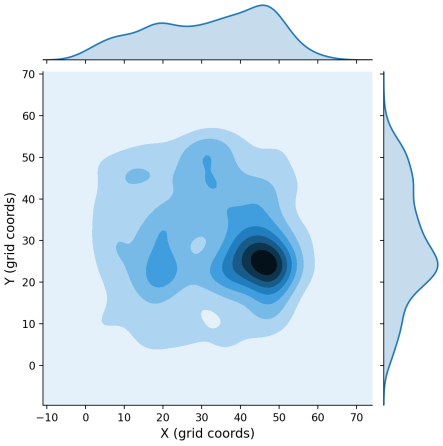

Figure 26: Distribution of destination coordinates generated by SeqGAN

3.3.3 DCGAN

Structure and Implementation Throughout the process of training DCGAN and generating data, we encountered several problems. The first of these problems was an imbalance between the discriminator and generator, resulting in discriminator completely winning, providing no gradient to the generator. In order to rectify this, we had to actually make the discriminator worse by decreasing the number of features in the discriminator and decreasing the learning rates of both nets significantly to increase stability. Once we were able to get the discriminator and generator balanced enough to maintain a relatively steady loss throughout the entire training process, we found that we were experiencing mode collapse, resulting in our generated data only representing a subset of acceptable data. The trajectories we generated seemed to all follow generally the same pattern and start and end at roughly the same points.

Results and Drawbacks Another issue we experienced is that the model could not learn the diffusion pattern we applied to our data. The outputted data sometimes resembled a trajectory, however the values within each grid cell did not formulate a trajectory in the same way we diffused our data. This tells us that our model was able to learn roughly what the shape of a trajectory should look like, but it couldn't learn our diffusion. The implications of the mode collapse were that the resulting KL divergence value and distributions were extremely skewed.

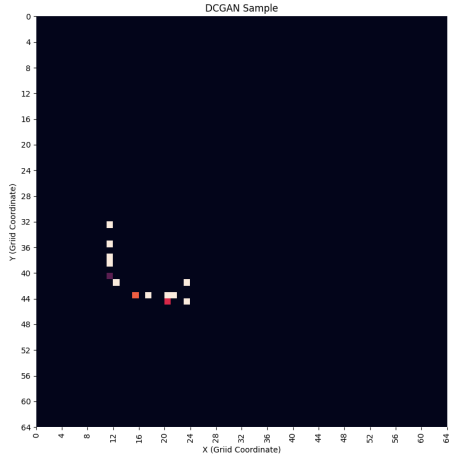

Figure 27: A relatively clean trajectory generated by DCGAN

We attribute our difficulties in training an accurate model to several different factors. We think the fact that the gradient of the data we inputted was quite

sparse is causing our model a lot of trouble in learning the underlying patterns. One trajectory from our diffused data consists of a matrix of mostly zeros, and a few values between 0 and 1 (or -1 and 1). This kind of gradient is very difficult for a neural network to learn, which we believe explains the mode collapse as well as why our generator couldn't learn our diffusion pattern.

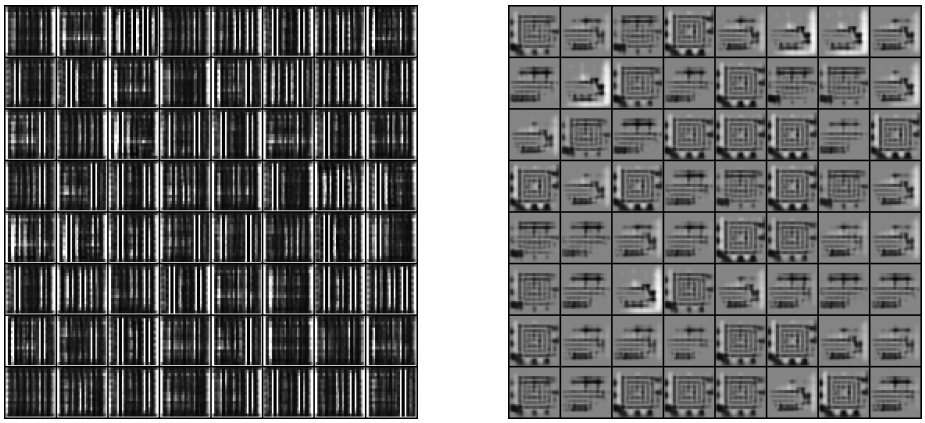

Figure 28: Two very poor batches of data generated by DCGAN.

4 Conclusion and Future Work

4.1 Conclusion

Our research affirmed how adversarial models are notoriously difficult to train, and we faced significant stability problems from all of the models we trained. Furthermore, even if adversarial stability is achieved, it's difficult to avoid mode collapse. Although we have created some trajectories that appear realistic, more advanced models are required to adequately capture the long-term spatio-temporal dependencies that exist in trajectory data. Some of the trajectories that we were able to generate did, however, show promising results. Although there were some realistic trajectories, and the SeqGAN model was able to decently capture the spatial distributions for the city, in order for the output of these models to be utilized by another user, like a transportation or traffic planner, they would need to produce a sufficient amount of batches such that the spatial distribution converges.

4.2 Future Work

We found that grid based representations of trajectories are inherently limited. It would make much more sense to model them as graphs, as not all city road networks have a defined, gridded architecture. Modeling trajectories in a graphical manner would be a more general solution, as road intersections would be the nodes in the graph, and the roads themselves would be the edges. However,

we were unable to find any generative models in the literature that create walks over graphs, nor could we find any large and reliable graph-based datasets. It may be possible to create such a model by adapting existing models such as GRETEL [8] and CSSRNN [29] alongside the use of map matching algorithms [20].

A similar reframing of the trajectory generation problem would be to view the generation process as a degenerative case of sequence imputation from only two points. A sequence imputation model could potentially solve the problem of trajectory generation and refinement of low-frequency trajectories simultaneously. It may be possible to adapt NAOMI [19] for this task.

All the models we utilized are limited to a single city topology, but it may be possible to adapt City2City [30] to create a model that generalizes over city topologies. This could allow researchers to test the effects of alterations to existing topologies or aid in the process of planning new ones.

With sufficient data, it may also be possible to create a model that generates coherent trajectories for multiple vehicles at the same time which could facilitate traffic models.

References

[1] Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning.

[2] M. Ehsan Abbasnejad, Qinfeng Shi, Iman Abbasnejad, Anton van den Hengel, and Anthony Dick. Bayesian Conditional Generative Adverserial Networks. 6 2017.

[3] Jaume Barceló, editor. *Fundamentals of Traffic Simulation*, volume 145 of *International Series in Operations Research & Management Science*. Springer New York, New York, NY, 2010.

[4] Lorenzo Bracciale, Marco Bonola, Pierpaolo Loreti, Giuseppe Bianchi, Raul Amici, and Antonello Rabuffi. CRAWDAD dataset roma/taxi (v. 2014-07-17). Downloaded from https://crawdad.org/roma/taxi/20140717, 7 2014.

[5] Francesco Camastra and Alessandro Vinciarelli. Markovian models for sequential data. In *Advanced Information and Knowledge Processing*, number 9781447167341, pages 295–340. Springer London, 2015.

[6] China Internet Information Center. Driving in China, May 2008.

[7] Christopher Olah. Understanding LSTM Networks, 2015.

[8] Jean-Baptiste Cordonnier and Andreas Loukas. Extrapolating paths with graph neural networks. *CoRR*, abs/1903.07518, 2019.

[9] Wenhao Ding, Wenshuo Wang, and Ding Zhao. A New Multi-vehicle Trajectory Generator to Simulate Vehicle-to-Vehicle Encounters. 9 2018.

[10] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. Technical report.

[11] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks. 3 2018.

[12] Igor Balteiro. GPS data from Rio de Janeiro buses, 2019.

[13] James Colyar. Next Generation Simulation Vehicle Trajectories, 2017.

[14] Lin Zhang Jing Lian. One-month beijing taxi gps trajectory dataset with taxi ids and vehicle status.

[15] Will Kurt. Kullback-leibler divergence explained, May 2017.

[16] Jiachen Li, Hengbo Ma, and Masayoshi Tomizuka. Conditional Generative Neural System for Probabilistic Trajectory Prediction. 5 2019.

[17] Jiachen Li, Hengbo Ma, Wei Zhan, and Masayoshi Tomizuka. Coordination and Trajectory Prediction for Vehicle Interactions via Bayesian Generative Modeling. 5 2019.

[18] Erik Linder-Norén. PyTorch-GAN, June 2019.

[19] Yukai Liu, Rose Yu, Stephan Zheng, Eric Zhan, and Yisong Yue. NAOMI: Non-Autoregressive Multiresolution Sequence Imputation. 1 2019.

[20] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. Map-matching for low-sampling-rate gps trajectories. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 352–361, 2009.

[21] Sobhan Moosavi, Behrooz Omidvar-Tehrani, R. Bruce Craig, and Rajiv Ramnath. Annotation of Car Trajectories based on Driving Patterns. 5 2017.

[22] Bruno A Olshausen. Bayesian probability theory and generative models. Technical report, 2006.

[23] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2016.

[24] Adrian E Raftery, S Lewis, JM Bernardo, JO Berger, AP Dawid, and AFM Smith. Bayesian statistics. *Oxford Sci. Publ*, pages 323–349, 1992.

[25] Ruslan Salakhutdinov. Learning deep generative models. *Annual Review of Statistics and Its Application*, 2:361–385, 2015.

[26] Alexander Shapiro. Monte carlo sampling methods. *Handbooks in operations research and management science*, 10:353–425, 2003.

[27] Sanjib Sharma. Markov Chain Monte Carlo Methods for Bayesian Data Analysis in Astronomy. *Annual Review of Astronomy and Astrophysics*, 55:1–49, 2017.

[28] Victor FS Sit. *Beijing: The nature and planning of a Chinese capital city*, volume 38. * Belhaven Press, 1995.

[29] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. Modeling trajectories with recurrent neural networks. IJCAI, 2017.

[30] Takahiro Yabe, Kota Tsubouchi, Toru Shimizu, Yoshihide Sekimoto, and Satish V Ukkusuri. City2city: Translating place representations across cities. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 412–415, 2019.

[31] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. 9 2016.

[32] Chaoyun Zhang, Xi Ouyang, and Paul Patras. ZipNet-GAN: Inferring Fine-grained Mobile Traffic Patterns via a Generative Adversarial Neural Network. 11 2017.

[33] Yu Zheng. T-drive trajectory data sample, August 2011. T-Drive sample dataset.

[34] Huiyu Zhou and Kotaro Hirasawa. Spatiotemporal traffic network analysis: technology and applications. *Knowledge and Information Systems*, 60(1):25–61, 7 2019.

[35] Kang Zhou, Shenghua Gao, Jun Cheng, Zaiwang Gu, Huazhu Fu, Zhi Tu, Jianlong Yang, Yitian Zhao, and Jiang Liu. Sparse-GAN: Sparsity-constrained Generative Adversarial Network for Anomaly Detection in Retinal OCT Image. 11 2019.