GENERATIVE ADVERSARIAL NETWORKS FOR THE GENERATION OF REALISTIC MOBILITY TRACES IN URBAN SCENARIOS

A PROJECT REPORT

Submitted by

AKILESH K	**(312217106014)**
GOKULA KRISHNAN S K	**(312217106044)**
HARIHARAN K	**(312217106051)**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

SRI SIVASUBRAMANIYA NADAR COLLEGE OF ENGINEERING, KALAVAKKAM

ANNA UNIVERSITY : CHENNAI 600 025

JUNE 2021

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"GENERATIVE ADVERSARIAL NETWORKS FOR THE GENERATION OF REALISTIC MOBILITY TRACES IN URBAN SCENARIOS"** is the bonafide work **AKILESH K (312217106014), GOKULA KRISHNAN S K (312217106044), HARIHARAN K (312217106051)** who carried out the project work under my supervision.

SIGNATURE

DR. S RADHA
HEAD OF DEPARTMENT
PROFESSOR
Department of ECE
Sri Sivasubramaniya Nadar
College of Engineering,
Kalavakkam - 603110.

SIGNATURE

DR. PRABAGARANE N
SUPERVISOR
ASSOCIATE PROFESSOR
Department of ECE
Sri Sivasubramaniya Nadar
College of Engineering,
Kalavakkam - 603110.

Submitted for the examination held on ……

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our sincere gratitude to our beloved Principal, **Dr. V E Annamalai** for the facilities provided to us during the project.

We take this opportunity to express our profound gratitude and deep regards to **Dr. S Radha**, HOD of ECE Department for her exemplary guidance, monitoring and constant encouragement throughout the course of this project.

We also take this opportunity to express a deep sense of gratitude to **Dr. Prabagarane N**, Associate Professor, ECE Department at Sri Sivasubramaniya Nadar College of Engineering, **Giacomo Morabito**, Professor at the University of Catania and **Sam Mertens**, University of Catania for their guidance, valuable information and cordial support which helped us in completing this task through various stages.

We are indebted to our panel members, **Dr. N Venkateshwaran,** Professor, ECE Department at Sri Sivasubramaniya Nadar College of Engineering, and **Dr. B Partibane**, Associate Professor, ECE Department at Sri Sivasubramaniya Nadar College of Engineering for their suggestions which made us improve our project. We are grateful for their guidance during the period of our project.

AKILESH K

GOKULA KRISHNAN S K

HARIHARAN K

ABSTRACT

Smart cities have been recognized as a promising research focus around the world. To realize smart cities, computation and utilization of big data are key factors. More specifically, exploring the patterns of human mobility based on large amounts of multi-source data plays an important role in analyzing the formation of social-economic phenomena in smart cities. Generative Adversarial Network (GAN) being one of the fast growing research fields can be used with big data to produce wonderful collaborative results in mobility traces. This project aims at the possibility of using GANs to create realistic mobility traces. GANs are one of the deep generative machine learning models that uses game theory. GANs accept an input vector and produce realistic data. The discriminator and generator neural networks play a major role in producing accurate results. The input vector could have information regarding climate, day of the week, time, etc. A user-interface is to be implemented through which these input values can be given to the GANs. Thus, the aim is to train and execute GANs for the generation of realistic mobility traces in urban scenarios.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
C GAN	Conditional Generative Adversarial Network
CDF	Cumulative Distribution Function
DC GAN	Deep Convolutional Generative Adversarial Network
GAN	Generative Adversarial Network
MNIST	Modified National Institute of Standards and Technology
PDF	Probability Density Function
RMSProp	Root Mean Square Propagation
SUMO	Simulation of Urban MObility
T GAN	Tabular Generative Adversarial Network
W GAN	Wasserstein Generative Adversarial Network

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

The past several years have witnessed a burgeoning development of computer science and data accumulation. Artificial intelligence (AI) is becoming a thriving field with a great number of meaningful applications and valuable research topics. In the AI community, machine learning exerts a huge impact on different aspects of our daily life (Murphy, 2012). Generally, according to whether the data set is labeled or not, machine learning algorithms can be divided into two categories, supervised and unsupervised learning .

For the supervised learning, a dataset with diverse features is required, and each example in the dataset must be labeled. However, unsupervised learning does not require such a labeled dataset. In case of supervised learning, it is difficult to collect or annotate labels automatically. Hence, researchers pay more attention to unsupervised learning. In the task of unsupervised learning, generative models are one of the most promising technologies (Pan et al., 2019).

Generative Adversarial Network is an approach to generative modeling using deep learning methods, such as convolutional neural networks. Vehicular networks have received much attention in recent years as they have emerged as one of the leading data communication solutions for smart cities. At the same time, the popularization of sensing devices has enabled the acquisition of a vast amount of vehicular mobility data (mobility traces). In this sense, a recent trend is to use mobility traces to extract hidden knowledge and apply it to improve solutions for vehicular networks (Celes, 2020).

GAN consist of two parts: generators and discriminators. The generator model produces synthetic data from random noise, sampled using a distribution. This synthetic data along with real data from a training dataset are fed to the discriminator, which attempts to distinguish between the two. Both the generator and discriminator improve in their respective abilities until the discriminator is unable to tell the real data from the synthesized data with better than 50% accuracy.

GANs train in an unsupervised fashion, meaning that they infer the patterns within datasets without reference to known, labeled, or annotated outcomes. The discriminator's work is to inform the generator every time the discriminator correctly identifies a synthesized data. This tells the generator how to tweak its output so that it might be more realistic in the future.

1.2 MOTIVATION

Vehicular networks have received much attention in recent years as they have emerged as one of the leading data communication solutions for smart cities.

The potential economic value of open data in transport is in the order of billions of dollars. It has been estimated that a lot of money can be saved with innovation in commuter decision-making and the growth of new mobility businesses.

GANs being one of the growing research areas can also be applied to the data communication solutions for vehicular networks. Moreover, predicting realistic mobility traces is challenging. Understanding mobility, being it from pedestrians or any other moving object is practical and insightful.

Generative models are expected to generate realistic data across a range of problem domains. GANs are an exciting and rapidly changing

field, which satisfies the above expectation. So we decided to use GANs to generate realistic traffic data.

1.3 OBJECTIVE

The main objective of our project work is to build and optimize a Generative Adversarial Network to generate realistic mobility traces. The objective of the project is summarised below:

1. To build a discriminator that distinguishes real and fake data.
2. To build a generator that generates fake data in a way that discriminator finds it difficult to distinguish between real and fake data.
3. To build a GAN with more accuracy and less generator loss and discriminator loss.

1.4 OUTLINE OF REPORT

Chapter 2 gives a brief overview of the literature survey undertaken for this work.

Chapter 3 gives an introduction to Generative Adversarial Networks, its algorithm and the involvement of various aspects of GANs.

Chapter 4 explains the various components of GAN. The significance of all the components of GAN is discussed.

Chapter 5 provides a detailed description on the implementation of GAN using PyTorch.

Chapter 6 discusses the metrics used for evaluating the accuracy and performance of the GAN.

Chapter 7 concludes the report and gives directions regarding future work.

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

In this chapter, a detailed review of previous work in the field of GANs is presented. The papers discussed here have used different variations of Generative Adversarial Networks on various types of data inputs.

2.2 LITERATURE REVIEW

Ian J. Goodfellow, (2014) proposed the idea of GAN. This paper describes the model of the generative system. The algorithms and theoretical results are discussed. The generator and discriminator were trained to reach Nash's equilibrium. Also, few experiments are done with the objective of producing images. They include generating numerical digits and human faces. Finally, the paper concludes by describing the advantages and disadvantages of GANs. From this paper, we can see that GANs have better computational and statistical advantages over Boltzmann machines. We can conclude that GANs are a good way to generate the data we need.

Z. Pan et al. and Y. Zheng, (2019) produced detailed research on the working of GANs and different approaches of its implementation. The paper begins by explaining the principle of GANs. GANs make use of game theory to train the discriminator and generator networks. Few evaluation metrics and training tricks are discussed. GANs can be used in computer vision, face synthesis, texture synthesis, natural language processing and much more domains. This motivates us to implement GAN in our use case.

Lei Xu and Kalyan Veeramachaneni, (2018) synthesized Tabular Data using Generative Adversarial Networks. This work points out the difficulties in implementing GAN for generating tabular data. Tabular data usually contains a mixture of discrete and continuous columns. Building such a model is a non-trivial task. Continuous columns may have multiple modes, while discrete columns are sometimes imbalanced, making modeling difficult. Tabular GAN (TGAN), a GAN-based model for generating relational tables containing continuous and discrete variables, is proposed. Numerical variables are clustered to deal with the multi-modal distribution for continuous features. Noise and Kullback–Leibler divergence are added into the loss function to effectively generate discrete features. GANs can effectively capture the correlations between features and are more scalable for large datasets.

Spohn et al., (2019) explains about different metrics used to analyse the mobility traces. Various mobility models are proposed and the details regarding the best fit is discussed. There are many possible mobility patterns, depending on who or what are the mobile entities (e.g. pedestrians, cars, buses). One important input for mobile network simulations regards the movement of mobile entities, which could be fed into the simulator or computed in real time. However, due to the lack of real mobility traces, synthetic mobility models came first, making them the standard when it comes to defining movement properties.

A. Brock et al., (2019) aimed to train Generative Adversarial Networks at the largest scale, and study the instabilities specific to such scale. We find that applying orthogonal regularization to the generator renders it amenable to a simple truncation trick, allowing fine control over the trade-off between sample fidelity and variety by reducing the variance of the generator input. Generative Adversarial Networks trained to model natural images of multiple categories highly benefit from scaling up, both

in terms of fidelity and variety of the generated samples. As a result, the models set a new level of performance among, improving on the state of the art by a large margin. Also they presented an analysis of the training behavior of large scale GANs, characterized their stability in terms of the singular values of their weights, and discussed the interplay between stability and performance.

M. Mirza and S. Osindero, (2014) introduces a conditional version of Generative Adversarial Network, which can be constructed by simply feeding the data, y, we wish to condition on both the generator and discriminator. We can see that this model can generate MNIST digits conditioned on class labels. It is also illustrated how this model could be used to learn a multi-modal model, and it provides preliminary examples of an application to image tagging in which it demonstrates how this approach can generate descriptive tags which are not part of training labels. In an unconditioned generative model, there is no control on modes of the data being generated. However, by conditioning the model on additional information it is possible to direct the data generation process.

2.3 SUMMARY

In this chapter, existing research papers related to the field of deep learning and GANs are presented. These research papers provided us with the basic ground to develop and optimize the GAN architecture for generating realistic mobility traces. The algorithm and applications of Generative Adversarial Networks will be discussed in the next chapter.

CHAPTER 3

GENERATIVE ADVERSARIAL NETWORKS

3.1 INTRODUCTION

Generative Adversarial Network is an approach to generative modeling using deep learning methods, such as convolutional neural networks. Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

3.2 ALGORITHM

Generative Adversarial Network is an algorithmic architecture that uses two neural networks, pitting one against the other in order to generate new, synthetic instances of data that is very much similar to the real data. They are used widely in image generation, video generation and voice generation.

Game theory, a branch of applied mathematics that provides tools for analyzing situations in which parties, called players, make decisions that are interdependent. This interdependence causes each player to consider the other player's possible decisions, or strategies, in formulating strategy. A solution to a game describes the optimal decisions of the players, who may have similar, opposed, or mixed interests, and the outcomes that may result from these decisions.

Based on the game theory there are two networks in GANs. One is generator and the other is discriminator. The role of the generator is to create as realistic data as possible to deceive the discriminator. The role

of the discriminator is to distinguish fake samples from real ones. In this case, we can train both models using backpropagation.

The outer loop of the algorithm involves iterating over steps to train the models in the architecture. One cycle through this loop is not an epoch, it is a single update consisting of specific batch updates to the discriminator and generator models. An epoch is defined as one cycle through a training dataset, where the samples in a training dataset are used to update the model weights in mini-batches. In the case of GAN, the number of training iterations must be defined based on the size of the training dataset and batch size. By calculating the number of batches per epoch and number of epochs, the total number of training iterations can be determined. One iteration of training results in possibly multiple updates to the discriminator and one update to the generator, where the number of updates to the discriminator is a hyperparameter that is set to one.

Updating the discriminator model involves the following steps: First, a batch of random points from the latent space must be selected for use as input to the generator model to provide the basis for the generated or ‘fake’ samples. Then a batch of samples from the training dataset must be selected for input to the discriminator as the ‘real’ samples. Next, the discriminator model must make predictions for the real and fake samples and the weights of the discriminator must be updated proportional to how correct or incorrect those predictions were. These can be provided as arguments to the training function. Points must be generated from the latent space and the generator model in its current form is used to generate some fake data. The size of the latent dimension is also provided as a hyperparameter to the training algorithm. The discriminator model must then make a prediction for each of the generated and real images and the weights must be updated. Next, the generator model must be

updated. Again, a batch of random points from the latent space must be selected and passed to the generator to generate fake images, and then passed to the discriminator to classify. The response can then be used to update the weights of the generator model. The discriminator is updated with two batches of samples each training iteration whereas the generator is only updated with a single batch of samples per training iteration (Murphy, 2012).

3.3 APPLICATIONS

GANs have various applications in various domains. There are many types of GANs available today for different purposes. GANs present a way to learn deep representations without extensively annotated training data. These networks achieve learning through deriving backpropagation signals through a competitive process involving a pair of networks. The representations that can be learned by GANs may be used in several applications. GANs have made significant advancements and tremendous performance in numerous applications. The essential applications include semantic image editing, style transfer, image synthesis, image super-resolution and classification. Following are its different variants and potential application in various domains. Some of the famous GANs include Deep Convolutional GAN (DC GAN), Conditional GAN (C GAN), Tabular GAN (T GAN), Wasserstein GAN (W GAN), etc. (Alqahtani et al., 2019). Some of the wide applications of GANs include:

1. Generate Examples for Image Datasets
2. Generate Photographs of Human Faces
3. Generate Realistic Photographs
4. Generate Cartoon Characters
5. Image-to-Image Translation
6. Text-to-Image Translation

7. Semantic-Image-to-Photo Translation
8. Face Frontal View Generation
9. Generate New Human Poses
10. Photos to Emojis
11. Photograph Editing
12. Face Aging
13. Photo Blending
14. Super Resolution
15. Photo Inpainting
16. Clothing Translation

3.4 SUMMARY

In this chapter, the algorithm of GANs and different applications of GANs were explored. A comprehensive review of the crucial applications of GANs covering a variety of areas, including study of the techniques and architectures used and further the contribution of that respective application in the real world were discussed. In the next chapter, we will discuss the components of GAN.

CHAPTER 4

COMPONENTS OF GAN

4.1 INTRODUCTION

A Generative Adversarial Network has the following two parts: The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator. The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results. The dataset which is one of the inputs of the discriminator is used as a reference for the discriminator to compare with the outputs of the generator to increase the accuracy of the GAN.

4.2 DATASET

The dataset plays a vital role in a GAN. Without a proper dataset the discriminator fails to distinguish between the real and fake data. The dataset chosen for this project is a taxi navigation dataset by ‘Uber Movement’ for Bangalore city. The dataset contains the following information:

1. Source ID
2. Destination ID
3. Time Slot
4. Day
5. Month

The expected output from the generator neural network is to produce a Source ID and Destination ID which cannot be identified as fake data by the discriminator model.

Table 4.1 The sample of the dataset used in the process.

sourceid	dstid	month	day	start_hour	end_hour
142	148	1	1	0	7
145	118	1	1	0	7
143	138	1	1	0	7
141	158	1	1	0	7
144	128	1	1	0	7
141	16	1	1	0	7
140	26	1	1	0	7

Table 4.1 shows the sample of the dataset used in the process. Source ID is the starting point and Destination ID is the end point where the passenger gets out. The month and day tell us the days and months of travel. The start and end hour is the time slot in which the journey takes place.

Since the dataset contains large sets of data, the data is initially split according to months and days and trained on the basis of timeslot. An analysis has been done on each dataset and graphs have been plotted for every dataset for finding the spread of Source ID and Destination ID in the given time slot.

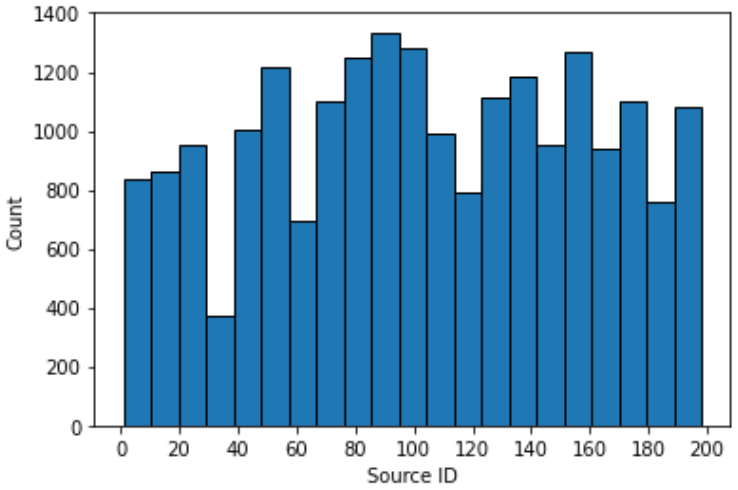

Figure 4.1 The histogram of Source IDs.

Figure 4.1 shows the histogram of Source IDs. The x axis shows the range of Source IDs which varies from 0-200 and the y axis shows the number of taxis starting from that Source ID. Each Source ID and Destination ID has a latitude and longitude connected to it.

4.3 GENERATOR

The principle of the generator is to generate fake data as much as possible to fit the potential distribution of real data, while the principle of discriminator is to correctly distinguish real data from fake data. The input of the generator is a random noise vector (usually a uniform or normal distribution). The noise is mapped to a new data space via generator to obtain a fake sample, which is a multi-dimensional vector.

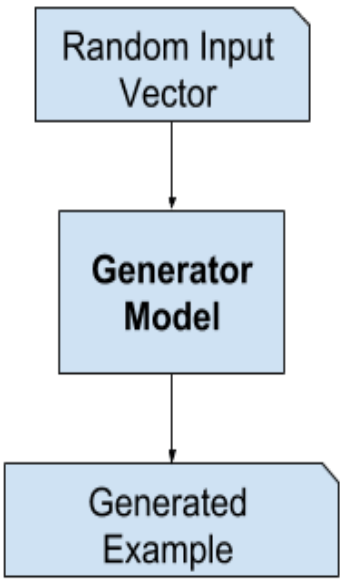

Figure 4.2 The flowchart of the generator model.

Figure 4.2 shows the flowchart of the generator model. The generator model must generate results in such a way that the discriminator finds it difficult to distinguish between the real and generated data.

4.4 DISCRIMINATOR

The discriminator is a binary classifier, it takes both the real sample from the dataset, and the fake sample generated by the generator as the input, and the output of the discriminator represents the probability that the sample is a real rather than a fake. When the discriminator cannot determine whether the data comes from the real dataset or the generator, the optimal state is reached. After the training process, the discriminator model is discarded as we are interested in the generator. Sometimes, the generator can be repurposed as it has learned to effectively extract features from examples in the problem domain. Some or all of the feature extraction layers can be used in transfer learning applications using similar input data.

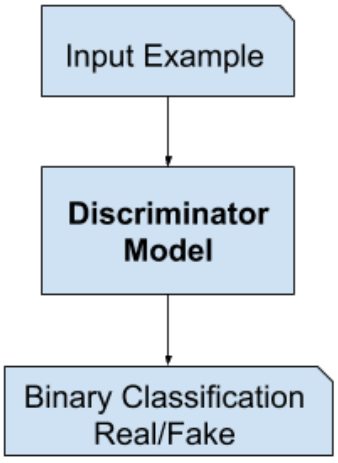

Figure 4.3 The flowchart of the discriminator model.

Figure 4.3 shows how the discriminator model tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it is classifying.

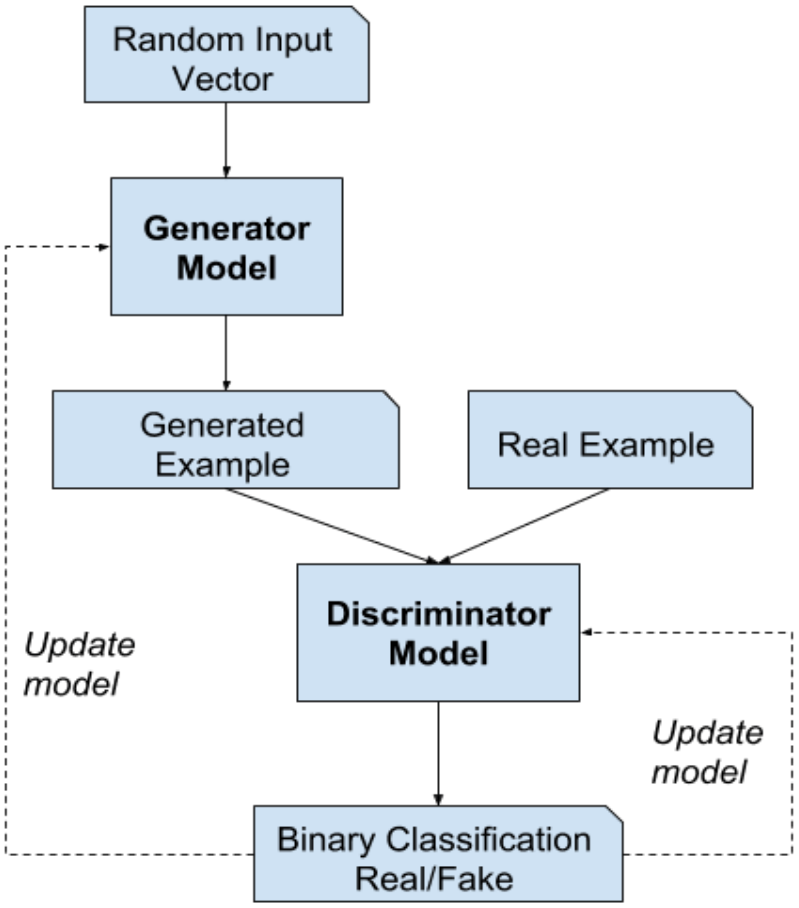

Figure 4.4 The complete flowchart of GAN architecture.

Figure 4.4 shows how the generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

4.5 SUMMARY

In this chapter, the different components of GAN - generator and discriminator are discussed. Dataset to be used for the input for the GAN is also explored. Flowcharts of the General Adversarial Network depicting the operation of every individual aspect of the network are shown. In the next chapter, Implementation of a GAN using PyTorch is explained.

CHAPTER 5

IMPLEMENTATION IN PYTORCH

5.1 INTRODUCTION

There are a bunch of frameworks available for developing neural networks. In this work, PyTorch is used for developing the Generative Adversarial Network. The main advantage is that PyTorch uses dynamic computation graphs, while Tensorflow (or Keras, or Theano) uses static graphs. In Tensorflow, everything one does must operate on the same structures, and it must always be the same computations. With this dynamic approach in PyTorch, one can fully see each and every computation and know exactly what is going on. When the flow of data and the corresponding operations are defined at runtime, the construction of the computational graph happens dynamically.

5.2 DATASET

A class is defined to store the dataset. This class reads a csv file and stores it as a list. This list is then converted to a numpy array and in turn into PyTorch tensors. This class also has a function to return the i th data and also to return the number of samples in the dataset.

5.3 GENERATOR

The first step is to build the generator component. We will start by creating a function to make a single layer/block for the generator's neural network. Each block should include a linear transformation to map to another shape, a batch normalization for stabilization, and finally a non-linear activation function. This enables the neural network to transform the output in complex ways. Now the generator can be built. It will take 3 values:

1. The noise vector dimension
2. The data dimension
3. The initial hidden dimension

Using these values, the generator will build a neural network with five layers/blocks. Beginning with the noise vector, the generator will apply non-linear transformations via the block function until the tensor is mapped to the size of the data to be outputted.

5.4 NOISE

To be able to use the generator, we should create noise vectors. The noise vector has the important role of making sure the data generated from the same class don't all look the same. It can be generated randomly using PyTorch by sampling random numbers from the normal distribution.

5.5 DISCRIMINATOR

The second component to construct in a GAN is the discriminator. As with the generator component, a function is created to build a neural network block for the discriminator.

The discriminator class holds two values: The dimension of the input data and the dimension of the hidden layer. The discriminator will build a neural network with four layers. It will start with the data tensor and transform it until it returns a single number (1-dimension tensor) output. This output classifies whether the input is fake or real.

5.6 LOSS FUNCTIONS

Now, the generator, discriminator, and optimizers can be initialised. Each optimizer only takes the parameters of one particular model, since each optimizer should optimize only one of the models. Adam optimizer is used in this work since it combines the best properties of the Adaptive

gradient descent and RMSprop algorithms.

Functions which calculate the discriminator's loss and the generator's loss are to be created. In this way, the discriminator and generator will know how they are performing and improve themselves. Generator is needed when calculating the discriminator's loss, the function ‘detach()’ is called to ensure that only the discriminator is updated.

5.7 TRAINING

Data is taken in batches of size fifty from the dataset and used to calculate the discriminator loss. This loss is used to update the weights of the discriminator. Now the generator loss is calculated and its weights are updated. The loss is tracked in an array for plotting.

5.8 SUMMARY

In this chapter, the implementation of GAN in PyTorch was discussed. The implementation of the dataset along with the generator, discriminator, noise and loss function were also discussed. How the dataset is partitioned and trained were also explained in this chapter. In the next chapter, the performance analysis and various evaluation metrics used for measuring the performance of the GAN will be discussed.

CHAPTER 6

PERFORMANCE ANALYSIS

6.1 INTRODUCTION

Performance of the GAN is evaluated by training it for one hundred and fifty epochs. Generally, there are no objective ways to evaluate the performance of a GAN model. One cannot calculate an objective error score for the generated data. Instead, data must be subjectively evaluated for quality by a human operator. This means that one cannot know when to stop training without looking at the generated data. In turn, the adversarial nature of the training process means that the generator is changing after every batch, meaning that once more accurate data can be generated, the subjective quality of the data may then begin to vary, improve, or even degrade with subsequent updates.

6.2 EVALUATION METRICS

There are three ways to handle this complex training situation:

1. Periodically evaluate the classification accuracy of the discriminator on real and fake images.
2. Periodically generate a lot of data and save them to file for subjective review.
3. Periodically save the generator model.

All three of these actions can be performed simultaneously in regular intervals, such as every five or 10 training epochs. The result will be a saved generator model for which we have a way of subjectively assessing the quality of its output and objectively knowing how well the discriminator was fooled at the time the model was saved.

Various properties of the distribution of the generated data are compared with that of the real data at various points of training.

6.3 GENERATED DATA

The generator is made to produce 5000 fake data. The first 20 of the generated data is given below:

Table 6.1 Generated Data.

src	Dst
34	115
66	129
212	132
165	97
89	120
69	103
43	55
51	127
41	94
122	104
35	126
68	96
39	110
32	58
84	88
227	126
131	145
282	191
163	115
127	190

Table 6.1 shows the table of Generated data. The Source ID (src) is shown in the first column and the respective Destination ID (Dst) is shown in the second column.

6.4 DISTRIBUTION OF DATA

The real and fake data are plotted in a scatter plot. The plots show how the Generator learns to produce better fake data as the training goes on.

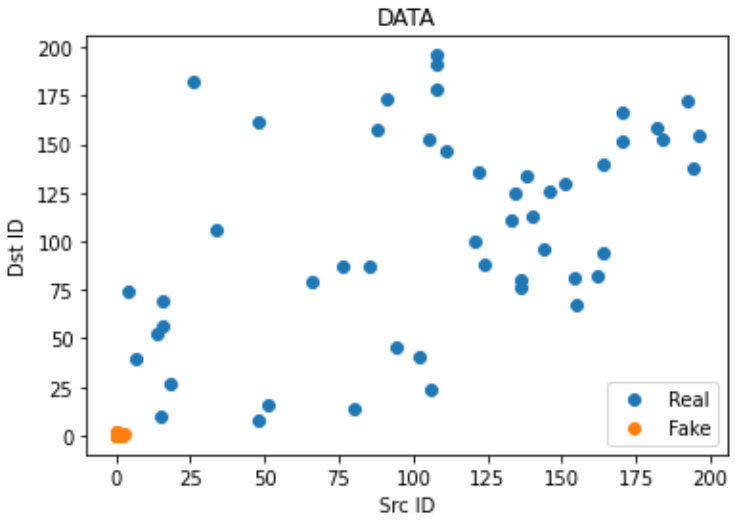

Figure 6.1 Scatter plot - Epoch 0.

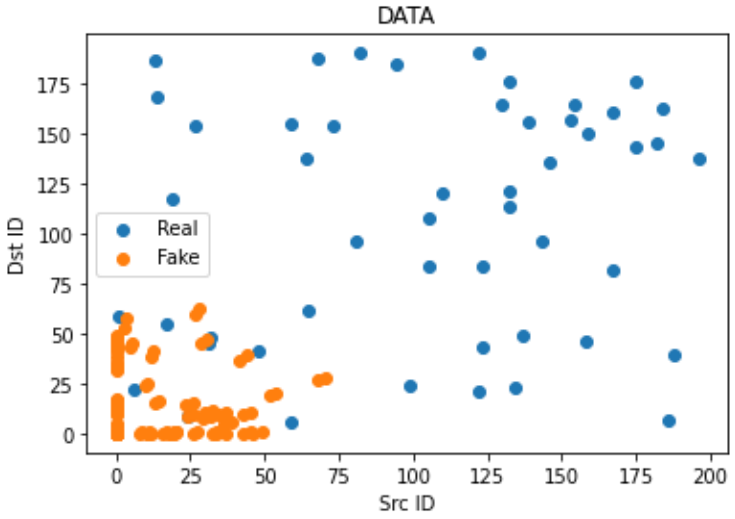

Figure 6.2 Scatter plot - Epoch 10.

Figure 6.1 and Figure 6.2 show how the generator learns to produce better fake data as the number of epochs increases.

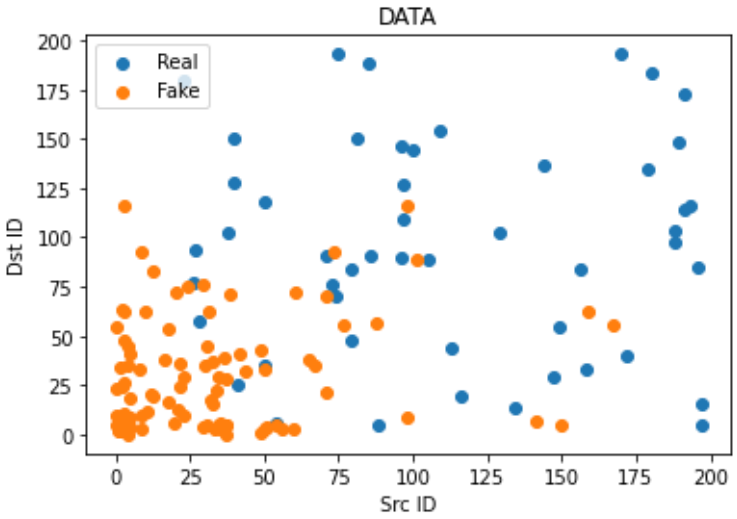

Figure 6.3 Scatter plot - Epoch 20.

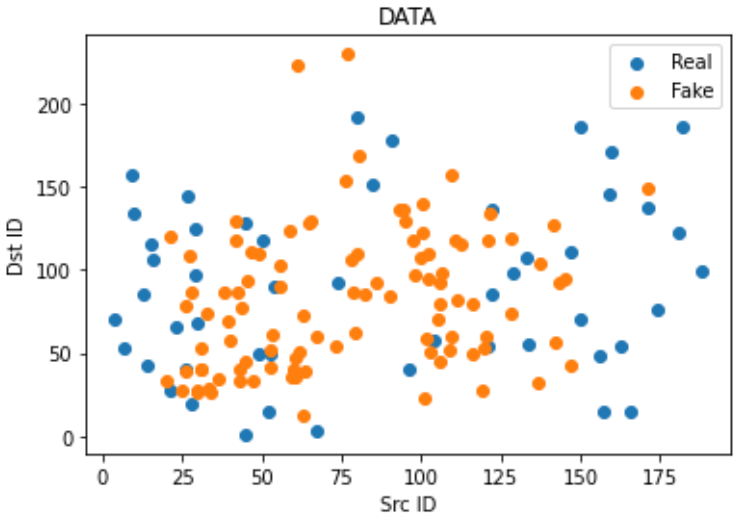

Figure 6.4 Scatter plot - Epoch 35.

Figure 6.3 and Figure 6.4 show that the efficiency of the generator increases drastically from epoch 20 to epoch 35.

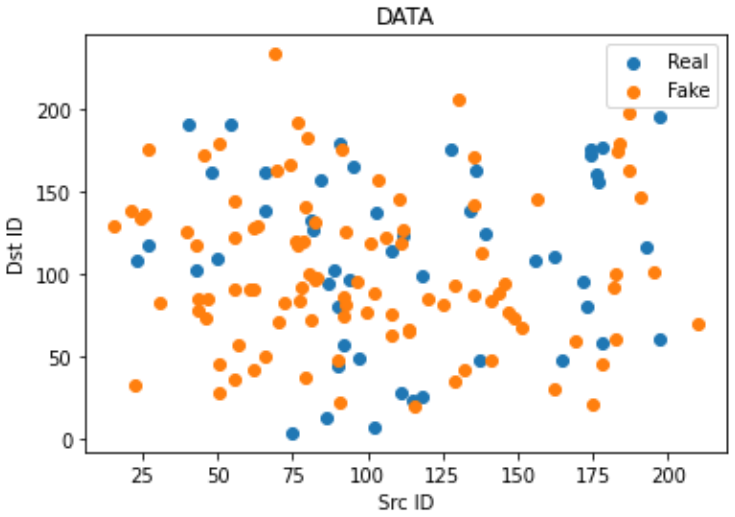

Figure 6.5 Scatter plot - Epoch 77.

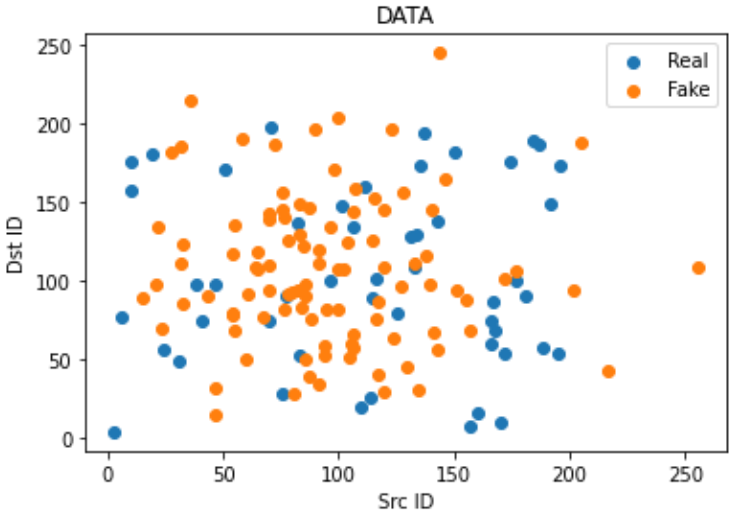

Figure 6.6 Scatter plot - Epoch 150.

Figure 6.5 and Figure 6.6 show that as the number of iterations of training increases, the data starts to become more and more consistent with the real data.

6.5 PROBABILITY DENSITY FUNCTION

Probability Density Function is a metric used here to measure the performance of the Generator. This data show the locations where the traffic is denser.

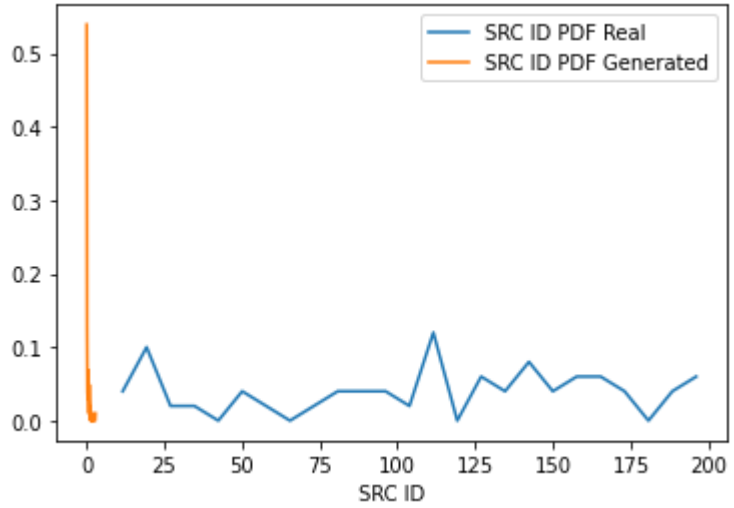

Figure 6.7 Probability Density Function - Epoch 0.

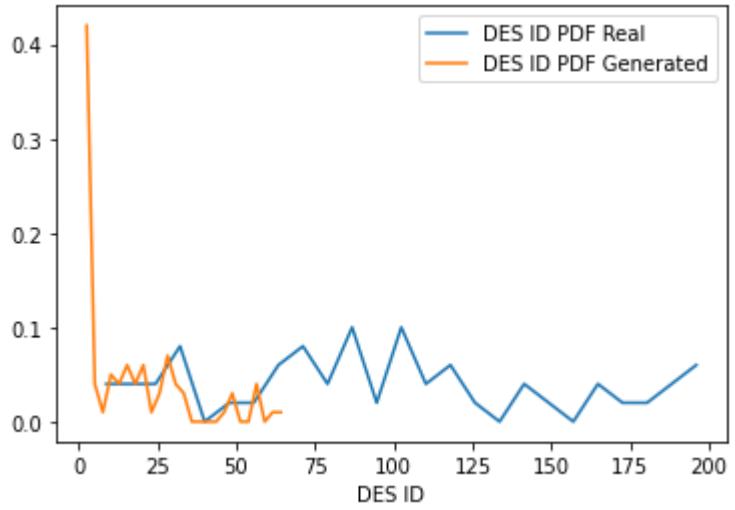

Figure 6.8 Probability Density Function - Epoch 10.

Figure 6.7 and Figure 6.8 show how the generator learns to produce better fake data with increase in training iterations. In the beginning, the generator produces random data.

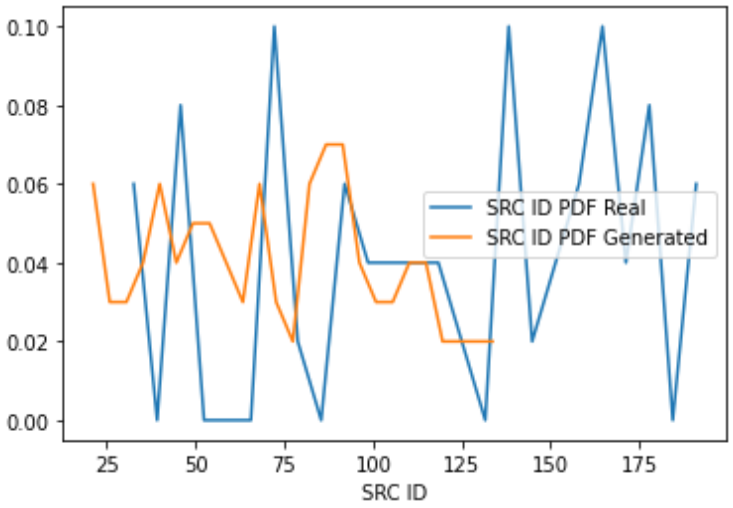

Figure 6.9 Probability Density Function - Epoch 35.

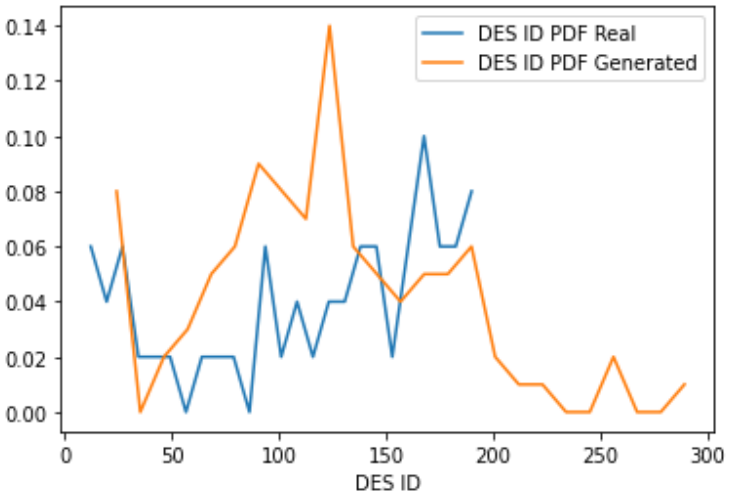

Figure 6.10 Probability Density Function - Epoch 70.

Figure 6.9 and Figure 6.10 show how the generator learns to produce better fake data with better accuracy. The efficiency of the generator increases drastically from epoch 35 to epoch 70.

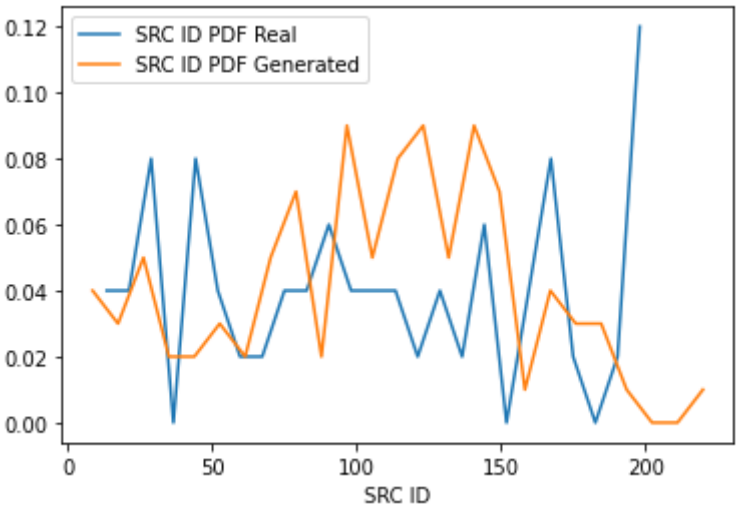

Figure 6.11 Probability Density Function - Epoch 105.

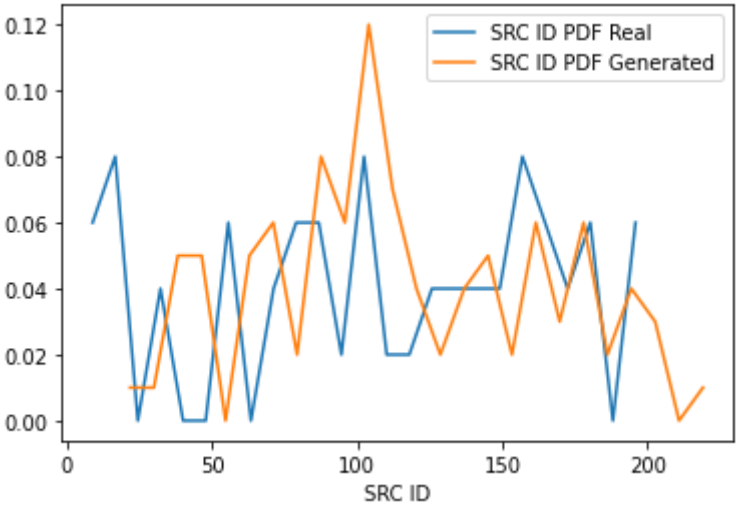

Figure 6.12 Probability Density Function - Epoch 150.

Figure 6.11 and Figure 6.12 show as the number of iterations of training increases, the data starts to become more and more consistent with the real data.

6.6 CUMULATIVE DISTRIBUTION FUNCTION

Cumulative Distribution Function is another metric used here to measure the performance of the generator.

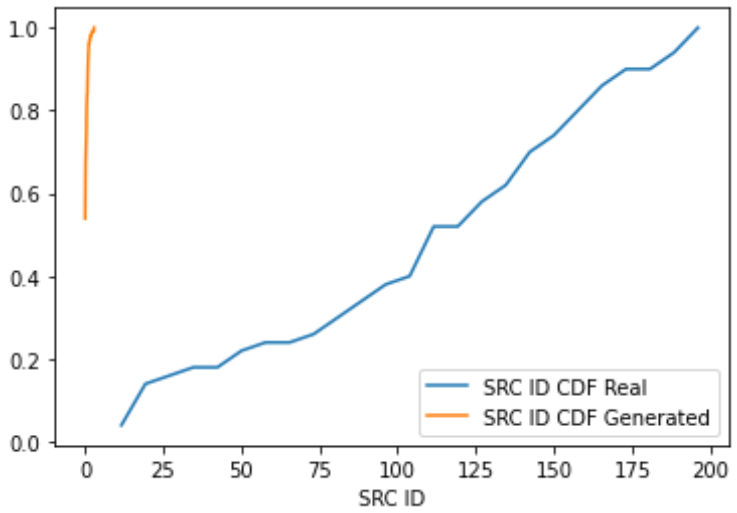

Figure 6.13 Cumulative Distribution Function - Epoch 0.

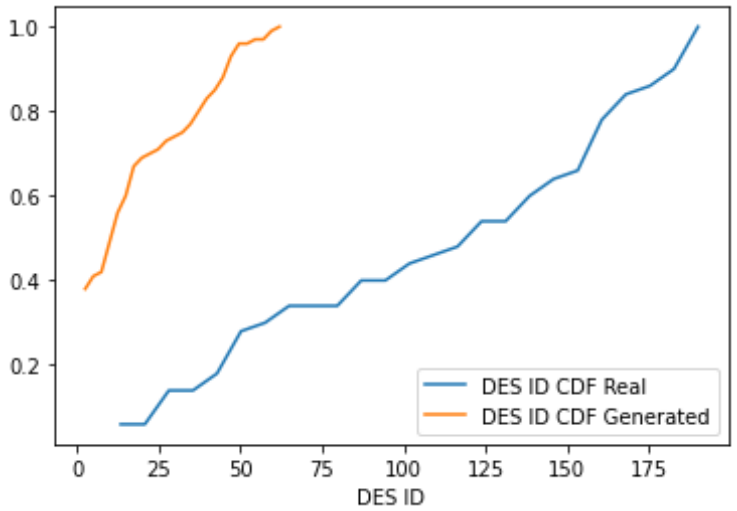

Figure 6.14 Cumulative Distribution Function - Epoch 10.

Figure 6.13 and Figure 6.14 show how the generator learns to produce better fake data with increase in training iterations. In the beginning, the generator produces random data.

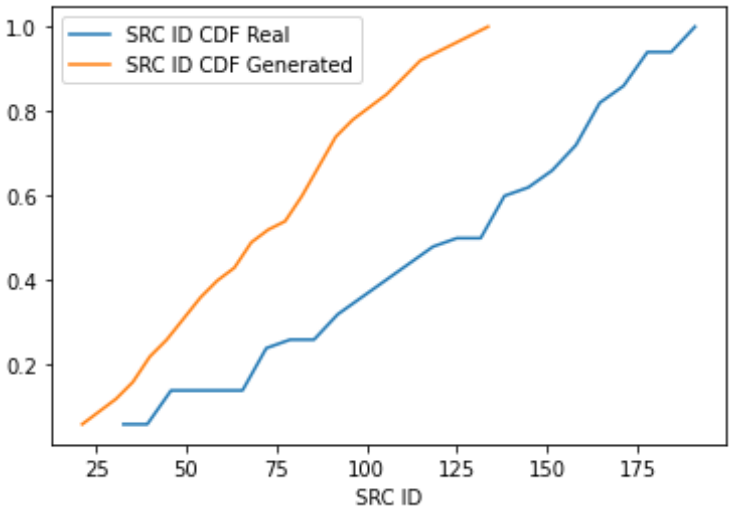

Figure 6.15 Cumulative Distribution Function - Epoch 35.

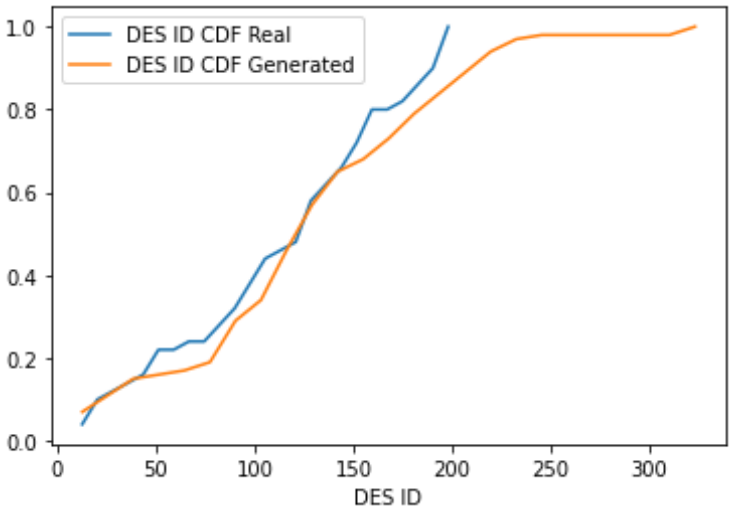

Figure 6.16 Cumulative Distribution Function - Epoch 50.

Figure 6.15 and Figure 6.16 show how the generator learns to produce better fake data with better accuracy as the number of iterations of training increases.

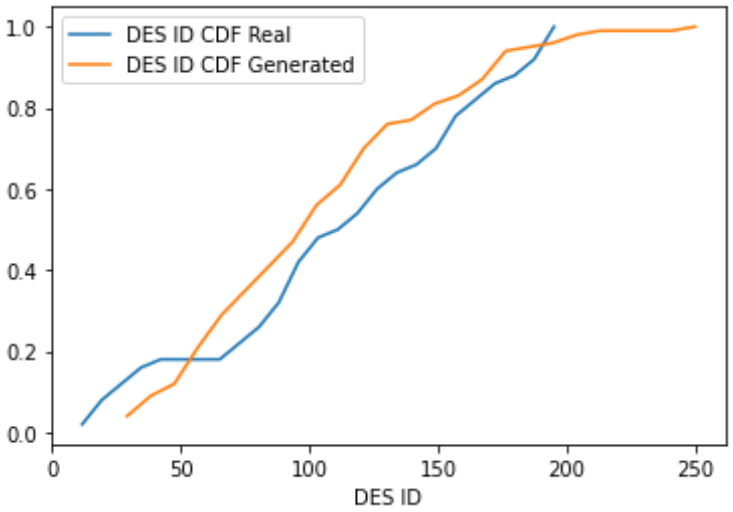

Figure 6.17 Cumulative Distribution Function - Epoch 100.

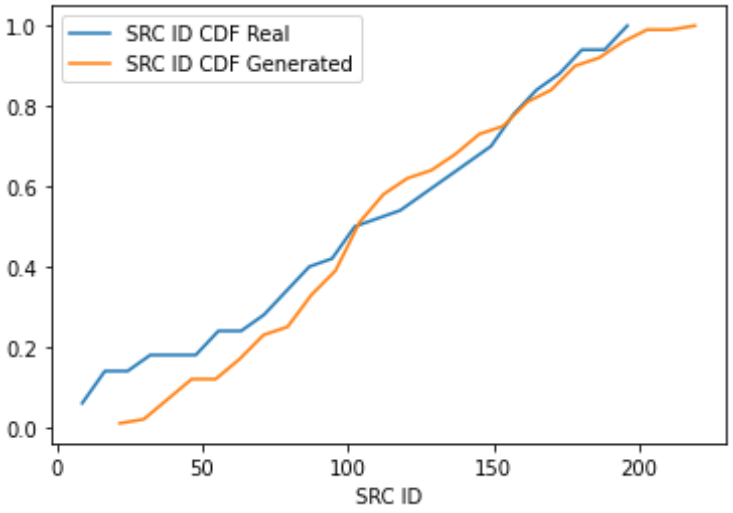

Figure 6.18 Cumulative Distribution Function - Epoch 150.

Figure 6.17 and Figure 6.18 show how the generator learns to produce better fake data with better accuracy as the number of iterations increases. The data starts to become more and more consistent with the real data.

6.7 ACCURACY OF THE DISCRIMINATOR

We can write a routine that will summarize the performance of the discriminator model. It does this by retrieving samples of real traffic data, as well as generating the same number of fake data with the generator model. This pool of data is then fed into the discriminator and its ability to identify the real data is calculated.

Table 6.2 Discriminator Accuracy.

Epoch	**Fake accuracy (%)**	**Real accuracy (%)**
Epoch 1	96	100
Epoch 10	94	96
Epoch 35	58	80
Epoch 50	24	74
Epoch 100	42	58

In Table 6.2, we can see that the discriminator could classify the inputs as real and fake much easily in the beginning. As the generator becomes much more efficient in producing fake data, the discriminator's accuracy drops. The ideal accuracy of the discriminator to identify fake and real data is 50%.

6.8 GENERATOR AND DISCRIMINATOR LOSS

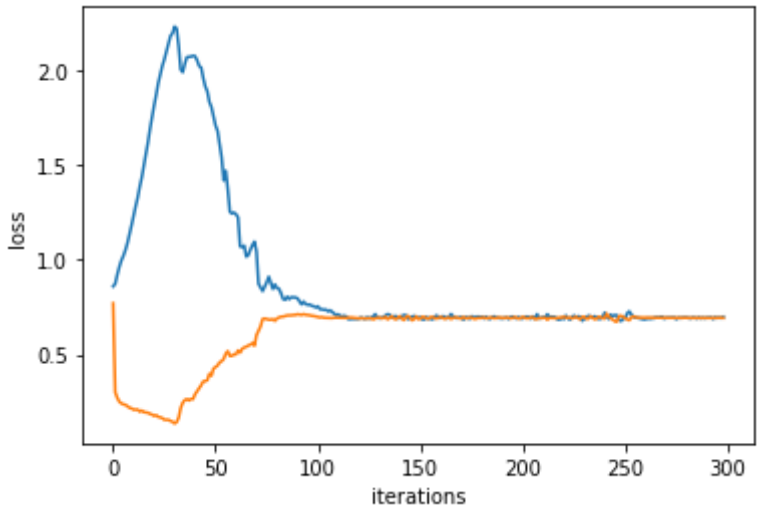

Figure 6.19 Loss vs Iterations.

Figure 6.19 shows the graph of Loss versus Iteration. We can see that the loss of both the generator and the discriminator converges. Initially the loss of discriminator is low. As the generator becomes smarter, the discriminator's loss increases.

6.9 SUMMARY

In this chapter, the generated data is analysed. From the analysis, we could see that the generator slowly learns to produce better fake data. From epochs 50 to 100, the data generated by the generator overshoots the real data. After epoch 100, the generator starts to produce the required output with more accuracy.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

Generative Adversarial Network was implemented using PyTorch to generate realistic mobility traces. The algorithm of GAN was explained and its applications were explored. The data used to train the GAN includes the start and end location of the journey. The generated values almost resemble the real data.

Various loss functions were studied and implemented to reduce the generator and discriminator loss. The performance and accuracy of GAN is evaluated by various evaluation metrics. It can be observed that the accuracy of the discriminator keeps on decreasing as the generator is getting better. This falls in line with the theoretical explanation of Nash's equilibrium. As a result, we obtained a generator which produces data very much similar to the real data which the discriminator fails to identify as fake.

7.2 FUTURE WORK

The future prospects of this project involve the GAN designed as a backbone to implement various other applications.

1. In this work, we have used Vanilla GAN to generate the traffic data. This can be improved by using Conditional GAN. This enables us to generate data based on inputs like day of the week, climate and many more factors.
2. By including various additional parameters while training the GAN, it is possible to gain more accuracy with multiple entities influencing the evaluation of the generated data.

3. The generated data can be simulated in any simulation tools like SUMO. This provides a visual representation of traffic in a region. This allows us to easily identify the region with high traffic density. This identification might help us in deciding factors like where to deploy traffic polices and traffic signals. This can also facilitate uber service providers by identifying which areas need more cars.

REFERENCES

1. Alqahtani, H., Kavakli-Thorne, M. and Kumar, G. (2019) ‘Applications of Generative Adversarial Networks (GANs): An Updated Review’, Archives of Computational Methods in Engineering, pp. 2-29.

2. Karpathy, A., Abbeel, P., Brockman, G., Chen, P., Cheung, V., Duan, R., Goodfellow, I., Kingma, D., Ho, J., Houthooft, R., Salimans, T., Schulman, J., Sutskever, I., and Zaremba, W. (2016) ‘Generative Models’, OpenAI, Stanford, pp. 1-20.

3. Brock, A., Donahue, J. and Simonyan, K. (2019) ‘Large scale GAN training for high fidelity natural image synthesis’, Proc. Int. Conf. Learn. Represent., pp. 1–35.

4. Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012) ‘Theano: new features and speed improvements’, Deep Learning and Unsupervised Feature Learning NIPS Workshop, Cornell University, pp. 2-11.

5. Bengio, Y. (2009) ‘Learning Deep Architectures for AI’, Foundations and Trends in Machine Learning, Vol.2, No.1, pp. 1-127.

6. Celes, C., Boukerche, A. and Loureiro, A. A. F. (2020) ‘From Mobility Traces to Knowledge: Design Guidance for Intelligent Vehicular Networks’, IEEE Network, Vol.34, No.4, pp. 227-233.

7. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014) ‘Generative Adversarial Networks’, Proceedings of the International Conference on Neural Information Processing Systems (NIPS), pp. 2672–2680.

8. Donahue, J., Krähenbühl, P. and Darrell, T. (2017) ‘Adversarial Feature Learning’, Cornell University, pp. 1-18.

9. Murphy, K. P. (2012) ‘Machine Learning: A Probabilistic Perspective’, Cambridge, MA, USA MIT Press, pp. 1-14.

10. Xu, L. and Veeramachaneni, K. (2018) ‘Synthesizing Tabular Data using Generative Adversarial Networks’, Cornell University, pp. 1-12.

11. Lin, Z., Khetan, A., Fanti, G. and Oh, S. (2018) ‘PacGAN: the power of two samples in Generative Adversarial Networks’, NIPS '18 Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 1505–1514.

12. Luc, P., Couprie, C., Chintala, S. and Verbeek, J. (2016) ‘Semantic Segmentation using Adversarial Networks’, NIPS Workshop on Adversarial Training, Dec, Barcelona, Spain, pp. 1-12.

13. Mirza, M. and Osindero, S. (2014) ‘Conditional generative adversarial nets’, Cornell University, pp. 1-7.

14. Isola, P., Zhu, J., Zhou, T. and Efros, A. (2016) ‘Image-to-Image Translation with Conditional Adversarial Networks’, Cornell University, pp. 1-17.

15. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X. (2016) ‘Improved Techniques for Training GANs’, Cornell University, pp. 1-10.

16. Spohn, M. and Trichez, M. (2019) ‘An Analysis of a Real Mobility Trace Based on Standard Mobility Metrics’, Revista de Informática Teórica e Aplicada, Vol.26, pp.1-10.

17. Jin, Y., Zhang, J., Li, M., Tian, Y., Zhu, H. and Fang, Z. (2017) ‘Towards the Automatic Anime Characters Creation with Generative Adversarial Networks’, Cornell University, pp. 1-16.

18. Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F. and Zheng, Y. (2019) ‘Recent Progress on Generative Adversarial Networks (GANs): A Survey’, IEEE Access, Vol.7, pp. 36322-36333.

Fwd: Project Report Approval

Inbox

Prabagarane N Jun 16
to me

---------- Forwarded message ---------
From: **ece.hod ECE** <ece.hod@ssn.edu.in>
Date: Wed, Jun 16, 2021 at 11:56 AM
Subject: Re: Project Report Approval
To: Prabagarane N <prabagaranen@ssn.edu.in>

Approved

Regards,
Radha

On Tue, 15 Jun, 2021, 7:31 PM Prabagarane N,
<prabagaranen@ssn.edu.in> wrote:

> Dear Ma'am
>
> Please find attached to this email, the
> following project report for your perusal and approval
>
> Name of project: GENERATIVE ADVERSARIAL NETWORKS
> FOR THE GENERATION OF REALISTIC MOBILITY TRACES IN
> URBAN SCENARIOS
>
> Name of students
> AKILESH K 312217106014
> GOKULA KRISHNAN S K 312217106044
> HARIHARAN K 312217106051
> --
> warm regards
>
> Prabaagarane. N
> Dept of ECE
> SSN institutions
> Chennai, INDIA
> www.ssn.edu.in