DEGREE PROJECT IN TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Rare Event Learning In URLLC Wireless Networking Environment Using GANs

KTH Thesis Report

Jón Rúnar Baldvinsson

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Authors

Jón Rúnar Baldvinsson (jrba@kth.se)
Machine Learning
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden

Examiner

Amir H. Payberah
Stockholm, Sweden
KTH Royal Institute of Technology

Supervisor

Mårtin Björkman

Stockholm, Sweden

KTH Royal Institute of Technology

Abstract

Industry 4.0 imposes strict requirements on Fifth Generation (5G) system, such as high reliability, availability, and low latency. Guaranteeing such requirements means that there are supposed to be a rare number of system failures. Such rareness can cause problems when access to a broader range of these failures is necessary (e.g., finding optimal scheduler or learning in Deep Reinforcement Learning (DRL)). This work will investigate the possibility of using Generative Adversarial Network (GAN) to generate rare events in wireless communication data that might cause failure events. Conventional training methods fall short when trained on such a dataset, as they will overfit the common values while ignoring the rare values. We propose an alternative training method for GANs, called incremental learning, that aims at increasing learning in the rare sections without sacrificing the learning of the rest of the dataset.

Keywords

URLLC, GANs, Simulation

Acknowledgements

I would like to express my gratitude to Abdulrahman Alabbasi and Milad Ganjalizadeh, my supervisors at Ericsson, for the weekly discussion, and support during this thesis. Thanks to Ericsson for giving me the chance to work on such an interesting project. Additionally, I would like to thank Mårten Björkman and Amir Payberah at KTH for their help during this work.

Acronyms

5G	Fifth Generation
DRL	Deep Reinforcement Learning
GAN	Generative Adversarial Network
URLLC	Ultra-Reliable Low Latency Communication
VAE	Variational Autoencoder
ALOE	At Least One Rare Event
IS	Importance Sampling
MC	Monte Carlo
MIS	Multiple Importance Sampling
SER	Symbol Error Rate
CGAN	Conditional Generative Adversarial Network
3GPP	The 3rd Generation Partnership Project
LST	Learning to Segment the Tail
KL	Kulback-Leibler
JSD	Jensen-Shannon Divergence
DNN	Deep Neural Network
ReLU	Rectified Linear Unit
WGAN	Wasserstein GAN
IL	Incremental Learning
IL-GAN	Incremental Learning GAN
CSI	Channel State Information
CE	Channel Estimate
SINR	Signal to Interference & Noise Ratio
IT	Interarrival Time
PL	Packet Lengths

Contents

Chapter 1

Introduction

The Fourth Industrial Revolution (or Industry 4.0) is the ongoing automation of manufacturing and industrial practices using modern technology such as Cyber-Physical Systems, Internet of Things (IoT), Cloud Computing, and Cognitive computing. 5G is the fifth-generation of technology standard for broadband cellular networks. 5G will be crucial for realizing the full potential of Industry 4.0. applications such as IoT sensing and control, autonomous driving, drones, remote control of autonomous vehicles, and virtual and augmented reality that require high data rates, low latency, and high reliability [1]. The requirements of Industry 4.0 are the primary motivation behind 5G, as 4G can not meet the high demands of Industry 4.0 applications.

Ultra-Reliable Low Latency Communication (URLLC) is a set of features and requirements used to ensure reliability, availability, and low latency in wireless communication. URLLC requires high reliability (e.g., 10 years without failure), high availability (e.g., 99.9999%) and low latency (e.g., <1 ms) [4]. URLLC is an important feature for ensuring that 5G networks are reliable and fast enough for Industry 4.0 applications.

Due to the reliability requirements in URLLC, there is a lack of rare event data. That can cause problems when, for example, training a Deep Reinforcement Learning (DRL) agent to act as a scheduler for URLLC. The limited access to rare event data causes a problem during training, as it will encounter these rare events infrequently. Collecting actual data and simulating it is both time-consuming and expensive. Simulators would have to run for long and consume many computational resources to get sufficient rare events. Collecting actual data is also time-consuming and has privacy concerns.

GAN is a type of generative model first introduced in 2014. GANs utilize an adversarial learning process, using two separate networks, a generator network and a discriminator network. The generator network generates a data sample, and the discriminator network receives the generated sample and a data sample from a real dataset. The discriminator's objective is to detect the generated data, while the generator's objective is to generate data that can trick the discriminator.

1.1 Research Question

This thesis aims to examine the possibility of using GANs to generate URLLC data. We implement an alternative learning method that helps with learning to generate data from all segments of the distribution of wireless communication data, including rare events.

The Research Question of this thesis is *"Can GANs incrementally learn real and simulated wireless communication data distributions without catastrophically forgetting previously learned segments?"*

Benefits, Ethics and Sustainability

The benefit of this project is that using GANs to generate URLLC data can be a solution to the problems described above. Generating data using GANs is faster then simulating data. There are fewer privacy concerns using generated data then using actual data and generative models can be used to generate further rare events that are not present in the dataset.

The work does not raise any ethical concerns.

1.2 Stakeholders

This degree project is a collaboration between KTH the Royal Institute of Technology, and Ericsson Research in Stockholm.

1.3 Outline

This thesis is divided into five chapters. Chapter 2 provides the necessary background in wireless communication, URLLC, GANs and related works. Chapter 3 goes into detail about the engineering methods used in this thesis, and Chapter 4 shows the implementation of them using experiments and presents the thesis results. Chapter 5 concludes the project with an analysis of the results, discussion on future works, and final words.

Chapter 2

Background

In this chapter the background material needed to understand the thesis is presented. First the area of wireless communication is introduced. Next an overview of machine learning and generative models and other related works. Finally, we motivate why GANs were chosen for this particular problem.

2.1 Wireless Communication

The area of wireless communication has evolved significantly over the past decades. It has evolved from analog technology and low data rates to making video calls and watching their tv shows on smartphones. 4G offers everything that most people need, but for Industry 4.0 applications, the needs are much greater.

The 3rd Generation Partnership Project (3GPP) is a combined effort between seven telecommunication standards development organizations. 3GPP defines the URLLC requirements for different applications. This thesis will follow the 3GPP standard in URLLC which includes a set of features used in wireless communication to ensure high reliability, availability, and low latency. Table 2.1.1 shows an example of requirements for a few different use cases. Each use case has different requirements, the number of users (UEs) that can connect to the base station, service availability and reliability requirements, maximum latency, etc.

From Table 2.1.1 we can see the strict requirements URLLC imposes. For example, reliability requirements of 10 years without failure, availability of 99.9999%, and latency of less than 1ms in some cases.

Table 2.1.1: 3GPP defined URLLC Requirements[1]

	Characteristic parameter			Infuence Quantity			
User Case	**Service availability**	**Service reliability**	**End-to-end latency maximum**	**Transfer Interval: target value**	**Survival time**	**# of UEs**	**Service area**
Motion Control	99,9999 % - 99,999999 %	~10 years	<transfer interval value	1 ms	1 ms	50	50 m x 10 m x 10m
Control-to-control in motion control	99,9999 % - 99,999999 %	~10 years	<transfer interval value	10 ms	10 ms	5 - 10	100 m x 30 m x 10 m
Mobile Robots	>99,9999 %	~10 years	<transfer interval value	1 ms - 50 ms	transfer interval value	100	10 km

Due to these strict requirements, URLLC is supposed to have a low number of system failures. There are many causes for possible failure events. Channel fading, delay, and load are factors that might lead to errors. Channel fading and load would lead to variable interference, and consecutive interference and delay violation lead to a failure event.

Wireless communication has many parameters involved in linking the client with the server.

Channel State Information (CSI) is information about known channel properties in the communication link between a server and a client. Channel Estimate (CE) uses the CSI to estimate how a signal will transmit between a client and server. Channel fading is included in the CSI.

SINR is used to quantify the rate of information that can reliably be transferred in wireless communication systems.

Delay is the difference between the time of arrival and time of sending.

IT is the time between the arrival of a signal and the arrival of the next signal.

PL is the number of bytes in a network packet.

2.2 Machine Learning

2.2.1 Neural Networks

Neural Networks were inspired by the nervous system in the human body. Data is used as an input and is passed through the neural network, and the network gives an output. A neural network commonly consists of an input layer, a number of hidden layers, and an output layer. At each layer, there are neurons. Between layers, there are weights and biases used to transform the input to the desired output.

Additionally, there is an activation function used on each node in a layer. Many different activation functions exist, but the most commonly used ones are the Sigmoid function, Rectified Linear Unit (ReLU) and Tanh. The activation function is used to introduce non-linearity into the neural network. Figure 2.2.1 shows a general architecture of a feed-forward neural network, and Equation 2.1 shows how the output is calculated in that neural network.

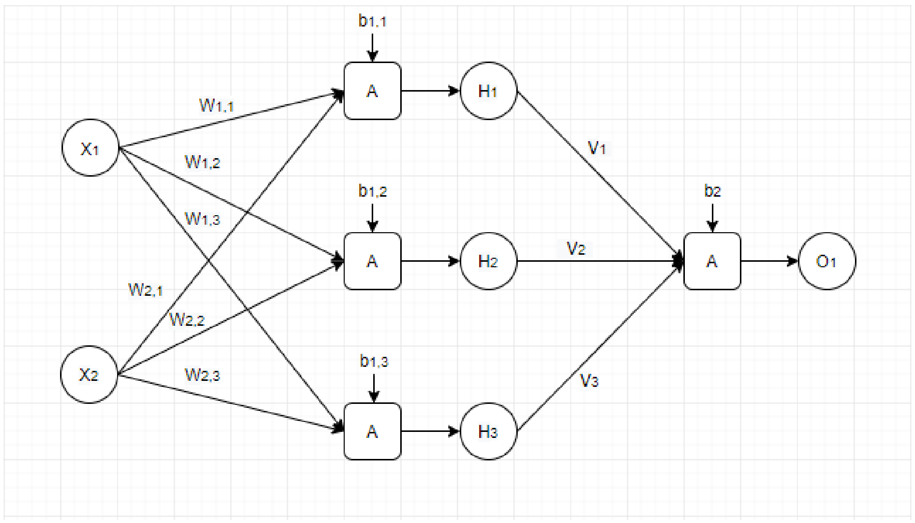


Figure 2.2.1: Neural Network

$$
\begin{aligned}
H &= A(X * W + b_1) \\
O &= A(H * V + b_2) \qquad (2.1)
\end{aligned}
$$

A loss function is used to calculate the difference between the output of a neural network and the desired output. To train a neural network, the gradient of the loss function is calculated, with regard to the weights and biases. Then, using Stochastic gradient descent, the weights and biases are modified. This method of training is called back-propagation.

Deep learning is a subset of Machine Learning where Deep Neural Network (DNN) are utilized. A DNN consists of many hidden layers. DNNs are an old concept but only became viable in recent years due to advances in processing units like CPUs and GPUs and the increased availability of enough data to train them.

Two subsections of machine learning are supervised learning and unsupervised learning. Supervised learning uses labeled data, while unsupervised learning uses unlabeled data.

Two common uses of machine learning are generative models and discriminative models. Discriminative models aim to maximize the probability of the output given the input ($P(y|x)$). Generative models aim to learn the distribution of a dataset and produce further examples from the same distribution ($P(x)$ or $P(x|y)$). A well-known dataset is the MNIST dataset consisting of images of handwritten digits from 0-9. In discriminative learning, the input (x) is the image's pixel values, and the output (y) is a class label. The discriminative model aims to predict which digit the image contains. In Generative learning, the input (x) is the image's pixel values and possibly a class label (y), and the output is a new data sample ($x^{'}$). Discriminative learning is a supervised learning method, while Generative learning can be either supervised or unsupervised.

2.2.2 Generative Models

While discriminative learning was the main reason for the increased popularity of deep learning in the last decade, interest in generative learning has increased steadily. Many different generative models exist, but this discussion will focus on the most common ones. Those are Variational Autoencoder (VAE) and GANs

VAE [12] is a type of generative models. VAEs are split into three parts, an Encoder, a Decoder, and a latent space. The Encoder encodes the input data as a distribution over the latent space. That distribution is sampled, and the Decoder decodes the sampled point. Instead of mapping the input to a single point, VAEs encode the input as a distribution over the latent space. To ensure that points close to each other in the latent space have similar output and that each point in the latent space has a meaningful output, the latent space is regularized. The latent space is regularized using Kulback-Leibler (KL) divergence. KL divergence is used to minimize the distance between the latent space and a target distribution. This method is called variational inference. Figure 2.2.2 shows the architecture of a variational auto-encoder.

To generate new data, z is sampled from the latent distribution and used as an input to the decoding network. The output should be a data point that follows the same distribution as the training data.

Generative Adversarial Network were first introduced in 2014 [6]. GANs utilize adversarial learning. A discriminative model estimates the probability that a data sample came from the real dataset or was generated by a generator. The generator

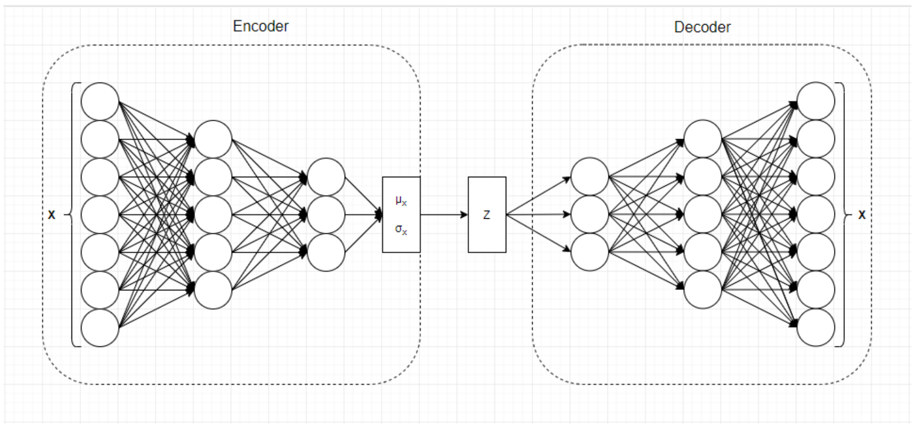


Figure 2.2.2: Variational Auto-encoder

tries to generate samples that lead to the discriminative model making an incorrect guess. Figure 2.2.3 shows the conventional architecture of GANs. The generator is fed input noise or a latent space, like a multivariate distribution, and it generates a data sample. The discriminator receives both the generated data and real data as input, and it predicts whether each data sample came from the real dataset or from the generator.

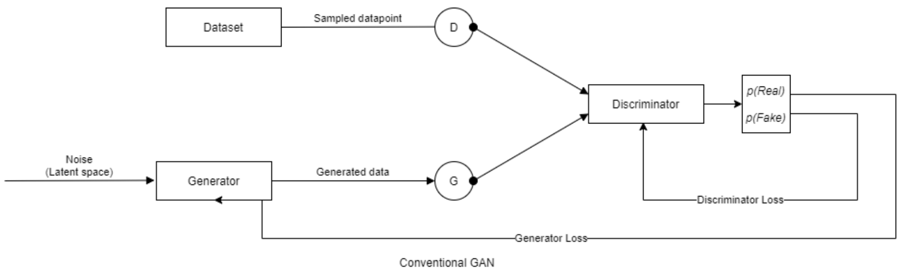


Figure 2.2.3: GAN

Conditional Generative Adversarial Network (CGAN) [13] is a modification of the original GAN architecture. A class label y is used as an additional input to the generator and the discriminator. This modification allows for control over the generated data. Figure 2.2.4 shows the architecture of a Conditional GAN.

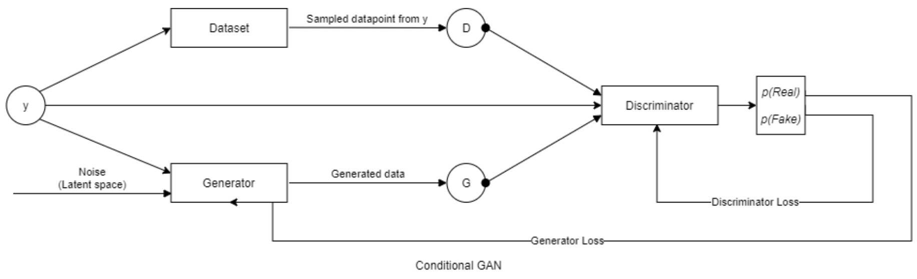


Figure 2.2.4: Conditional GAN

The original loss function used in [6] was

$$\text{min}_G \text{max}_D V(D, G) = E_{\mathbf{x} \sim p_{data(\mathbf{x})}}[\log D(\mathbf{x})] + E_{\mathbf{z} \sim p_{\mathbf{z}(\mathbf{z})}}[\log(1 - D(G(\mathbf{z}))] \tag{2.2}$$

where the discriminator D is trained to maximize the probability of predicting the correct label while the generator G is trained to minimize it. This is the equivalent of a zero-sum game between two players.

2.3 Related Works

Training GANs is unstable, and it has been shown that the KL-divergence and JS-divergence used in minimizing GANs is a possible source of instability and vanishing gradients in GAN training [2] [3]. Wasserstein GAN (WGAN) [3] propose using the Earth-Mover distance, also known as the Wasserstein-1 distance, instead.

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} \left[||x - y|| \right] \tag{2.3}$$

And using the Kantorovich-Rubinstein duality to remove the infimum, the loss function becomes

$$\underset{w \in W}{max} \quad \mathbb{E}_{x \sim \mathbb{P}_r} \left[f_w(x) \right] - \mathbb{E}_{z \sim p(z)} \left[f_w(G_\theta(z)) \right] \tag{2.4}$$

where f_w is the set of 1-Lipschitz functions. To enforce the Lipschitz constraint on the Discriminator,a gradient penalty is used on the parameters of the Discriminator [7].

Instead of the Discriminator predicting a probability of a sample being real, the WGAN discriminator output values are not bounded to be between 0 and 1. Instead, the Discriminator attempts to assign the low values to generated samples and high values to real samples. The loss function for the generator becomes

$$\underset{\theta \in \Theta}{max} \quad \mathbb{E}_{z \sim p(z)} \left[f_w(G_\theta(z)) \right] \tag{2.5}$$

So the generator tries to get the Discriminator to give the generated data high values. WGANs have been shown to increase stability in training GANs as it has a more reliable gradient in all areas.

Other methods have been utilized to speed up simulators in rare-event environments. Importance Sampling (IS) is an alternative to Monte Carlo (MC) sampling. It is used when MC sampling is infeasible or inefficient, e.g., in rare-event environments. IS is different from MC in that it modifies the sampling procedure to increase the probability of sampling from a specific area of the distribution. So while MC samples from the Gaussian distribution of the observed variables, $\pi(\mathbf{x})$, IS samples from a proposal distribution $q(\mathbf{x})$. Let us call that area D. D can, for example, be the area that the rare events are in. To provide an unbiased result, the samples from D are weighed proportional to the increase in sample rate.

Multiple Importance Sampling (MIS) is an extension of IS. In MIS multiple proposal distributions are used.

The authors of [5] analyzed the effects of estimating a MIS technique called At Least One Rare Event (ALOE), first introduced in [14], does in evaluating the Symbol Error Rate (SER) in the presence of Gaussian Noise. The proposal distributions in ALOE sample the system conditional on an error taking place, providing an unbiased estimation of the SER. In this environment, where the signal-to-noise ratio is high (symbol errors happen rarely), ALOE provides a speedup of orders of magnitude over MC methods.

It has been shown that IS and ALOE offer speedup of orders of magnitudes over MC methods in different rare-event environment simulations [5, 17]. The speedup is only on the simulation side, so while IS provides a faster simulation time, it also requires problem-specific analysis to design a suitable proposal distribution before the simulation is run.

Many environments have the characteristic of having a few head classes occupying most of the instances and a long-tail distribution having very few instances. In this thesis, we call those environments rare-event environments. Training robust systems in these rare-event environments is challenging. The danger is that the focus goes towards the head classes while the tail classes are ignored. Methods such as undersampling the head classes or oversampling the tail classes have the risk of either underfitting the head classes or over-fitting the tail classes.

Learning to Segment the Tail (LST) [8] is a method to increase learning in rare event environments. The first step in LST is to split the unbalanced dataset into T balanced datasets ($C_{0,\ldots,T-1}$). The most frequently occurring head classes are in dataset C_0, and the least frequently occurring tail classes are in dataset C_{T-1}. Each dataset is trained in a class-incremental learning style. Class-incremental learning aims to create a model that can classify both the new and the old classes. The architecture for LST is shown in Figure 2.3.1. A base model θ_0 is trained on C_0, then sequentially $C_{1,\ldots,T-1}$ is used to train a new model $\theta_{1,\ldots,T-1}$. During each training phase, data from the previously seen classes are also used. This is necessary to prevent catastrophic forgetting of the previously seen data. The data from the previously seen classes have a significantly higher number of occurrences, so it needs to be sampled to enable the model to learn the new classes. Additionally, a meta-learning-based weight generator is used to increase learning in rare-event cases. The meta-learning-based weight generator transfers knowledge from the previously seen classes to the rarely seen classes. The paper shows 8% AP improvements in the tail classes over baseline models, while overall, the improvement is 2.2%.

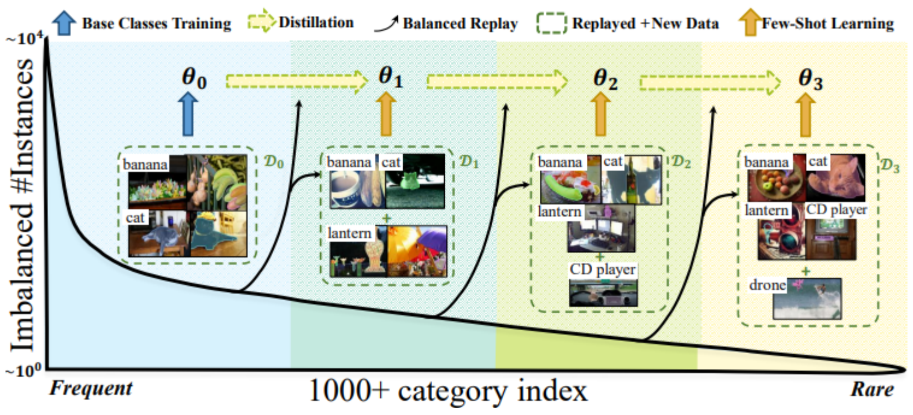


Figure 2.3.1: LST architecture

Using GANs to augment the training of a DRL trained as a URLLC scheduler has been done before [11]. The objective was to give the agent more experience before deployment. The input to the GAN was synthetic data of traffic arrival and channel information. They conditioned the GAN so that the output of the generator network would be similar to the input. Using this GAN, they created a comprehensive dataset that was used to pre-train the agent before deployment. The paper shows that the agent trained on the generated data recovered faster in unexpected extreme events that can cause failures. The environment used in that work is not comparable to the one in our work.

2.4 Summary

GANs were chosen for this project due to the simplicity of the adversarial learning process. GANs make no assumptions about the data, model, or latent variables, unlike VAEs. CGANs also allow for a conditional generation, which is what we will be focusing on in this thesis.

Chapter 3

Methods

3.1 Data

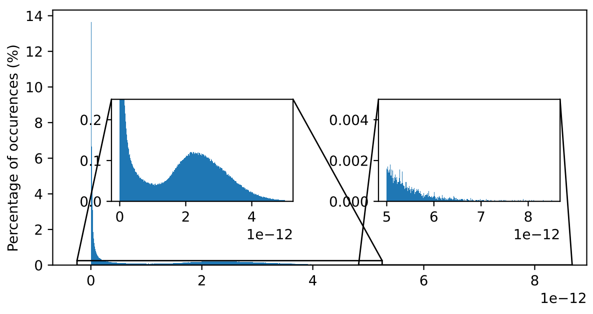
This section will describe the data used to train the models in question. We found no real dataset using URLLC data that includes rare events that we can train on, so a different approach was taken. The dataset used in this thesis is a mix of real data and simulated data. We hypothesize that training a Generative model to correctly and faithfully generate data from this dataset is a step in the right direction to solving the problem using real URLLC data and that using the methods described in this paper will translate to solving that problem. Additionally, showing that the model can generate data in the tail distribution sections with no data shows that the model can generate rare events not present in the dataset.

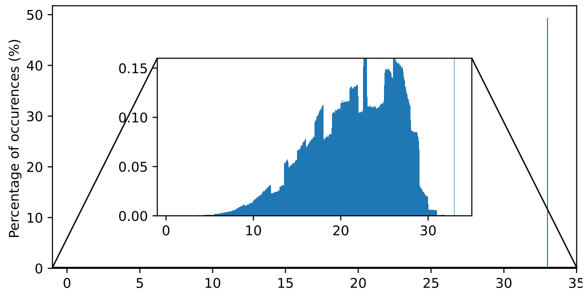
3.1.1 Data collection and processing

Two datasets were gathered from simulated data. The first dataset consists of Channel Estimate (CE) data and has 10.000.000 data points. 3.1.1a shows the distribution of the CE data along with a zoom-in of the less common section.

The second dataset consists of Signal to Interference & Noise Ratio (SINR) and delay data. That dataset has 15.000.000 data points. Figure 3.1.1b shows a plot of the distribution of the SINR data along with a zoom-in on the less common sections, while 3.1.1c shows the distribution of the Delay data.

The real dataset was gathered from Crawdad [16] and consists of wireless networking


(a) CE


(b) SINR

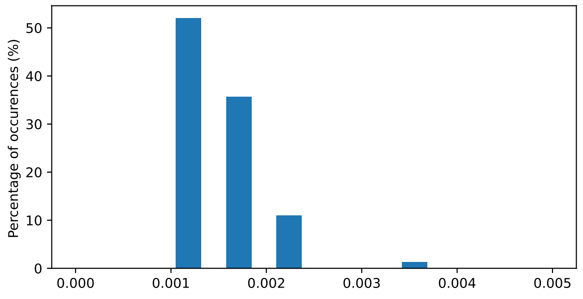
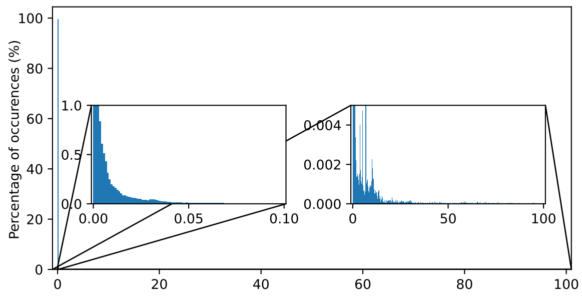

(c) Delay

Figure 3.1.1: Simulated Dataset


(a) IT

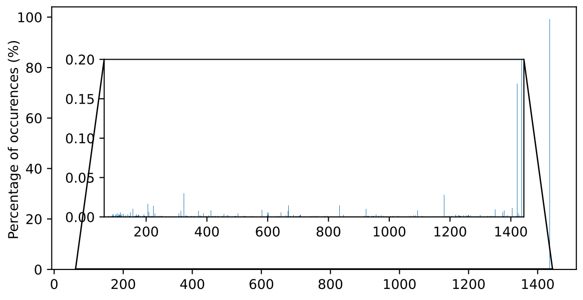

(b) PL

Figure 3.1.2: Crawdad Dataset

data gathered from various video streaming platforms, like YouTube, Skype, Google Hangouts, etc. From that data, the Interarrival Time (IT) and Packet Lengths (PL) was extracted and calculated. The dataset has 2.494.707 data points. Figure 3.1.2a shows a plot of the IT data along with a zoom of the less common sections. Figure 3.1.2b shows a plot of the PL data along with a zoom of the less common sections.

3.1.2 Data segmentation

In this work, we will be segmenting our datasets and using it to train a CGAN conditioned on data segments. Figure 3.2.1 shows how the data can be segmented. Then a CGAN can be trained, conditioning on the segment. CE, SINR and IT are continuous variables while Delay and PL are discrete variables. The continuous variables were normalized to be between 0 and 1, and the discrete variables were categorized. For the CE and SINR datasets, the whole dataset was normalized to be between 0 and 1. For the IT the whole dataset is from values 0 - 100, and the majority of the samples are below 0.002, so normalizing it caused problems when training the model. So for the IT, we normalized each segment of the data individually. Delay has four possible values, so we categorized it into four categories, and PL has 1368 possible values, so we categorize it into 1368 categories.

Table 3.1.1 shows the segmentation and distribution of the Channel Estimate dataset. From the table, we can see a clear head distribution. The first segment has more than half of the data points in the dataset, while the first three segments have over 99% of the data. The last two segments can be considered rare events.

Table 3.1.1: Segmentation and distribution of the Channel Estimate dataset

Segment	**Segmentation**	**Distribution**
Segment 1	0 - 0.125	5374842 (54%)
Segment 2	0.075 - 0.4	2307424 (23%)
Segment 3	0.3 -0.55	2279126 (22%)
Segment 4	0.45 - 0.75	38449 (0.3%)
Segment 5	0.7 - 1	159 (0.00016%)

Figure 3.1.3 shows a plot of the segmentation of the normalized CE dataset.

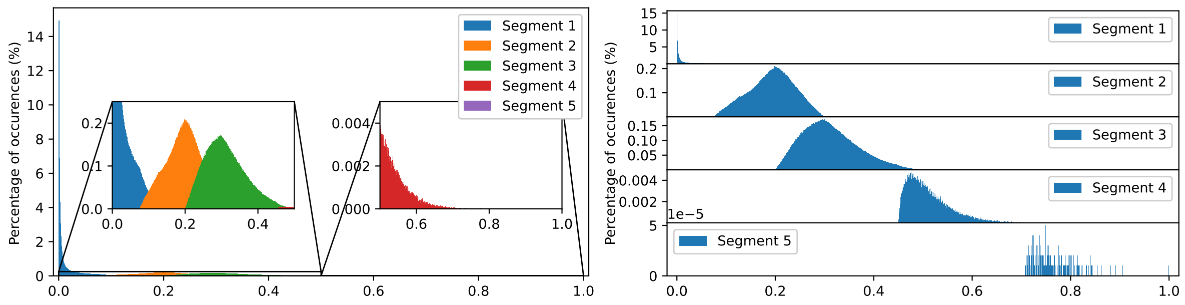


Figure 3.1.3: CE Data

Table 3.1.2 shows the segmentation and distribution of the SINR and delay dataset. The table shows that both features have a clear head distribution, with the first segment having around 50% of the data and the first three segments around 99% of the data. There is a big difference in the number of data points in the fourth segment. SINR has only 1712 data points which constitute around 0.001% of the dataset, while the delay data has 199.996 data points in the fourth segment or around 1.3%.

Table 3.1.2: Segmentation and distribution of the SINR & Delay dataset

Segment	**SINR**		**Delay**	
	Segmentation	**Distribution**	**Segmentation**	**Distribution**
Segment 1	1	7394029 (49%)	0	7799941 (52%)
Segment 2	0.4 - 0.9999	5769242 (38%)	1	5349919 (36%)
Segment 3	0.1 - 0.65	1834850 (12%)	2	1649977 (11%)
Segment 4	0 - 0.15	1712 (0.001%)	3	199996 (1.3%)

Figure 3.1.4 shows a plot of the segmentation of the normalized SINR & delay dataset.

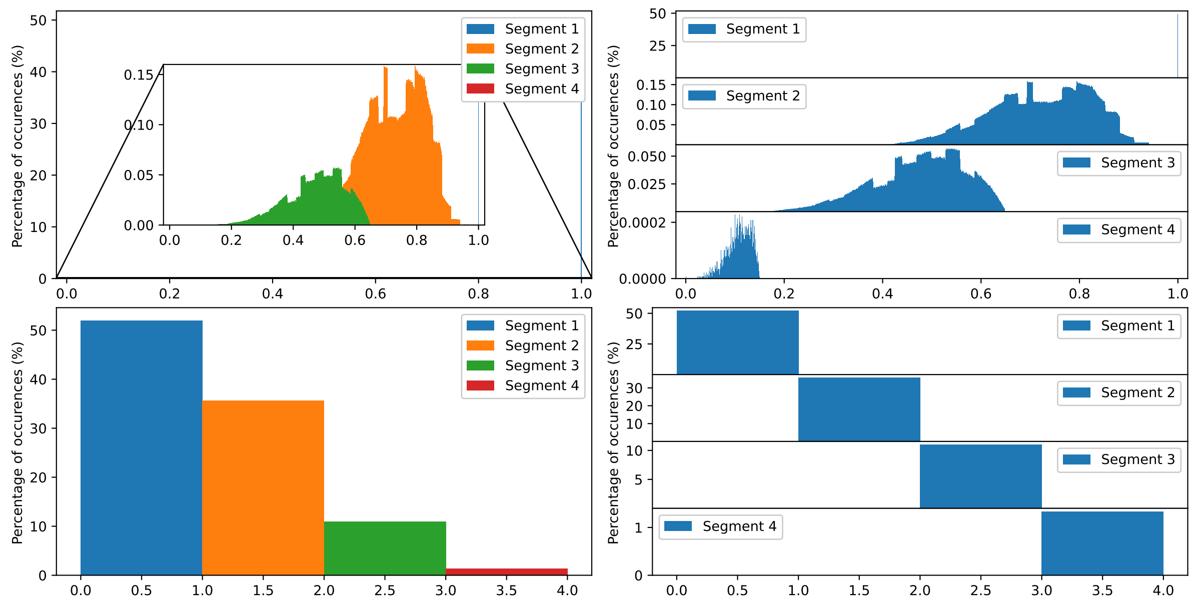


Figure 3.1.4: SINR and Delay Data

Table 3.1.3 shows the segmentation and distribution of the IT and PL dataset. This dataset is the most unbalanced. The first segment of the IT data has around 91% of the data and the first two have around 99% of the data. The first segment of the PL data has around 99% of the data.

Table 3.1.3: Segmentation and distribution of the IT & PL dataset

Segment	**Interarrival**		**Packet**	
	Segmentation	**Distribution**	**Segmentation**	**Distribution**
Segment 1	0 - 0.002	2265737 (91%)	1367	2472335 99%
Segment 2	0.001 - 0.2	190501 (8%)	900 – 1366	9254 0.4%
Segment 3	0.01 - 2	34429 (1%)	600 – 899	2632 0.1%
Segment 4	1 - 20	3311 (0.1%)	300 - 599	3977 0.16%
Segment 5	10 - 100	729 (0.03%)	0 - 299	6509 0.2%

Figure 3.1.5 shows a plot of the segmented IT & PL dataset.

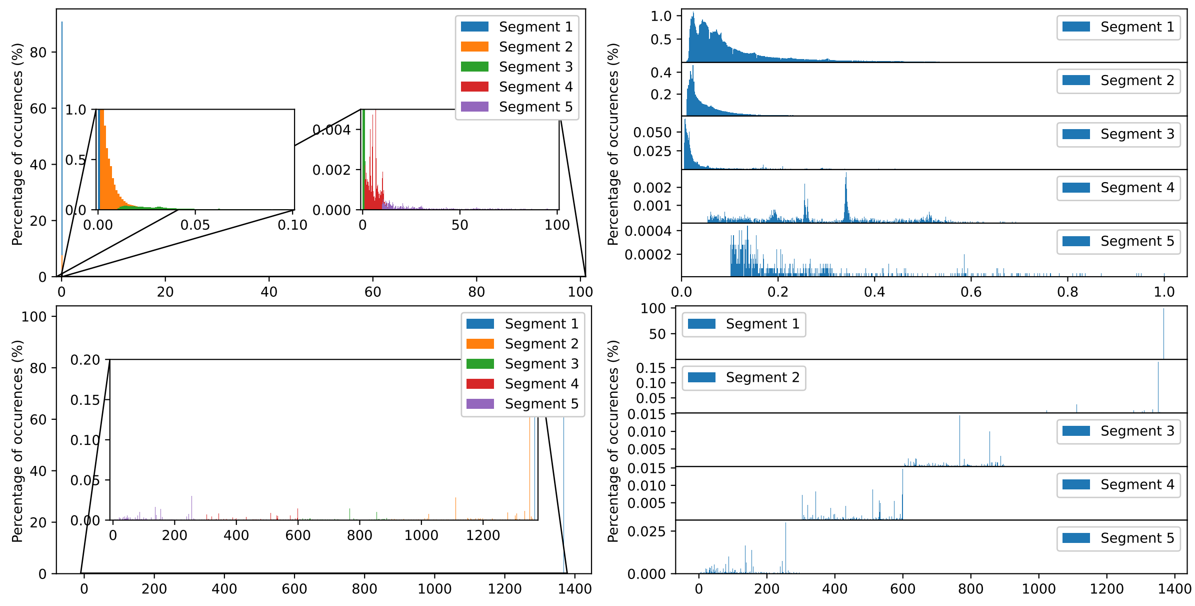


Figure 3.1.5: IT & PL Data

When jointly generating more than one feature, such as generating SINR & Delay or IT & PL, it is not possible to split the data into training segments by the segments of the features. Since a data point can have SINR from segment one and Delay from segment 3, the data needs to be segmented into training segments. Many possible methods, such as using the maximum of the feature segments, their sum, or joining them into both segments. Table 3.1.4 shows the different methods

3.2 Training methods - Incremental Learning

Many different training methods are possible when training generative models. Each method has its pros and cons, and they will be analyzed in this work.

Table 3.1.4: Segmentation

Method	SINR segment	Delay segment	Training segment
Max	1	3	3
Sum	1	3	4
Join	1	3	1 & 3

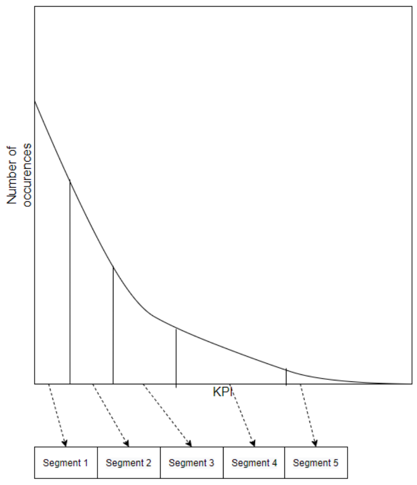


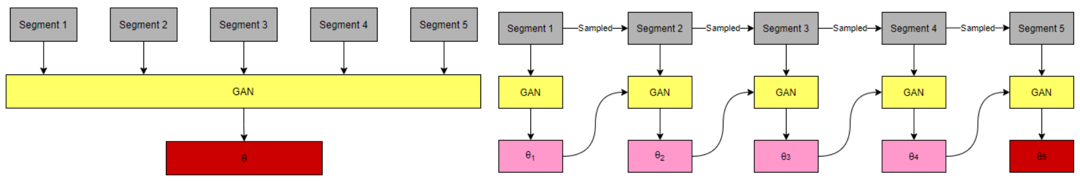
Figure 3.2.1: Data segmentation

Training on data dominated by a large head distribution and a long tail distribution is problematic. A simple GAN trained on the whole dataset will most likely fit well to the head distribution while ignoring parts of the distribution.

Another method is the one used in [8], that was discussed in Section 2.3. The model is trained incrementally on the data, beginning with the head segment and ending with the tail segment. During training on each segment, the model learns the data of the current segment while remembering the data of the previous segments. This method is called Incremental Learning in this work. Figure 3.2.2 show the different training methods.

The method used in [8] is used to train a discriminative model. The main work of this thesis is to modify the method in that work to work for generative models. As far as the author knows, this has not previously been done.

The main idea behind incremental learning is learning new segments without forgetting the old segments. The first step is to divide the data into segments as described in the previous section. Then a model is trained on the data in the first



(a) Single GAN (b) Incremental learning

Figure 3.2.2: Different training methods

segment, the one which includes the head distribution. The model is trained until some convergence criteria is met or for a fixed number of epochs. After segment one training is complete, the model is trained using the data from segment two. This continues until all segments are done and then we have a final model. Figure 3.2.2b shows how incremental learning works. During incremental learning, the focus is on learning new segments without forgetting the previous segments. To combat catastrophic forgetting, we propose two methods, first described in [8] and modified to work on GANs. Those methods are sampling from the previous segments and having training data from all previous segments in the current training segment, and freezing the weights of the model after finishing training on a segment.

The first method is Balanced Replay, where we sample from the old segments, so there is a balance between previously learned segments and the current segment. Data is sampled from the previous segments so that there is a representation of all segments seen so far, along with the current segment data. Balanced Replay allows the model to modify the weights to remember the old segments and learn the new segments. Since there is a balance between the segments during training, the model can learn to generate data from the rare-event segments. Also, since there is still a representation of the old segments, there should be no catastrophic forgetting. Figure 3.2.3 shows how an example of the balance in the training data at each segment. In that Figure, an equal amount of data is sampled from previous segments, as there is in the current segment.

The method of freezing nodes in a neural network are a known way to keep a feature representation of a batch of data. They have been used in other methods, such as class incremental learning [15]. Freezing causes parts of the model to never be updated again during backpropagation. The motivation behind this is that some of the representation learned in each segment is kept, helping the model remember old classes despite less representation during training of new segments. Many different methods for freezing

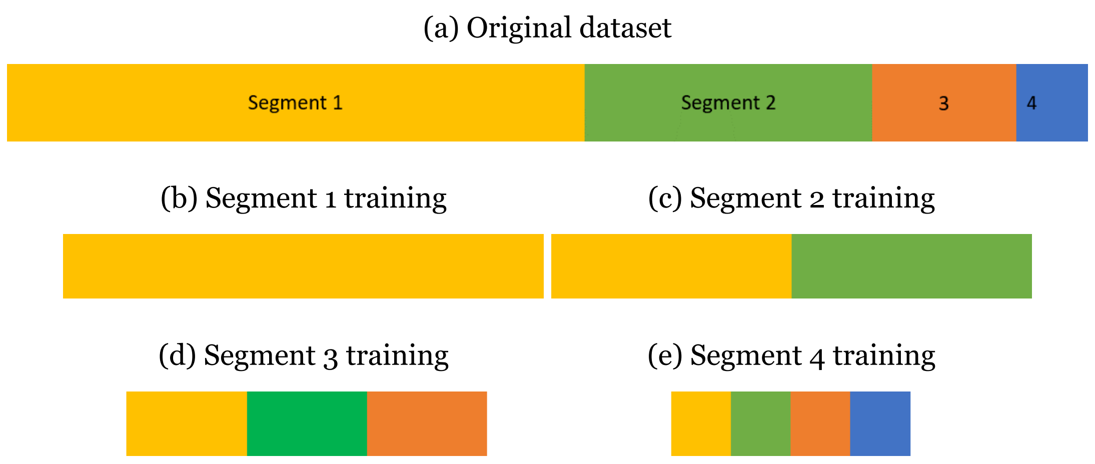


Figure 3.2.3: Balanced Replay

exists. One method is to freeze and grow an network. In that scenario, after training on a batch of data, all the weights are frozen and the network is grown. That saves the representation learned in the original network, and creates new layers and nodes to learn new representations.

Many other possible methods for freezing are possible, such as layer freezing, vertical freezing or gradually changing vertical freezing [9]. In layer freezing, a number of layers are frozen completely. In vertical freezing a fixed portion of many layers are frozen. Gradually changing vertical freezing can either start with most of the layer being frozen and then the number of frozen nodes gradually decrease, or start with a small portion of the layer being frozen and gradually increase the number of frozen nodes. Figure 3.2.4 shows how the different freezing methods might look.

Another method is incrementally freezing layers vertically. Initially all weights are unfrozen, but after training on the first segment, some of the weights and biases between the input layer and the first layer are frozen. After training on the second segment, some of the weights between the first layer and second layer are frozen. This continues until training of all segments is done. The motivation behind incrementally freezing is freezing a representation of each segment in the model. Since the focus of the thesis is on equally generating data from each segment, this is preferable to other freezing methods. Figure 3.2.5 shows the freezing of the model parameters during training of different segments.

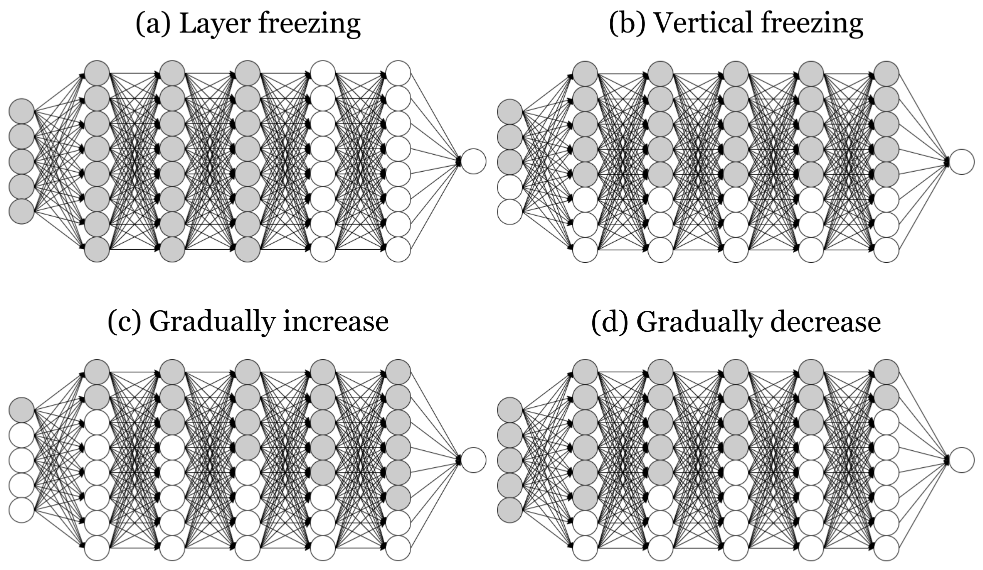


Figure 3.2.4: Different freezing methods. Weights and biases between grey nodes are frozen

3.3 Models

The previous section outlines the incremental learning training method for GANs. We will call this model Incremental Learning GAN (IL-GAN) in this work. The IL-GAN will be trained as a CGAN conditioned on training segment. The model will be tested on a few experiments and it will be compared to a baseline model. The baseline model will also be a CGAN conditioned on training segments.

Both the IL-GAN and the baseline model will have five hidden layers with 200 nodes in each layer. The latent space has a dimension of 100 and it is sampled from normal distribution with $\mu = 0$ and $\sigma = 1$. Dropout is used for regularization. Each model will be trained as a Wasserstein GAN using a gradient penalty of 10 like recommended in [7]. The Adam optimizer with a learning rate of 0.0002 is used.

Since wireless communication data can both be categorical and continuous, the generator needs to be able to generate both types of data. To generate continuous variables the Sigmoid activation function is used in the output nodes of the generator. The data has been normalized to be between zero and one, so the Sigmoid activation function is well suitable to generate data in that range. To generate categorical variables, the Gumbel-softmax[10] is used. The data has been one-hot encoded so the Gumbel-softmax is used to generate one-hot encoded data

The IL-GAN uses both Balanced Replay and layer freezing. The Balanced Replay

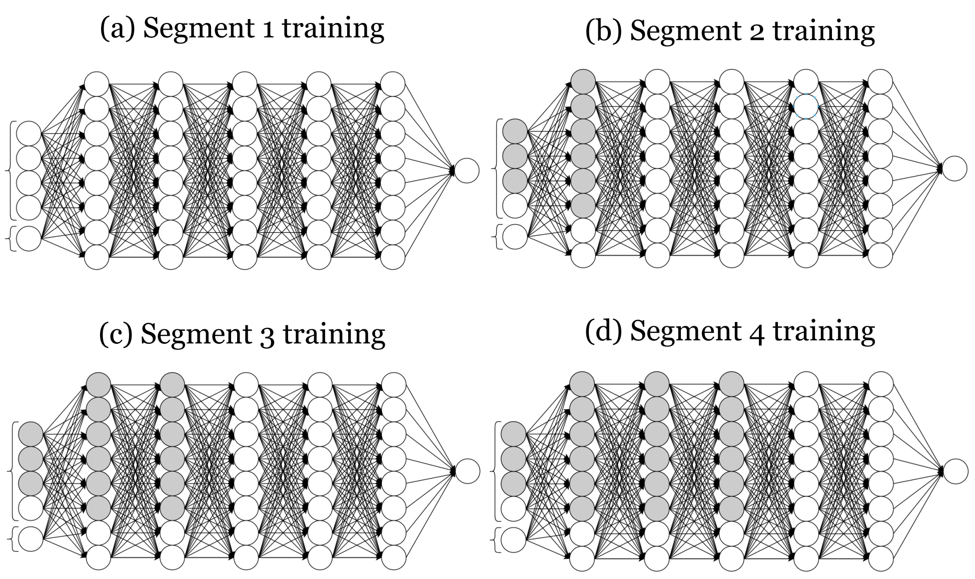


Figure 3.2.5: Incremental freezing. Weights and biases between grey nodes are frozen

scheme used is the one shown in figure 3.2.3. During training of each segment, an equal amount of data is sampled from previous segments, as there is in the current segment. The layer freezing method used is the one shown in Figure 3.2.5. After training on a segment, a portion of the model is frozen. In this work, 75% of the nodes in each layer are frozen. The weights from the labels in the conditional GAN are never frozen and are not considered a part of the 75% that are frozen.

For training, the number of epochs during training on each segment can vary. For the first segments, the number of epoch can be relatively low, since learning those distributions does not take a long time and most of the weights are unfrozen. During training on the later segments, and especially the last segment, the epochs need to be increased. That is due to the fact that a majority of the weights have been frozen, and the fact that each epoch has a relatively low number of datapoints. For the Channel Estimate dataset, the first segment has 5.374.842 datapoint, but the last segment only has a 159 datapoints. Due to Balanced Replay there will be 159 datapoints sampled from segments 1-4, and that means that there are 795 datapoints used during segment 5 training. Due to that, the time that is saved by using fewer epochs during training on the first segments, is significantly higher then the time lost by having more epochs for the last segments. Due to the increased complexity of the problem in later segments, and the low number of datapoints, the model often need more then 1000 epochs to successfully learn the rare event distribution.

The convergence criterion used for training IL-GAN is the Jensen-Shannon Divergence (JSD). JSD will be explained in the next section. We will use early stopping as a way to determine when to stop training on each segment and when to stop training on the baseline GAN. The patience hyper parameter determined the number of epochs the model continues training without an improvement of the JSD. For each segment the patience hyperparameter is different. For the first segment the patience parameter is set very low. For the last segment the patience parameter needs to be set very high.

3.4 Methods for analysis

To analyze the outputs of each GAN, we will use the JSD, a measurement to compare probability distributions. JSD will be used to compare the generated distributions with the true distribution. JSD is based on KL-divergence but is symmetric and bounded to be inbetween 0 and 1, unlike KL-divergence.

$$JS(P,Q) = \frac{KL(P,M}{2} + \frac{KL(Q,M}{2} \tag{3.1}$$

where $M = 0.5 * (P + Q)$, P is the true distribution and Q is the generated distribution. $JSD = 0$ means that the distributions are the same, and $JSD = 1$ means that there are no common elements in the distributions.

Chapter 4

Experiments and Results

4.1 Experiments

This thesis will compare the generated data of a conventionally trained GAN with one trained using incremental learning. The comparison will be made using JSD, and the distribution of the generated rare events will be investigated.

Three experiments will be done

1. Generate Channel Estimate data
2. Jointly generate SINR and Delay
3. Jointly generate Interarrival time and Packet length

Three IL-GANs and three baseline GANs will be trained for each experiment and the average JSD was taken.

4.2 Results

4.2.1 Channel Estimate

The first experiment was generating Channel Estimate. The Channel Estimate data was normalized from the original range to be between 0 and 1. The patience hyperparameter for early-stopping was set at [1, 3, 5, 15, 2000] for the IL-GAN and 5 for the baseline GAN. On average, the baseline GAN used $6.67 * 10^7$ datapoins for training while the IL-GAN used $1.4 * 10^8$. So the IL-GAN used over twice the amount of datapoints as the baseline GAN.

Table 4.2.1 shows that the baseline GAN performs well on the first three segments. However, it performs poorly on the last two segments. Incremental GAN performs better on all segments, and most notably it shows significant improvement over the baseline GAN on the last two segments.

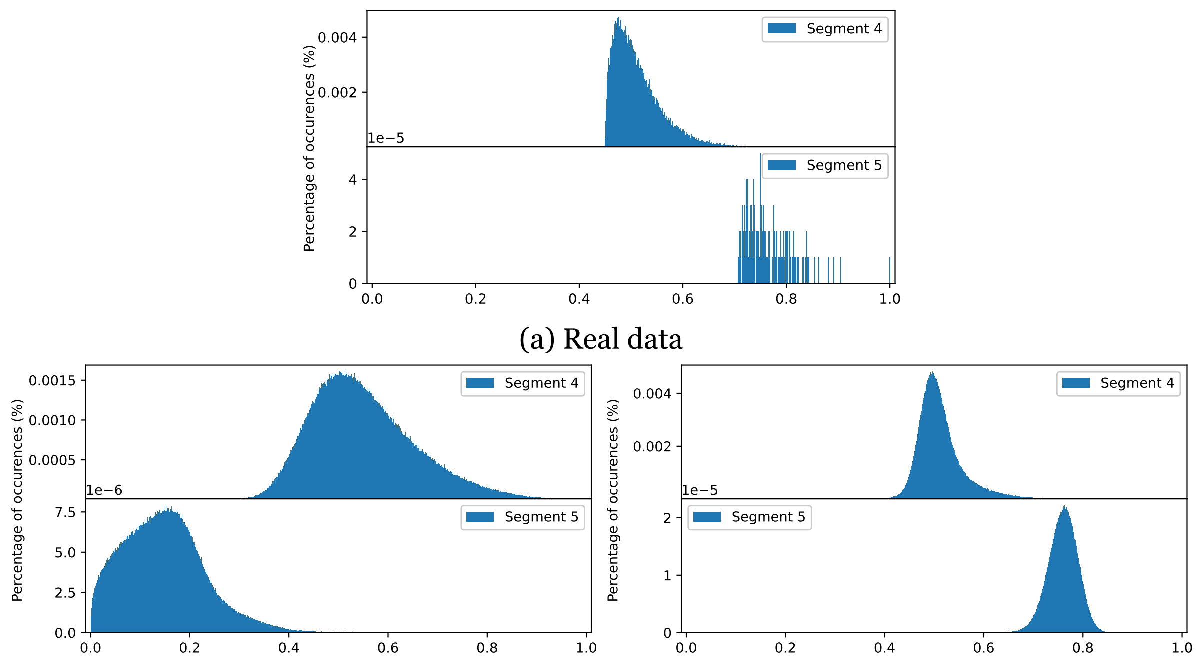
Table 4.2.1: JS error on the Channel Estimate dataset

Segment	**Normal GAN**	**Incremental GAN**
Segment 1	0.0211	0.0193
Segment 2	0.0524	0.0107
Segment 3	0.0636	0.0184
Segment 4	0.27	0.0548
Segment 5	1.0	0.264
Average	0.285	0.0777

Figure 4.2.1 shows the generation of rare events. For comparisons, both the baseline GAN and incremental GAN generated 1.000.000 data points from the rare event segments. As can be seen, the baseline GAN performed worse on the last two segments. The distribution of segment 4 is not comparable to the real distribution. On the other hand, incremental learning is very comparable to the true distribution of segment four. In segment 5, the baseline GAN has the distribution in the incorrect area and gets a JSD of 1.0. That means that the generated data distribution of the baseline GAN has no common elements with the true distribution. On the other hand, the IL-GAN has the distribution in the correct area. The generation of the rare events can not be fully judged using JSD because the GANs need to predict the true distribution of the rare events from only 159 data points. Nevertheless, we can see that the IL-GAN predicted distribution is reasonable and significantly improved over the baseline GAN.

4.2.2 Signal to Interference & Noise Ratio (SINR) and Delay

The second experiment was jointly generating SINR and Delay using a single GAN. The SINR data is normalized from the original range to be between 0 and 1. The Delay data is discrete, so it was categorized into four possible categories. The patience hyperparameter for early-stopping was set at [1, 1, 1, 10] for the IL-GAN and 5 for the



(a) Real data

(b) Baseline GAN

(c) IL-GAN

Figure 4.2.1: Channel Estimate segment 4 and 5

baseline GAN. On average, the baseline GAN used $1.95*10^8$ datapoins for training while the IL-GAN used $1.35*10^8$. So the IL-GAN used 70% of the datapoints the baseline GAN used.

Table 4.2.2 shows that the baseline GAN performs well on the first three segments of the SINR. However, it performs poorly on the last segment. IL-GAN also performs well on the first three segments of the SINR, having a lower JSD in segments 1 and 3. However, on the last segment, like in the previous experiment, IL-GAN shows significant improvement over the baseline GAN. Both the baseline GAN and the IL-GAN have low JSD on the delay dataset, but the IL-GAN has lower in segments 3, and 4.

Figure 4.2.2 shows the generation of rare events. For comparisons, the baseline GAN and IL-GAN generated 1.000.000 data points from the rare event segment. As can be seen, the baseline GAN performed worse on the last segment on the SINR. Like in the previous experiment, the distribution of segment 4 is not comparable to the real distribution. On the other hand, the IL-GAN distribution is closer to the true distribution of segment four. The baseline GAN has not learned the true distribution of the rare event section, as it generates data in all areas. The IL-GAN does better, but there are still some problems with the generated distribution. The generated

Table 4.2.2: JS error on the SINR & Delay dataset

Segment	**SINR**		**Delay**	
	Normal GAN	**Incremental GAN**	**Normal GAN**	**Incremental GAN**
Segment 1	0.0023	0.0093	0.0005	0.0019
Segment 2	0.0356	0.047	0.0003	0.0037
Segment 3	0.0391	0.0339	0.0018	0.0005
Segment 4	0.799	0.444	0.0069	0.0003
Average	0.219	0.133	0.0024	0.0016

distribution is not as believable as in the previous experiment.

For the Delay distribution on the last segment, the baseline GAN has a low JSD, but the generation leaks into category one. For the IL-GAN is slightly more accurate.

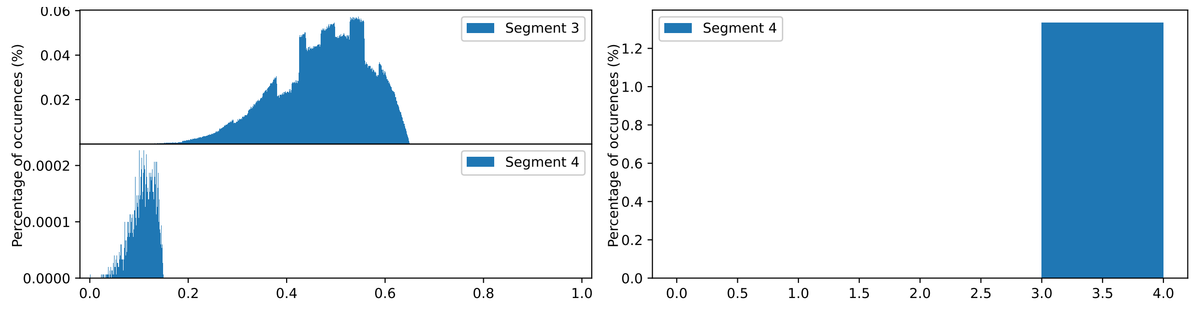
4.2.3 Interarrival Time (IT) & Packet Lengths (PL)

The third experiment was jointly generating IT and PL using a single GAN. The IT was split into segments, and each segment was normalized individually. The PL data was discrete, so it was categorized into 1368 categories. The patience hyperparameter for early-stopping was set at [3, 5, 10, 25, 500] for the IL-GAN and 10 for the baseline GAN. On average, the baseline GAN used $8.4 * 10^7$ datapoins for training while the IL-GAN used $8.7 * 10^7$. So the IL-GAN and the baseline GAN used a similar amount of data for training.

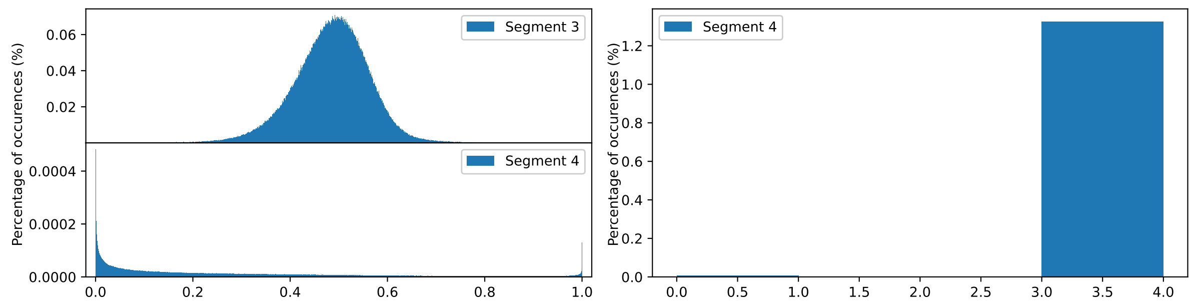
Table 4.2.3: JS error on the Interarrival time & Packet length dataset

Segment	**Interarrival time**		**Packet length**	
	Normal GAN	**Incremental GAN**	**Normal GAN**	**Incremental GAN**
Segment 1	0.0279	0.0168	0.0003	0.0003
Segment 2	0.0784	0.0492	0.509	0.186
Segment 3	0.1457	0.0908	0.707	0.180
Segment 4	0.1927	0.1627	0.6403	0.078
Segment 5	0.4393	0.2883	0.543	0.036
Average	0.1767	0.1217	0.4797	0.0963

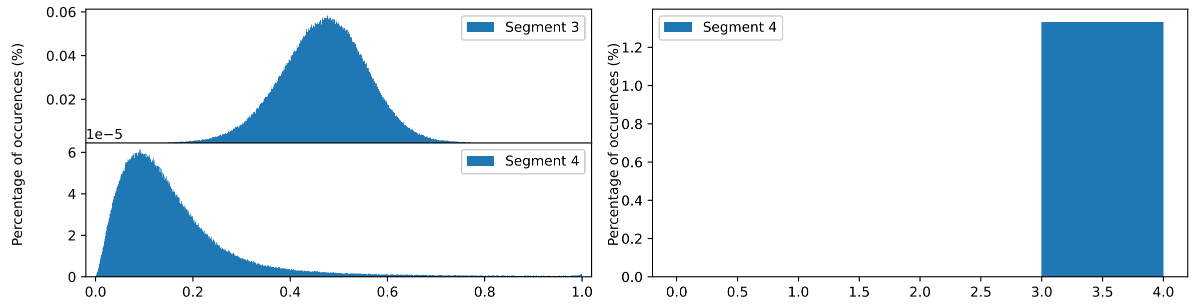
Table 4.2.3 shows that the IL-GAN performs better then the baseline GAN on all



(a) Real SINR & Delay

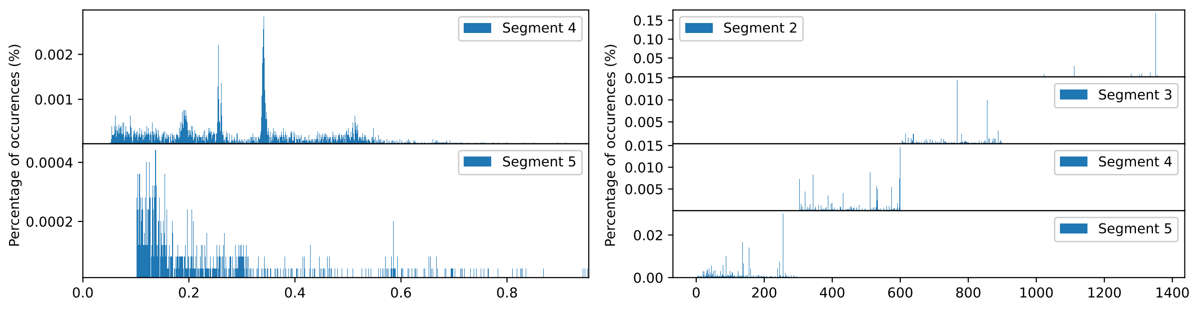


(b) Baseline GAN

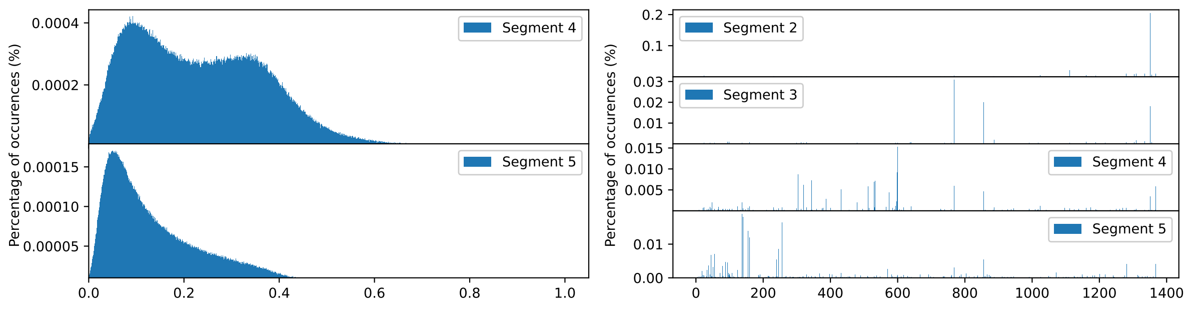


(c) IL-GAN

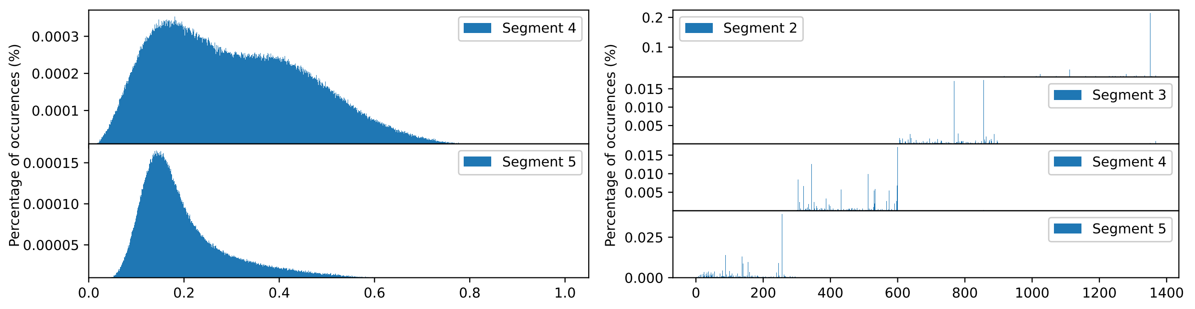
Figure 4.2.2: SINR and Delay segment 4



(a) Real IT & PL



(b) Baseline GAN



(c) IL-GAN

Figure 4.2.3: Interarrival time and Packet length segment 4 and 5

segments. The most significant improvement in all the experiments can be seen on the PL dataset. The baseline GAN does significantly worse on segments 2, 3, 4, and 5.

Figure 4.2.3 shows the generation of rare events. For the IT dataset, the baseline GAN generated data at and close to zero where there is no data in the dataset. The IL-GAN is close to the correct starting place of the distribution. Both the GANs fall short on generating data from all the distribution, but the IL-GAN covers more of the area that the dataset distribution lies in.

For the PL dataset, the baseline GAN has not managed to learn the segmentation of the rare event data in any meaningful way, while the IL-GAN generates data from the correct area in every segment. The improvement seen here is significant over the baseline GAN.

Chapter 5

Conclusion

In this work, we have shown that IL-GANs give better performance in generation of all segments on wireless communication data. The biggest improvement is on the rare event data, where the baseline model almost always fails to generate data from a likely distribution.

The generated data for SINR using IL-GAN was not as good as in other experiments. We hypothesize that the joint generation of SINR and Delay and the significant discrepancy in the number of datapoints in segment 4 caused those problems. While SINR only has 1712 data points in segment four, delay has almost 200.000 data points in the same segment. So during segment four learning, there are too many SINR values from other segments paired with the delay values in segment 4. A possible improvement for these cases is using an alternative method for Balance Replay than the one used in this work to keep a better balance between previous segments and the current segment.

For datasets such as the Interarrival time dataset, where the data can not be normalized to be between 0 and 1, finding a better method than the one provided in this thesis will be reserved for future work.

For the SINR & Delay and Incremental Learning (IL) & PL datasets, the IL-GAN used similar or less amount of data then the baseline GAN. For the CE dataset, the IL-GAN used over twice the amount of data. That means that the methods had a similar training for two of the three datasets, while IL-GAN had over twice the training time on one of the datasets. Since IL-GAN uses more epochs then the baseline GAN, some additional time is added between epochs to calculate the validation error.

Finding a better balance between the Balanced Replay, freezing and the number of epochs to decrese training time and improve learning is reserved for future work.

Bibliography

[1] 3GPP. *Service Requirements for cyber-physical control applications in vertical domains*. URL: `https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3528`.

[2] Arjovsky, Martin and Bottou, Léon. "Towards Principled Methods for Training Generative Adversarial Networks". In: (Jan. 2017). URL: `http://arxiv.org/abs/1701.04862`.

[3] Arjovsky, Martin, Chintala, Soumith, and Bottou, Léon. "Wasserstein GAN". In: (Jan. 2017). URL: `http://arxiv.org/abs/1701.07875`.

[4] Bennis, Mehdi, Debbah, Mérouane, and Poor, H. Vincent. "Ultra-Reliable and Low-Latency Wireless Communication: Tail, Risk and Scale". In: (Jan. 2018). URL: `http://arxiv.org/abs/1801.01270`.

[5] Elvira, Victor and Santamaria, Ignacio. "Multiple importance sampling for efficient symbol error rate estimation". In: *IEEE Signal Processing Letters* 26.3 (Mar. 2019), pp. 420–424. ISSN: 10709908. DOI: `10.1109/LSP.2019.2892835`.

[6] Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. "Generative Adversarial Networks". In: (June 2014). URL: `http://arxiv.org/abs/1406.2661`.

[7] Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent, and Courville, Aaron. "Improved Training of Wasserstein GANs". In: (Mar. 2017). URL: `http://arxiv.org/abs/1704.00028`.

[8] Hu, Xinting, Jiang, Yi, Tang, Kaihua, Chen, Jingyuan, Miao, Chunyan, and Zhang, Hanwang. "Learning to Segment the Tail". In: (Apr. 2020). URL: `http://arxiv.org/abs/2004.00900`.

[9] Isikdogan, Leo F, Nayak, Bhavin V, Wu, Chyuan-Tyng, Moreira, Joao Peralta, Rao, Sushma, and Michael, Gilad. *SemifreddoNets: Partially Frozen Neural Networks for Efficient Computer Vision Systems*. Tech. rep.

[10] Jang, Eric, Gu, Shixiang, and Poole, Ben. “Categorical Reparameterization with Gumbel-Softmax”. In: (Nov. 2016). URL: `http://arxiv.org/abs/1611.01144`.

[11] Kasgari, Ali Taleb Zadeh, Saad, Walid, Mozaffari, Mohammad, and Poor, H. Vincent. “Experienced Deep Reinforcement Learning with Generative Adversarial Networks (GANs) for Model-Free Ultra Reliable Low Latency Communication”. In: (Nov. 2019). URL: `http://arxiv.org/abs/1911.03264`.

[12] Kingma, Diederik P and Welling, Max. “Auto-Encoding Variational Bayes”. In: (Dec. 2013). URL: `http://arxiv.org/abs/1312.6114`.

[13] Mirza, Mehdi and Osindero, Simon. “Conditional Generative Adversarial Nets”. In: (Nov. 2014). URL: `http://arxiv.org/abs/1411.1784`.

[14] Owen, Art B., Maximov, Yury, and Chertkov, Michael. “Importance sampling the union of rare events with an application to power systems analysis”. In: (Oct. 2017). URL: `http://arxiv.org/abs/1710.06965`.

[15] Rebuffi, Sylvestre-Alvise, Kolesnikov, Alexander, Sperl, Georg, and Lampert, Christoph H. “iCaRL: Incremental Classifier and Representation Learning”. In: (Nov. 2016). URL: `http://arxiv.org/abs/1611.07725`.

[16] Sengupta, Satadal, Gupta, Harshit, Ganguly, Niloy, Mitra, Bivas, De, Pradipta, and Chakraborty, Sandip. *{CRAWDAD} dataset iitkgp/apptraffic (v. 2015-11-26)*. Nov. 2015. DOI: `10.15783/C77S3W`.

[17] Townsend, J Keith, Haraszti, Zsolt, and Freebersyser, James A. *Simulation of Rare Events in Communications Networks*. Tech. rep.