Synthetic Channel Generation using Generative Adversarial Networks

Master's thesis
Mads Bangshaab
Signal Processing
Aalborg University
4. June 2020

AALBORG UNIVERSITY
DENMARK

Aalborg University
Department of Electronic systems
Signal Processing
Fredrik Bajers vej 7
9220 Aalborg Øst
www.es.aau.dk

Title:

Synthetic Channel Generation using Generative Adversarial Networks

Theme:

Signal Processing and Computing

Period of the project:

1. Feb 2020 - 4. Jun 2020

Author:

Mads Bangshaab

Supervisors:

Carles Navarro Manchon
Sajad Rezaie

Pages: 54
Ended: 4. Jun 2020

Abstract:

In this report Generative Adversarial Networks (GAN) are designed and implemented with the purpose of generating synthetic responses of a radio channel. An investigation of relevant theory of neural networks and specifically GANs is conducted and based on this, choices with regards to architectures are made. For the purpose of training and evaluating the GANs, synthetic channel response data is generated based on a stochastic model. From this data relevant statistical properties are computed as to form a basis for evaluation. Two different realisations of a GAN are then designed based on the theory reviewed and relevant design choices made regarding the training data. These two realisations are based on a FCNN and a CNN respectively. Through testing it is seen, that the DCGAN has the best performance. The DCGAN is then evaluated with regards to the robustness of the network with regards to noise added to the training data and the size of the data set. Based on these findings, the model used for the DCGAN is then trained anew with another data set containing real measurements, where both training data is limited and noise are included in the measurements. Through testing of the GAN it is found, that though not achieving perfect results, the use of GANs for stochastic radio channel response generation is feasible.

Preface

This paper has been compiled by Mads Bangshaab as the Master's thesis under the main theme *Signal Processing and Computing*, at the Department of Electronic Systems, at Aalborg University, Spring 2020.

This paper is indexed in chapters chronologically numbered after the order in which they appear. Sections and subsections in chapters are numbered likewise, while sub-subsections are without index numbers. Figures, tables and equations are also indexed in numbers equivalent to the chapter and chronological order in which they appear, appendixes are lettered in alphabetical order, in which they appear.

For the purpose of simulations conducted in this project, Python 3.6.8 has been used as the main simulation tool. Tensorflow version 2.0.1 has been used as deep learning library. Furthermore Numpy version 1.18.2 and Scipy version 1.4.1 are used.

Mads Bangshaab
<mbangs14@student.aau.dk>

Contents

1 Introduction **1**

2 Problem statement **3**
2.1 Methodology . . . 4
2.2 Related work . . . 5
2.3 Outline of report . . . 6

3 Neural Networks **7**
3.1 The multilayer perceptron . . . 7
3.2 Multi layer percepton . . . 8
3.3 Activation functions . . . 9
3.4 Traning of neural networks . . . 11
3.4.1 Loss function . . . 11
3.4.2 Back propagation . . . 12
3.4.3 Training time . . . 13
3.5 Deep neural network . . . 14
3.5.1 Convolutional neural network . . . 14
3.5.2 Pooling . . . 15
3.5.3 Architecture of a CNN . . . 16
3.6 Generative adversarial networks . . . 16

4 Generation of synthetic data **19**
4.1 Generation model . . . 19
4.2 Data Generation . . . 22
4.2.1 Statistical representation of the synthetic data . . . 22
4.3 Data structure and pre-processing . . . 24

5 Network Design **25**
5.1 Design of fully connected generative adversarial network . . . 25
5.1.1 Discriminator architecture . . . 25
5.1.2 Generator architecture . . . 27
5.1.3 Design of generic fully connected generative adversarial network . . . 29
5.2 Design of deep convolutional generative adversarial network . . . 30
5.2.1 Discriminator architecture . . . 30
5.2.2 Generator architecture . . . 31
5.2.3 Design of generic deep convolutional generative adversarial network . . . 33

6 Training and testing of GAN **35**
6.1 Training and initial testing of FCGAN . . . 35
6.1.1 Pre processing of input data . . . 35
6.1.2 Training time . . . 35

6.1.3 Final model for the FCGAN . . . 38
6.2 Training and initial testing of DCGAN . . . 38
6.2.1 Pre processing of input data . . . 38
6.2.2 Training time . . . 40
6.2.3 Final model for the DCGAN . . . 41
6.3 Comparison of GAN models . . . 42
6.4 Robustness of network . . . 44
6.4.1 Impact of noise on the model . . . 44
6.4.2 Results with noisy data . . . 46
6.4.3 Results with limited data . . . 48

7 Evaluation with measured channel data **51**
7.1 Data set description . . . 51
7.2 Evaluation of DCGAN on the Lund data set . . . 51

8 Conclusions **53**

Bibliography **55**

1 Introduction

Radio communication is one of the most indispensable technologies of the modern society. The range of applications are wide, and stretch from pure convenience such as remote controls to mobile communications which can be essential en the case of emergency use. Radio communication is the obvious choice when the use of wired communication is not practical or feasible. Systems for radio communication exist for nearly any application, ranging from being just cable replacement to communication between Earth and distant satellites. Such radio communication systems consist of one or more transmitters that create a dynamic electromagnetic field and one or more receivers away from these transmitters that sense said field [1].

For the purpose of studying or designing communication systems, a conceptual simplification of such a model can be used, where the communication channel is modelled as just a simple transmitter and receiver linked by an unknown propagation environment, the radio channel. Such a model is depicted in Figure 1.1 [2][1].

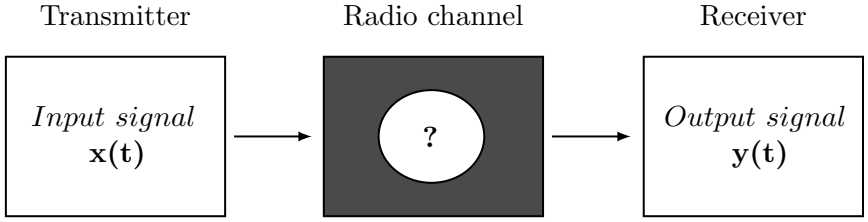


Figure 1.1: A communication system with one transmitter, a "black box" radio channel, and one receiver [2].

In this model a input signal $x(t)$ is applied to the transmitter and an electromagnetic wave is thus produced which propagates throughout an unknown environment. The electromagnetic wave then reaches the receiver and induces a signal $y(t)$ at the output ports of the receiver. This signal $y(t)$ received will be a transformed version of $x(t)$. This transformation that is suffered from the transmitter to the receiver is modelled by what is referred to as the radio channel. When referring to a radio channel, the response of said channel is often of interest, since this channel response holds the information for the transformation applied to the input signal. The radio channel is thus defined as the system, where the input signal $x(t)$ is applied and the output signal $y(t)$ is observed [2][1].

Designing such a communication system obviously relies on the requirements to the system with regards to input type, quality and reliability of the transmission, etc. These requirements are though not satisfactory to ensure that the design of the communication system is successful. It is in addition to these predefined requirements also necessary to obtain knowledge of the relation between the transmitted signal and the signal sensed at the receiver, since these will have a strong influence on whether the system's requirements can be fulfilled.

Therefore, it is very important to be able to characterize these channels well and, in addition, to be able to generate synthetic channel responses. These synthetic channel responses enables emulation of a communication system before a physical implementation, which means optimisation of the system is possible via computer simulations. Another more practical use is also testing of actual devices without the use of field test [1]. This obviously requires that these synthetic channel responses are able to represent the channels actually observed in practice accurately, since the better these synthetic responses are, the better a given system can be optimised.

Problem statement 2

For the purpose of emulating the channel response of a wireless communication channel and generating data from said channel response, the concept of Generative Adversarial Network (GAN) will be reviewed as a solution.

GANs have shown to have a wide range of applications with regards to generating data similar in structure and statistical properties of a given training set. Many fields of application, such as image generation, are very well studied and results have shown that GANs have very good performance for image data.

A less studied field is the performance of GANs in wireless communication and specifically generating of channel responses in wireless communication channels. This report seeks to examine different architectures of GANs and their performance in generating wireless communication channel responses. The data generated from the GAN will be reviewed based on statistical properties of the data set used to train the networks that the GAN consists of.

The study will be limited to only investigate time-invariant radio channels. This means that only the generation of a single snap-shot of the channel response is studied. Hence, the impulse response is modelled as a linear time-invariant system, which is represented with the random process $h(\tau)$, which has the form of

$$h(\tau) = \sum_{l} \alpha_l \delta(\tau - \tau_l) \tag{2.1}$$

Where the values α_l and τ_l are the gains and delays in the system, which will be modelled as random variables that take specific values for each realisation of the channel.

This leads to the following problem statement:

Can generative adversarial networks be used to generate channel responses of a complex communication channel?

From this problem statement the following research questions have been established.

- How do GANs differ in performance based on network structure and training procedure?
- Which types of channel responses can be better reproduced by GANs?
- How similar are the statistics of the GAN-generated responses to those of the original data set?
- How does the size of the data set used to train a GAN affect the performance?
- How does quality of the data set with regards to noise level impact the performance of the GAN?

2.1 Methodology

This section seeks to outline the methodology that will be used in order to answer the established problem statement and research questions.

For the purpose of establishing knowledge of the topic of GANs relevant theory is reviewed to form a basis on which to build experience with the design and implementation of GANs. Different realisations of GANs will then be reviewed, and they will be compared based on training dynamics and performance. For the purpose of increasing performance of these networks, it is investigated if different types of channel responses yield different levels of performance.

For the purpose of this evaluation of performance relevant statistical properties of the training data is also established. These properties are used to compare the quality of the GAN generated channel response data with regards to mimicking these statistics. This will form a basis on which the performance of the network can be evaluated.

Further evaluation of the network will be conducted by limiting the amount of training data available for the network. This will yield another performance measure of the network, which is based on the robustness of the GAN. A further evaluation of this robustness will be made by training the networks with training data where noise is added to the measurements. Thus an understanding of the impact of noise in the system can be obtained.

Finally the results obtained through testing will enable conclusions on the feasibility of using GANs for generating wireless communication channel responses.

2.2 Related work

The popularity of GAN has exploded in recent years since their emergence in 2014, and have been used for a very wide range of applications [3] [4]. A very widely studied field of applications for GANs is that of image processing. The framework has shown to work very well with many different tasks of image processing.

One of the very prominent applications is generating fake images through the use of GAN. Within the fake image generation application one highly discussed subject is the generation of faces. It has been seen that the use of a progressively growing GAN architecture shows very good performance with regards to image generation. For this application it has been seen, that the progressively growing architecture both reaches good results but also speeds up training time [5].

Another application for GANs is that of image restoration with a focus on key components. For the purpose of the reconstruction of face images, these key components could be mouth or eyes, since these contain large appearance variation. For this purpose it is seen that the GAN framework shows very good performance for this application as well [6].

The GAN framework's performance on image processing related task is very extensively studied. Other fields of application outside of image processing though not as extensively studied are also starting to emerge. This is also the case within the field of wireless communication.

One example of this, is simulation and imitation of RF communications signal. For this purpose generative replay of such signals have been used to produce examples of signals with similar structure and properties of arbitrary signals. This enables the testing of radio systems with regards to deceiving and probing these and thus strengthen the system against attacks [7].

Another example is the use of GAN with regards to developing an end-to-end communication system. In such a system Deep Neural Network (DNN)s have been used for several key functions such as, encoding, decoding, modulation, and demodulation. For this purpose the channel transfer function has been estimated before hand in order for the networks to be optimised for their respective tasks [8].

Another application concerns one of the most critical studies in wireless communication, which is that of channel modeling. Here the stochastic behaviours of a radio channel is sought emulated through the use of GANs. This is evaluated based on the distributions of the output data from the training data compared to the generated data [9] [10].

In this project a similar approach to the two ladder approaches is taken, but with the use of a more extensive channel model than the additive white Gaussian noise channels attempted to recreate in the aforementioned work.

2.3 Outline of report

This section serves as a summary of the content of the chapters contained in this report.

In chapter 3 the concept of neural networks will be introduced and relevant theory with regards to the scope of the project will be examined. This theory section will serve as a reference point throughout the report when it comes to implementation and design choices.

In chapter 4 the model from which the synthetic channel response data used to train the GAN will be introduced and elaborated on with regards to parameters and assumptions in the model. Parameter values are chosen and the synthetic channel response data is then generated based on these choices. From the generated data statistical properties of the data set are established, on which future generative models will be evaluated. Lastly a description of how the data is structured and pre-processed is conducted.

In chapter 5 two different realisations of a GAN are designed. The design process and choices are based on the knowledge obtained in chapter 3 and the choices made with regards to the data set generated in chapter 4. These two GANs are generated in such a fashion, that the models are generic in the way that the parameters which are desired to be tested can be varied.

In chapter 6 initial testing of the two GANs is conducted and based of this a choice of network model for final testing is made. This test is evaluated based on performance with regards to the statistical properties of the training data established in chapter 4. The final model is then further evaluated based on the robustness of the network with regards to limiting the amount of training data and adding noise to the training data.

In chapter 7, a short description of a data set containing real world measurements is made. This data set is then used as training data for the best performing model from chapter 6. The model is thus further evaluated based on the performance on the real measurement data set.

finally in chapter 8, conclusions are drawn with regards to the results documented throughout the report. These conclusions and results will be compared to the general purpose of the report.

Neural Networks 3

This chapter will give a brief introduction to the concepts of neural networks. Relevant network architectures with regards to the purpose of this project will be reviewed, this includes network layers and activation functions.

3.1 The multilayer perceptron

The idea of an Artificial Neural Network (ANN), is as the name implies, to make an attempt of modelling the way a human brain functions. Nerve cells, which are also called neurons, are used in the brain for intercommunication. A simplification of such a neuron is, that based on the input applied to it, it either fires or does not. This makes them a sort of binary classifier, and this is what a ANN tries to model [11]. In an ANN, neurons are usually referred to as nodes. A model for a single node was described in 1957 by Frank Rosenblatt. This model was called the perceptron model [12]. In this model a node was described by a fixed non-linear transformation applied to a weighted sum of an input vector, x, as seen in (3.1).

$$y(x) = f(w^T x + b) \tag{3.1}$$

Where w is a vector containing the weights, x is the input vector, and b is a bias. The purpose of adding the bias is for the input transformation to be affine rather than linear. Having an affine transformation applied to the input enables shifting of when the perceptron fires independently of the input [13]. $f(x)$ is a non-linear transformation, which for the original percepton model is given as

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3.2}$$

A visualization of a single neuron with inputs $[x_1,...,x_n]^T$ with a weight w_i associated to every entry in the input vector, can be seen in Figure 3.1.

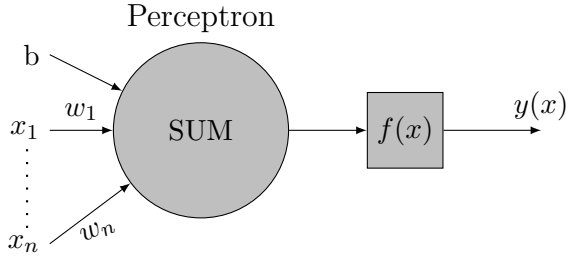


Figure 3.1: An illustration of a single neuron with inputs x_1 up to x_n and an added bias.

The perceptron thus performs a binary classification based on a linear decision boundary, where the boundary is determined by w and b [13]. If the input vector, x, is not linearly separable, the perceptron becomes an unsuited classifier [13].

3.2 Multi layer percepton

A method that enables classification of these non-linear separable problems is the Multi Layer Perceptron (MLP). This model is formed by connecting multiple perceptrons together [13]. In the MLP a single perceptron will be referred to as a node. The MLP, in contrast to the perceptron model, enables multi class classification [13].

The general architecture of an MLP is, that it consists of one input layer, a hidden layers and one output layer as illustrated in Figure 3.2 [13]. The number of nodes in the output layer is usually related to the number of classes in the training data. One very important aspect to the MLP, compared to the perceptron, is that the MLP must us continuous non-linear transformations in the hidden layers, rather than the step function described in (3.2)[13].

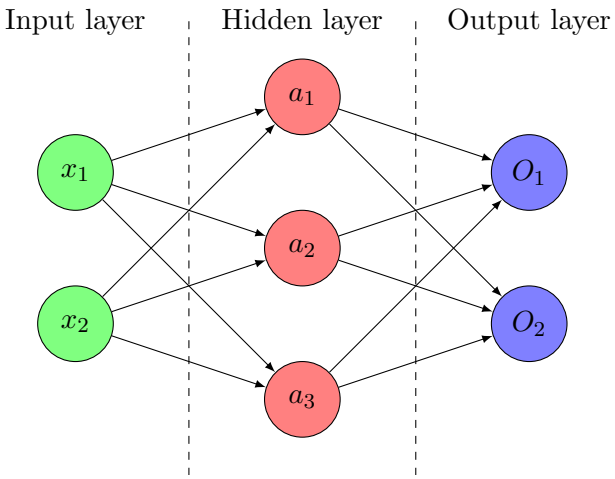


Figure 3.2: Illustration of an MLP with implicit weights and biases, two input nodes, three hidden nodes in one hidden layer, and two output nodes.

In Figure 3.2, it should be noted that weights and biases are implicit between the layers. The activation between the layers is thus given as

$$a = \sigma((w^{L-1})^T x + b^{L-1}) \tag{3.3}$$
$$O = \sigma((w^L)^T a + b^L) \tag{3.4}$$

where L is the number of layers, w^l, b^l are weight and biases for the L'th layer and σ is a differentiable continuous activation function. If the training of the network is successful, the output node O_n with the highest value compared to the other output nodes, will indicate that the input belongs to class n [13]. A normalized activation function is very commonly used on the output nodes such that these all sum to one. This makes the output relateable to a probability of an input belonging to a specific class. The MLP is often referred to as a Fully Connected Neural Network (FCNN), since all the nodes are connected to all the subsequent nodes [13].

3.3 Activation functions

In any ANN the activation function of a given node defines the output of said node when an input is applied. A fixed mathematical operation is performed by this activation function on the input it receives. As stated earlier MLPs must use non linear activation functions. If linear activation functions would be used, which means that the subsequent layers and nodes would just perform linear transformations on the previous layer. Thus the entire MLP could be reduced to a single node which would be a single linear transformation. In this section an overview of the activation functions used in this project will be given.

The Rectified Linear Unit (ReLU) activation function

One of the most commonly used activation functions is the ReLU.

ReLU is given as

$$\sigma(x) = \max(0,x) \tag{3.5}$$

ReLU is computational efficient, and it has shown to train faster than both the logistic sigmoid and the hyperbolic tangent function, which are two other commonly used activation functions [14]. For $x \neq 0$ the gradient of ReLU is either 0 or 1, meaning that the gradient never saturates due to the depth of the neural network. It does however mean that the activation of many nodes could be 0 instead of just numerically small, resulting in "dead nodes". An illustration of the ReLU function can be seen in Figure 3.3.

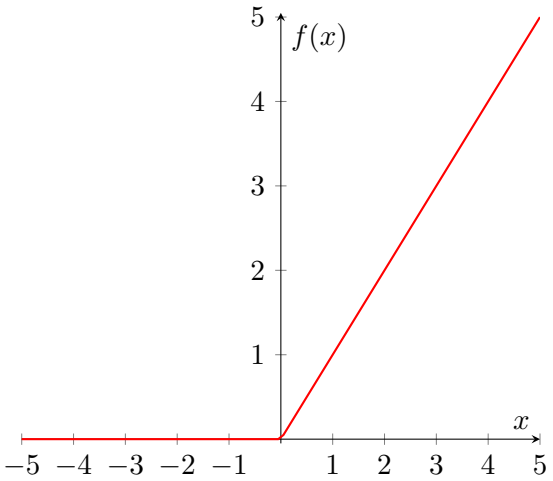


Figure 3.3: The ReLU activation function.

It should be noted that though a continuous function, ReLU is not differentiable at $x = 0$. As stated earlier it is required for activation functions to be differentiable. For implementation purposes, the gradient for $x = 0$ is therefore defined to either be the same as for $x < 0$ or $x > 0$.

Leaky ReLU activation function

Another commonly used activation function is leaky ReLU. Here the problem about gradients being zero instead of just numerically small in normal ReLU is countered by introducing a slope, a, of the

function when x is below zero [14]. leaky ReLU is therefore given as:

$$\sigma(x) = \max(a \cdot x, x) \tag{3.6}$$

An illustration of the leaky ReLU function can be seen in Figure 3.4.

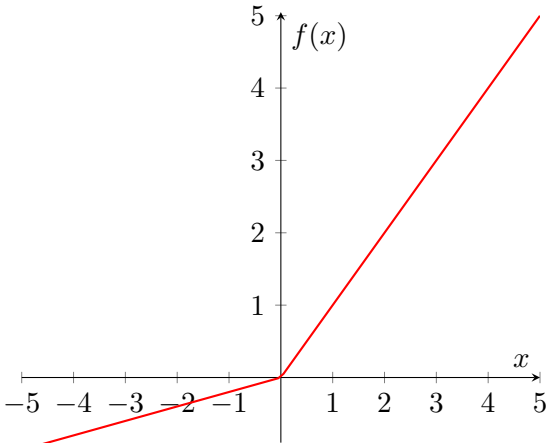


Figure 3.4: The leaky ReLU activation function with a slope, $a = 0.2$.

The Softmax activation function

The Softmax activation function is a commonly used output activation function. Softmax turns numeric output of the last linear layer of a multi-class classification neural network into what can be interpreted as probabilities of the input belonging to a certain class. This is done by taking the exponents of each output and then normalizing each number by the sum of those exponents so the entire output vector adds up to one. The Softmax activation function is thus given as

$$S(y_i) = \frac{exp(y_i)}{\sum_j exp(y_j)} \tag{3.7}$$

An illustration of the Softmax function can be seen in Figure 3.5.

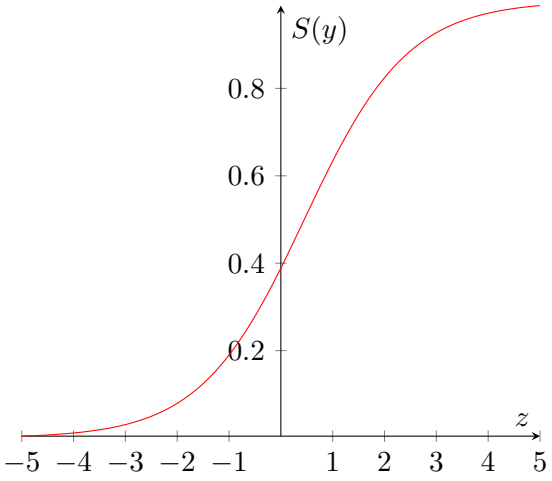


Figure 3.5: The Softmax activation function.

3.4 Traning of neural networks

Now that the fundamentals of neural networks have been reviewed, the process of training them will be described. The training of neural networks most often fall under the category of supervised learning. This means that the network is evaluated based on the labels of the training data. This section will thus investigate the basics of supervised neural network training.

3.4.1 Loss function

In order to evaluate the neural network, a measure of the performance must be introduced. Here what is known as a loss function is introduced. A loss function is a function that tells how good the performance of a neural network for a certain task is. An intuitive way of doing so is seen in (3.8).

$$L(y,\hat{y}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{3.8}$$

Here the sum of squared differences between the actual value or label, y_i and the output of the neural network, $\hat{y}_i$, is found for m inputs. This is just one of many loss functions that can be used for the evaluation of a neural network [14].

Since the purpose of this project can be seen as a binary classification problem, a loss function for binary classification is reviewed. One of the most common loss functions used for binary classification is that of binary cross-entropy. This loss function is given as

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{3.9}$$

Where y is the label, which is usually either 1 or 0, $p(y)$ is the value predicted by the network, and N is the number of inputs. From this it can be seen, that for label= 1, $log(p(y))$ is added to the loss.

Conversely it adds $log(1 - p(y))$ to the loss for label= 0. As stated the purpose of this project is an example of binary classification, and therefore it is chosen from this point on only to work with binary cross-entropy as loss function.

3.4.2 Back propagation

It can be see from (3.3) and (3.4) that the output of the neural network heavily relies on the weights and biases associated to the network. It is desired that these weights and biases are computed in such a way, that the difference between the predicted class and the true class is minimized. This also means minimizing the loss function, which for this purpose must be differentiable. For the case of differentiable continuous loss functions, the minimisation problem is a well defined problem. This problem is on the other hand not trivial to solve and may include many local minima. Instead of minimising as one big problem, the backpropagation algorithm is used. Backpropagation evaluates the error from layer to layer [13].

This is done by computing the partial derivatives of the loss function, which for notation purposes in this section will be denoted, C, with regards to all the node's input weights and biases, $\frac{\delta C}{\delta w}$ and $\frac{\delta C}{\delta b}$. These derivatives yield a measure for two things, in which direction weights and biases are to be changed and which change results in the largest decrease of the loss function [13].

For the purpose of referring to these weights and biases the following notation is used.

- w^l_{jk} denotes the weight from the k'th node in the $(l-1)$'th layer to the j'th node in the l'th layer
- b^l_j denotes the bias for the j'th node in the l'th layer.
- a^l_j denotes the activation of the j'th node in the l'th layer.

From this notation the activation a^l_j can be related to the activation a^{l-1}_j, as seen in (3.10).

$$a^l_j = \sigma\left(\sum_k w^l_{jk} \cdot a^{l-1}_k + b^l_j\right) \tag{3.10}$$

This can be written in a compact vectorised form where each element in the vector corresponds to a node as seen in Equation 3.11[15].

$$a^l = \sigma((w^l)^T a^{l-1} + b^l) = \sigma(z^l) \tag{3.11}$$

$\sigma()$ denotes an element wise application of the activation function.

With this notation established, it is posssible to describe the backpropagation algorithm, which is build around four steps [13] [15].

In the first the step, th error, δ^L, of the output layer is calculated and is seen in (3.12) [15].

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{3.12}$$

Where $\odot$ means an element wise multiplication, $\nabla_a C$ is a vector that consists of the partial derivatives of $\frac{\partial C}{\partial a_j^L}$, and $\sigma'(z^L)$ is the derivative of the activation of z^L. This error, δ^L can be seen as a measure of how much a change in z^L has on the loss function. This means that if δ^L, a change z^L contributes to little change in the loss.

The second step, the computed error of the output layer is propagated back the layers. This is done as seen in (3.13)[15].

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{3.13}$$

For this equation, w^{l+1} is the weight matrix in the $(l+1)$'th layer. The rows in w^{l+1} each consist of the weights associated with the j'th node in the $l+1$'th layer for every node in the l'th layer.

Similarly to the first equation, δ^l is a measure of the change in error given some δ^{l+1}, w^{l+1} and $\sigma'(z^l)$.

When combining the two equations from (3.12) and (3.13), the error, δ^l, can be computed for every layer in the network. (3.12) is firstly used to compute δ^L and then (3.13) propagated these errors back through the layers.

When all the errors, δ^l, in the network are computed, the following two equations, (3.14) and (3.15) , are used to compute at what rate the loss function is changed with regards to any bias, b_j^l, and any weight, w_{jk}^l [13] [15].

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3.14}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \cdot \delta_j^l \tag{3.15}$$

The actual update of these weights and biases depends on what optimiser algorithm is used. There are many different optimisers suitable for such a task, where gradient based optimisers are usually used [14]. It is not deemed necessary for the scope of this project different to review different optimisers. For implementation it has been chosen to use the optimiser, Adam. This has been chosen since Adam has shown to be able to train neural networks in less time and more efficiently than other common optimisers [16]. Since the purpose of this project is a feasibility study, it is deemed valuable to be able to train fast and thus it is not considered if other optimisers might yield better results over longer training time. Therefore Adam is chosen.

3.4.3 Training time

Another aspect that has a big impact on the performance of the network is how long it is trained. This means the number of epochs the networks are trained for. An epoch is a term used for when a neural network has been trained one time on every sample from the data set. This is though not only thing that decides how much a networks is trained. For this, batch size is also a very relevant factor. Where the number of epochs decides how many times the full data set is fed through the neural network the batch size dictates how many gradient updates are made every epoch. The batch size is a measure of how many samples from the data set is fed through the network, before the loss function is applied to

the results and gradients of the network are updated. It is apparent that the number of epochs and the batch size both have a big impact on the training time for the network. Therefore it is desirable to find a balance between the number of epochs and the batch size. There is no perfect way of finding this balance, and thus the number of epochs and the batch size is usually determined experimentally and evaluated with regards to performance measures [14].

3.5 Deep neural network

As described in section 3.1, the MLP is a model for an ANN. The definition of this model is, that it has at least three layers; input-, hidden-, and output layer. Introducing more hidden layers to the MLP will result in having a DNN. The ”depth” of such a network is a measure of the number of hidden layers. There are many ways of choosing the number of hidden layers and the nodes in each layer when designing DNNs. This includes different types of DNNs, e.g. convolution neural networks. The choice of architecture for a DNN is very dependent on the application. Due to the scope of this project, this section will only review the deep neural network architecture of a Convolutional Neural Network (CNN) and some of the key layers of this type of network. The architecture behind a deep FCNN will not be reviewed in this section since it simply consists of just adding more hidden layers to the MLP model.

3.5.1 Convolutional neural network

A CNN is on of the most commonly used DNNs. CNNs have show to work very well with spatial data such as for example images, but has also shown very usefull for many other applications. CNNs have been extensively studied since their origin in the 1980s, and they are still heavily used today [17][14].

There are many different possible realizations of a CNN. What all these have in common is, that they are based on the mathematical operation, convolution. The convolution operation is though rarely used for implemenation purposes. Instead what is known as the cross correlation operation is used. These two operations almost identical, and the only difference is, that the convolution operation flips the kernel relatively to the input, which is not the case for the cross correlation operation [14]. The cross correlation will be used throughout this report and will henceforth also be referred to as convolution.

Given an input matrix, I, a kernel K, the output of the cross correlation operation, S, can be computed as seen in (3.16).

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot k(m,n) \tag{3.16}$$

For the purpose of this project the cross correlation will be used throughout the report, but will henceforth also be referred to as convolution.

Kernels are of finite size, and it is chosen to define all values of the kernel $k(m,n) = 0$ for m and n pointing outside the defined input. This means that convolution operations are only made with “valid” kernel positions, which means that the entire kernel must be placed inside the input dimensions at all times. The kernel is moved across the input with some chosen stride. A stride is defined as a step

with size of the number of columns and rows the kernel moves across after each performed convolution [14]. This is illustrated for a 4×4 input with a 2×2 kernel and a stride of 2×2 in Figure 3.6.

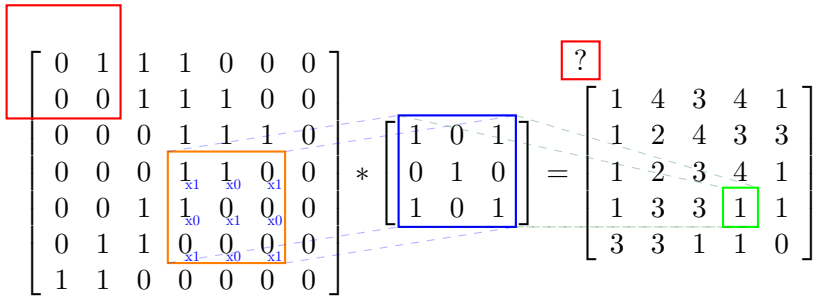


Figure 3.6: Illustration of 2D convolution. The output is a weighted sum of neighbourhoods in the input. The weights are given by a kernel.

The kernel performs these operations in step sizes of the chosen stride until all valid kernel positions have been covered and convolved. Each convolution layer consists of a number of kernels, which is defined by the number of filters chosen. Each of these filters has a specific kernel assigned to it, and all kernels perform the convolution independent of each other. The purpose of this is that each kernel is meant to capture some feature of the input, which could be something like and edge or something more complex. Thus the number of kernels in a layer defines how many different features may be extracted from the input by that specific layer [14].

3.5.2 Pooling

Another layer very commonly used in CNNs is a pooling layer. The pooling function's purpose is to reduce the dimension of the output data from the previous layer. The purpose of this is to make sure, that small changes in the input does not affect the output, and thus it makes the network more robust. There are many different kinds of pooling function, and some of the more commonly used functions are max pooling and average pooling.

The max pooling operation is very simple. Within a chosen kernel size it simply slides across the input and returns the largest number for each kernel position. This results in the output of the pool to just be a downsampled version of the input to the function. An illustration of the function, max pooling, with a kernel of 2×2 and a stride of 2 can be seen in Figure 3.7.

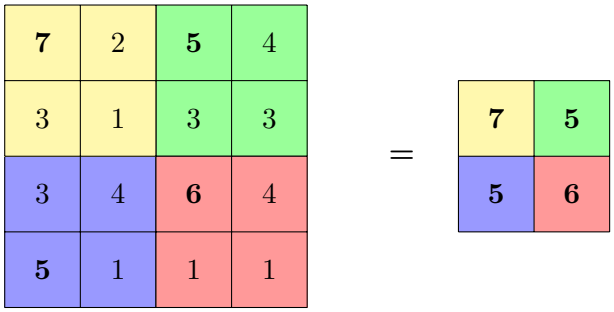


Figure 3.7: Diagram illustrating max pooling, of a 4x4 feature map by a 2x2 kernel and a stride of 2. Finding the largest value in each section (different color).

Average pooling works in the exact same way as max pooling, except it returns the average of the numbers inside the kernel. In general the most important property of pooling, is the reduction of dimensions since it both makes the network more robust, but it also makes computation more efficient.

3.5.3 Architecture of a CNN

As stated earlier a CNN can have many different realisations. A simple architecture for a CNN used for classification can be seen in Figure 3.8.

Figure 3.8: A simple architecture for a convolutional neural network.

This architecture consists of just one convolution layer for feature extraction, A pooling layer is used for dimensionality, and one fully connected layer for additional weights and better classification in the system. Lastly the softmax function is used as output activation.

3.6 Generative adversarial networks

The GAN is a fairly new type of neural network architecture proposed in 2014, and has quickly become exceedingly popular for many research purposes [4][3].

The idea behind the GAN architecture is, that unlike conventional neural networks it takes up a game-theoretic approach. This approach is based on a two-player game where the network is trained to generate data from a training distribution. In this game there are two entities, the generator network and the discriminator network. These two networks are the adversaries in this game and throughout the training process the networks improve themselves in order to outsmart the other network [4]. The game can be summarised as seen in Figure 3.9.

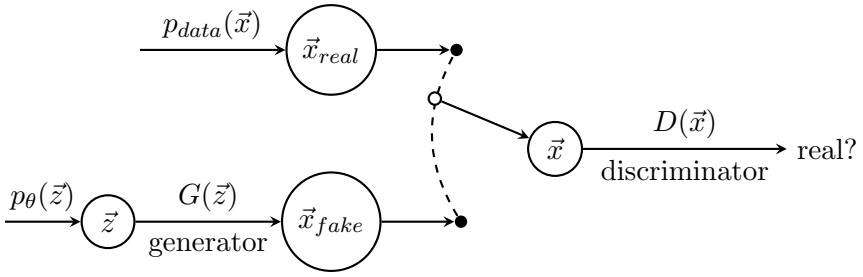


Figure 3.9: Illustration of a general GAN architecture.

Here the generator receives a random noise input vector, $\vec{Z}$, from which it generates a fake data output vector, $\vec{x}_{fake}$. The discriminator network either receives a data vector from the real data, $\vec{x}_{real}$ or from the fake data, $\vec{x}_{fake}$, and then it performs binary classification on the input [4].

As so heavily implied in the names of the two adversaries the discriminator seeks to classify input data and the generator seeks to generate and replicate input data. The training process for the GAN is, that the discriminator network is fed an input from either the ”real” training data set or the ”fake” data set generated from the generator network. The discriminator network’s task is then to

distinguish between the data coming from the training data or the generator network. The weights in both the generator and the discriminator is the updated according to the performance of each network. The discriminator's performance is obviously evaluated on the basis of how many samples it misclassifies, but what is maybe a little less obvious, is how the generator's performance is evaluated. The generator's purpose is to fool the discriminator into predicting that the samples produced by the generator actually comes from the real training data. Therefore the generator is evaluated according to how many samples it produces that the discriminator misclassifies [4].

Since a game-theoretic approach is taken, our objective function is represented as a minimax function. In order to learn the generator's distribution p_g a prior on the input noise is introduced. This is denoted, $p_z(z)$. Then a mapping to data space is represented as $G(z)$, where G is a differentiable function represented by a neural network. This is what was previously defined as the generator. We also define a second neural network, $D(x)$, that outputs a single scalar. $D(x)$ can be interpreted as the probability of x belonging to either the training samples or the samples generated from G. This second network is what was previously defined as the discriminator. This adversarial model is most straight forward to apply if both D and G are MLP[4].

Both these networks are trained but with different purposes. D is trained to maximize the probability that the correct label is assigned to the received input. Simulataniously, G is trained to minimize $log(1-D(G(z))$. This yields the following two-player minimax game, with the value function, $V(G,D)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[logD(x)] + \mathbb{E}_{z\sim p(z)}[log(1 - D(G(z)))] \tag{3.17}$$

In practice this may not provide sufficient gradient for G to learn well in the early stages of training, where G has poor performance and D can easily reject the samples. This means that $log(1-D_d(G(z)))$ saturates. A solution to this is to train G to maximize $logD(G(z))$, this does not change the dynamics of G and D, but provides much stronger gradients in early learning [4].

Though most straight forward to implement as consisting of two competing MLPs, the GAN architecture may also be implemented consisting of other types of networks. One very common implementation is using convolutional networks since they have shown to work well with spatial data such as images, but also show very good performance in general for a wide range of applications [14].

Generation of synthetic data 4

For the purpose of designing and training the GAN, synthetic data of a communication channel response is generated. This data will serve as a preliminary test as to whether GANs can be used for the generation of channel responses in a communication system.

4.1 Generation model

The synthetic data will be generated based on the Turin model for multipath propagation [18]. The model can be used to characterize an indoor, reverberant environment, which will be the use for this project. As stated in chapter 1, the radio channel is something that maps $x(t)$ to $y(t)$. For the purpose of this the radio channel is modelled as a linear time invariant filter as seen in (4.1).

$$y(t) = \sum_{l} \alpha_l x(t - \tau_l) \tag{4.1}$$

Where the sum is performed over the number of multipath components, l, α_l is a complex gain associated to the l'th multipath component, and τ_l is the delay associated to the l'th component. This describes a system, where $y(t)$ is a sum of all the l delayed versions of the signal $x(t)$.

From this the impulse response of the filter is given as

$$h(\tau) = \sum_{l} \alpha_l \delta(\tau - \tau_l) \tag{4.2}$$

From this impulse response, the transfer function, H_k, for the frequency response of the multipath channel is given as

$$H_k = \sum_{l} \alpha_l exp(-j2\pi\Delta f k \tau_l) \tag{4.3}$$

The transfer function is a sum over the number of multipath components, l, which is given as a Poisson process with mean, $\mu_l = \lambda_0 \tau_{max}$. Where λ_0 is the intensity of the first multipath component, which is constant i.e. $\lambda(\tau) = \lambda_0$, and τ_{max} is the maximum delay in the channel. The parameters k and Δf denote the number of measurements and the frequency separation between two samples respectively [18]. From this the bandwidth is given as $B = \Delta f k$.

The coefficients α_l and τ_l are the complex gain and time-delay of the l'th component, where the delays τ_l form a homogeneous Poisson point process with arrival rate $\lambda(\tau) = \lambda_0$. The gains α_l, conditioned on the delays, are modelled as a circular complex Gaussian variable with variance $\sigma_\alpha^2(\tau)$ [18]. The complex gains are thus distributed according to

$$\alpha = \mathcal{CN} \sim (0, \sigma_\alpha^2(\tau)) \tag{4.4}$$

The variance of the Circular complex Gaussian variable can thus be estimated as.

$$\sigma_\alpha^2(\tau) = \frac{P_h(t)}{\lambda_0} \tag{4.5}$$

The power delay spectrum for this type of model is given as

$$P_h(t) = \sigma_\alpha^2(\tau)\lambda_0 \tag{4.6}$$

For indoor measurements, the power delay spectrum has a exponentially decaying behaviour which can be approximated as follows:

$$P_h(t) = \begin{cases} G_0 exp(-\frac{\tau_l}{T}), & \tau_l > \tau_o \\ 0, & otherwise \end{cases} \tag{4.7}$$

Where G_0 is the reverberation gain and T is the reverberation time [19].

From the transfer function in (4.3) and the establishment of the parameters, the model is implemented in Python according to Algorithm 1.

Algorithm 1

Initialise:
$K = 801$, $T = 10\text{ns}$, $G_0 = 10^{-8}$, $\lambda_0 = \frac{1}{10^{-9}}\text{ns}$, $B = 4\text{GHz}$,
$\Delta f = \frac{B}{K}$, $\tau_{max} = \frac{1}{\Delta f}$, $\mu_L = \tau_{max}\lambda_0$ ▷ Initialising parameters
procedure
    $L = Poisson.rvs(\mu_L)$ ▷ Number of components
    for l in range(L) **do**
        $\tau_l \sim \mathcal{U}(\tau_0, \tau_{max})$
        $\sigma_\tau^2 = \frac{G_0}{\lambda_0} \cdot exp(-\frac{\tau_l}{T})$
        $\alpha_l \sim C\mathcal{N}(0, \sigma_\tau^2)$
        for k in range(K) **do**
            $H[k] = \sum_l \alpha_l exp(-j2\pi\Delta f k \tau_l)$ ▷ Computes the frequency response
        end for
    end for
    return H ▷ Returns the frequency response for L components
end procedure

For the purpose of also reviewing a scenario where noise is included in the measurements, an extension of the transfer function is considered, where the noise is added for each of the 801 measurements. This is described as

$$H_{kN} = H_k + N_k \tag{4.8}$$

Where H_k is the noiseless transfer function data and N_k is the noise added to the $k'th$ measurement.

4.2 Data Generation

The data is generated according to the model described in section 4.1. The values chosen for the parameters that do not depend on other parameters can be seen in Table 4.1. These values are chosen based on the settings used in [19].

Parameter	**K**	**T**	G_0	**B**	λ_0
Value	801	10 ns	10^{-9}	4 GHz	1 ns^{-1}

Table 4.1: Values of the parameters that are not dependent on other parameter values.

For these parameter choices, simulations have been performed with 10000 realisations of the channel response. Where based on the chosen parameter values we get $\tau_{max} = 200$ns, $\Delta f = 5$ MHz and $\mu_L = 200$.

4.2.1 Statistical representation of the synthetic data

For the purpose of forming a statistical foundation for evaluating the data generated in a GAN, two statistics will be considered. Firstly what is known as the Power Delay Profile (PDP) will be used.

This is used since the exponentially decaying behaviour of the power delay spectrum is apparent, when the PDP is inspected. The average PDP of these 10000 realisations can be seen in Figure 4.1.

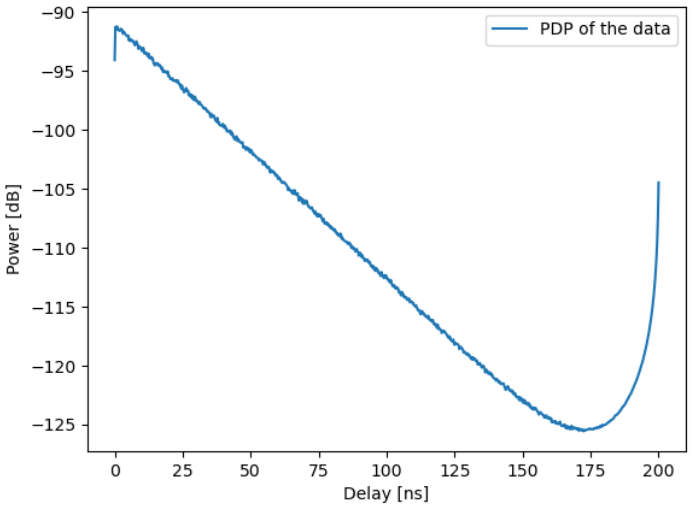


Figure 4.1: PDP of the average of 10000 realisations of the channel response.

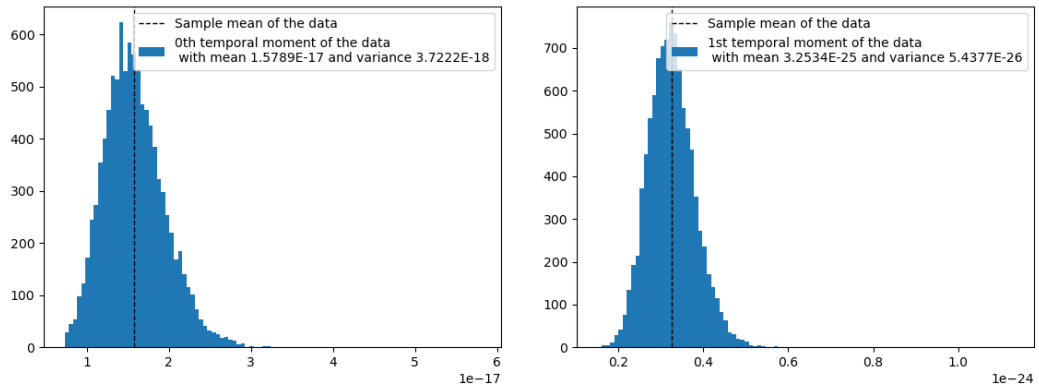
Thus it will be possible to evaluate GAN generated data based on whether or not it follows the same exponentially decaying PDP.

Furthermore, what is known as the temporal moments of the data will be used as evaluation. A particular realization of the channel response can be summarized in terms of its temporal moments. These temporal moments are a measure of how power is distributed over delays. For example the 0th

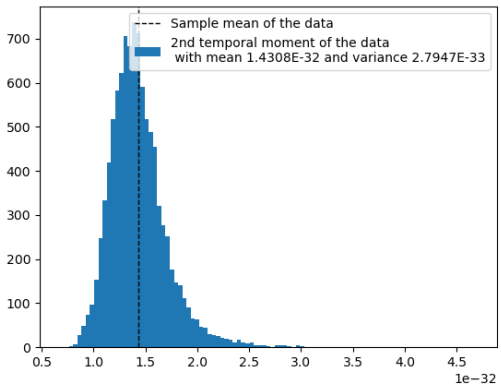
temporal moment is a measure of the instantaneous power received [20]. The temporal moments are computed as

$$m_k = \int_0^{\frac{1}{\Delta f}} t^k |y(t)|^2 dt, \quad k = 0,1,...,(K-1) \tag{4.9}$$

For each realisation k temporal moments are computed. This means that there is no averaging over multiple realisations. For the purpose of this project the first three temporal moments are considered. For the 10000 realizations of the synthetic data the three first temporal moments have been computed and can be seen in Figure 4.2.



(a) Histogram of the 0th temporal moment. (b) Histogram of the 1st temporal moment.



(c) Histogram of the 2nd temporal moment.

Figure 4.2: Histograms of the 0th, 1st and 2nd temporal moments computed from 10000 realisations of the channel response.

These temporal moments computed from the data set will in addition to the PDP throughout the report be used as a point of reference with regards to the statistical properties of the data set. The performance of the GANs trained and tested in this report will thus be evaluated based on these statistical properties of the data generated from the GANs.

4.3 Data structure and pre-processing

Considering the purpose of the data set it is desired to establish a structure for the way it is stored. The data set consists of multiple frequency responses and thus have complex parts associated to it. For the purpose of an implementation in a machine learning platform in python, it is necessary to work only with real numbers. Therefore each realisation of the frequency response is split into its real and imaginary part. From this a data structure is made as seen in Figure 4.3.

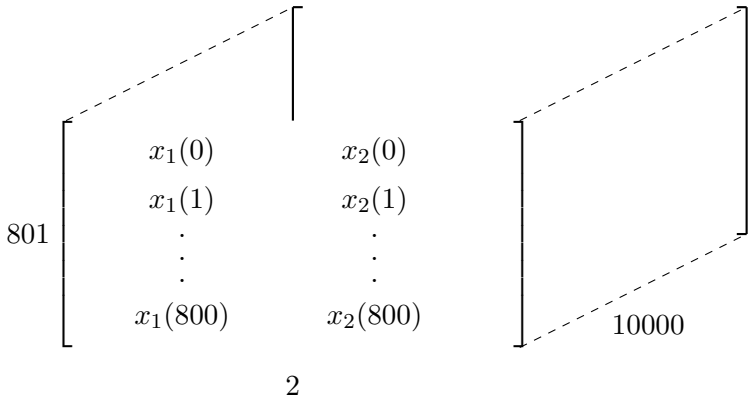


Figure 4.3

The data is organised in such a way, that each realisation of the channel response is organised in two columns, where each of these columns consist of 801 measurements. The total data set consists of 10000 of these realisations, and thus the total shape of the data set is $801 \times 2 \times 10000$.

For the purpose of pre-processing the data, two things are considered. Firstly the data is transformed to the time domain. This is done by taking the inverse Fourier transform of the frequency response. This is done to be able to compare the impact of training the networks with training data from both the time and frequency domain. The two data sets, containing the time and frequency domain data respectively, are stored. Secondly normalisation is applied to these data sets. This is done according to the standard score, which is given as, $\frac{x(k)-\mu(k)}{\sigma(k)}$. The data is normalised over the measurements, k, for all the realisations of the channel response. Thus each data point is normalised according to the corresponding measurement of all the realisations. The is done since normalisation of training data is shown to both decrease training time and increase performance [14].

Network Design 5

This chapter will describe the design process of different neural network architectures realised as GANs. For the purpose of the structure of this process, the discriminator - and generator networks will be designed separately. Afterwards the process of combining these networks to a realisation of a GAN will be described.

For the purpose of designing the networks, two different architectures will be used. The first choice of architecture is that of a FCNN. This is chosen to be studied initially because of the simplicity in the network structure, and thus it will form a basis of the study.

Secondly a CNN architecture will be reviewed, which will be based on the results from the FCNN. This architecture is investigated, since it is believed to be able to extract further information from the training data and thus improve the performance.

5.1 Design of fully connected generative adversarial network

From the universal approximation theorem, also known as the deep learning theorem it is stated that a feed-forward network with a single hidden layer containing a finite number of neurons can approximated any continuous function, under mild assumptions of the activation functions [13]. Therefore it seems appropriate to design a network based on a single hidden layer neural network model. Therefore the initial design of the Fully Connected Generative Adversarial Network (FCGAN) will take its base in an FCNN with a single hidden layer.

As stated in section 3.1 an FCNN is just a case of the MLP model. Though the deep learning theorem is based on a single hidden layer network with a finite number of nodes, this number of nodes might be very large. It is therefore chosen to review neural networks with more than one hidden layer as well. The number of hidden layers and nodes in different network realisations will thus be reviewed and compared based on the cost function described in section 3.6 and evaluated on the statistics from the training data described in subsection 4.2.1.

For the purpose of designing the neural networks, a generic architecture will be designed. This makes it possible to evaluate on number of layers and nodes in each layer throughout the training process.

5.1.1 Discriminator architecture

Since multiple hidden layers is also a possible solution for the realisation of the discriminator network, it is chosen to firstly design the input- and output layer of the network. When these two layers are established a generic architecture for the number of hidden layers is then introduced.

Input layer

When designing the input layer, it must be decided what the input dimensions to the layer must be. In section 4.3 the structure of the data set was established. This structure will form the basis of the input dimensions and thus also the input layer of the network. The shape of a single sample from the training data is given as, 801×2, where the two columns contain 801 measurements of the real part and the imaginary part of the channel response. Since the input to a FCNN must be one-dimensional, the two columns are concatenated and transposed to a single row vector with dimensions, 1×1602.

Output layer

Since the discriminator network is to act as a binary classifier, it is an output layer that is easily interpretable is desired. As stated in section 3.3, the Softmax function is a very commonly used output activation function. Having this as the output activation function, will enable the possibility of interpreting the output of the discriminator as a probability of the two classes, real data or fake data. Therefore the output layer will consist of a fully connected layer with a single node with the Softmax function as activation. This single node will thus act as a weighting of all the outputs from the previous layer, and from these outputs the Softmax function will output a probability, p, of the input belonging to the training data. The probability of the input belong to the GAN generated data is thus obtained, as $1 - p$.

Hidden layers

As stated earlier it is decided to make a generic architecture for the hidden layers in the network. There are, in principle, an infinite number of possible designs combining the number of hidden layers and the number of nodes per layer. This is of course not desirable, and therefore an approach of searching through this infinite space is devised. The approach of starting with a simple FCNN model with a single hidden layer is thus taken. This model is then evaluated on, and then an additional hidden layer is added to the model. This is continued until performance increase is not significant. For the number of nodes in each layer, another approach is taken. It is decided, that the output layer has a single node. From there the design is done in a backwards sort of fashion, where the first hidden layer before the output layer is chosen to have 256 nodes. Each prior layer is then chosen to have twice the amount of nodes as the subsequent one. This is done since it has shown, that having parameters in neural networks with values that can be described as 2^n, helps the network to train faster [14]. Since adding more hidden layers to the neural network will also mean adding more nodes to the network. This architecture results in the number of nodes in each layer will be decreasing by a factor of two until the output layer. It is deemed that having a lower bound of 256 nodes in each hidden layer will suffice, and thus it will only be of focus to change the number of hidden layers in the network. Leaky ReLU is used as activation function for each node in each hidden layer to introduce non-linearities.

Combining the input-, hidden-, and output layers yields the generic design for the discriminator, that can be seen in Table 5.1.

Specifications for the generic FCNN discriminator architecture	
Layers	**Parameters**
Input layer	Input size = 1×1602
N hidden layers	$\sum_{n=1}^{N} 256 \cdot 2^{n-1}$ total nodes, activation = leaky ReLU
Output layer	1 node, activation = Softmax

Table 5.1: Network specifications for the generic discriminator FCNN architecture.

A graphical illustration of the generic FCNN discriminator architecture can be seen in Figure 5.1. Weights and biases are implicit.

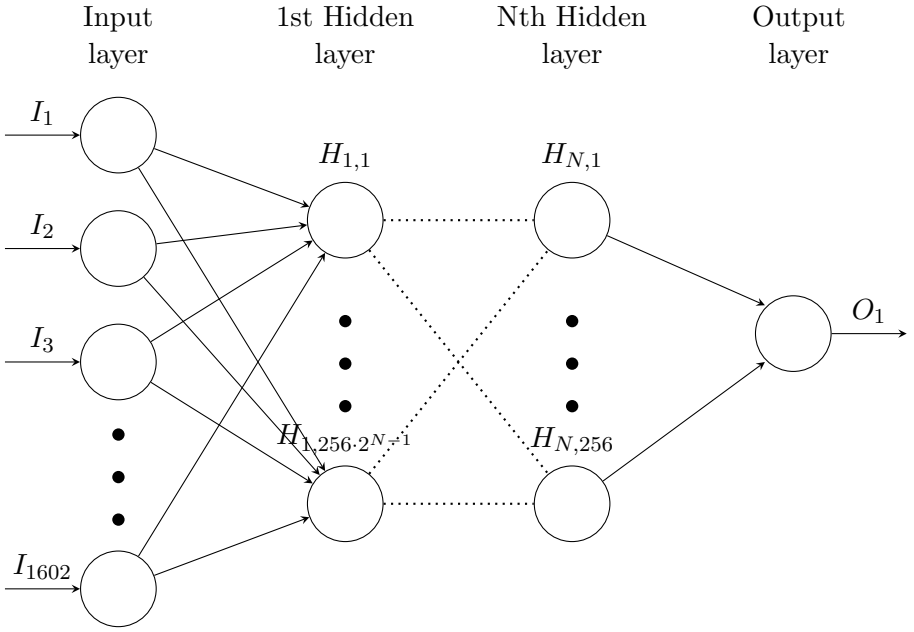


Figure 5.1: Illustration of the generic architecture for the discriminator of the FCGAN. Weights and biases are implicit.

5.1.2 Generator architecture

Similarly to the discriminator, it is chosen for the generator to firstly design the input- and output layer of the network, and then establish a generic architecture for the number of hidden layers.

Input layer

In contrast to the discriminator the input to the generator's input layer is not the training data. Instead random noise is used as the input as described in section 3.6. For the purpose of this project, the effect of the input dimensions of the noise is not investigated, and an arbitrary noise input size is chosen as 1×100.

Output layer

The output layer of the generator is designed to have an output size of the expected input size to the discriminator. This means that the output layer of the generator must produce an output with dimensions 1×1602. This is done by having the output layer consist of a fully connected layer with 1602 nodes. No activation function is used for the output layer.

Hidden layers

Similarly to the discriminator it is desired to design a generic architecture for the generator with regards to the number of hidden layers in the network. For the design of the hidden layers in the generator the same approach as with the discriminator is taken, but it is done in the opposite manner. This means that where the discriminator was decreasing with a factor of two in the number of nodes for each layer, the generator is designed to increase with a factor of two for each hidden layer. In a way this means that the generator architecture becomes the opposite of the discriminator. This intuitively makes sense, since it is desired for the discriminator to take all the information available from the data and compress it into a single guess, where as the generator's goal is to increase data dimension to produce a fake data sample. As with the discriminator, leaky ReLU is used as the activation for each node in each hidden layer.

Combining the input-, hidden-, and output layers yields the generic design for the generator, that can be seen in Table 5.2.

Specifications for the generic FCNN generator architecture	
Layers	**Parameters**
Input layer	Input size = 1×100,
N hidden layers	$\sum_{n=1}^{N} 256 \cdot 2^{n-1}$ total nodes, activation = leaky ReLU
Output layer	1602 nodes

Table 5.2: Network specifications for the generic discriminator FCNN architecture.

A graphical illustration of the generic FCNN generator architecture can be seen in Figure 5.2. Weights and biases are implicit.

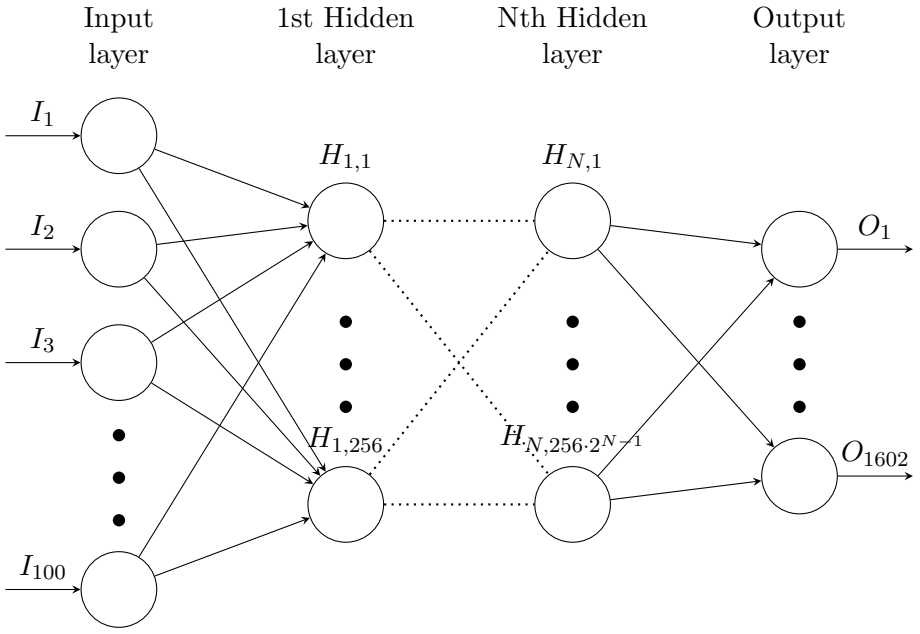


Figure 5.2: Illustration of the generic architecture for the Generator of the FCGAN. Weights and biases are implicit.

5.1.3 Design of generic fully connected generative adversarial network

Now that both the discriminator and generator networks are designed, it is straight forward to combine these two to make up the GAN. As described in section 3.6 and as seen in Figure 5.3, the concept of a GAN is simply the process of linking the two networks.

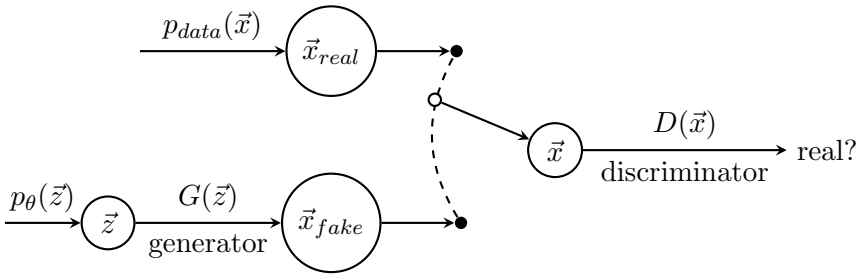


Figure 5.3: Illustration of a general GAN architecture.

As already briefly stated, the generator must produced outputs of the same size as the expected input of the discriminator, since it is from there the two models are linked. This is obvious since we want the generator to produce outputs similar to the training data. The training process of the GAN will be conducted as explained in section 3.6. For the purpose of training and evaluating both the discriminator and the generator the loss function of the system is chosen to be binary cross entropy, which is described in section 3.4. For the choice of optimiser to the system, Adam is chosen.

These choices combined with the design of the generic architectures for both discriminator and generator yields the following generic architecture for the FCGAN, which can be seen in Table 5.3.

Specifications for the generic FCGAN architecture	
Generator layers	**Generator parameters**
Input layer	Input size = 1×100
N hidden layers	$\sum_{n=1}^{N} 256 \cdot 2^n$ total nodes, activation = leaky ReLU
Output layer	1602 nodes
Discriminator layers	**Discriminator parameters**
Input layer	Input size = 1×1602
N hidden layers	$\sum_{n=1}^{N} 256 \cdot 2^n$ total nodes, activation = leaky ReLU
Output layer	1 node, activation = Softmax
Optimizer	**Loss function**
Adam	Binary Cross Entropy

Table 5.3: Network specifications for the generic FCGAN architecture.

Since the scope of the project is to focus on evaluating the feasibility of the approach at the conceptual level, rather than on finding a fully optimized network structure for the particular type of channel data, the number of hidden layers and nodes are not deemed necessary to fine tune. For the purpose of the project it is therefore chosen to not further investigate model parameters when satisfactory results are reached, though a more optimal choice of the number of parameters may exist.

5.2 Design of deep convolutional generative adversarial network

As mentioned in subsection 3.5.1, the fundamental building block of a CNN is the convolution operation. This operation makes the network able to extract spatial features of the input data. In order to improve on the concept of the FCGAN, it is desired to explore if adding convolutional layers to the model will yield better results. The stepping stone of this design will though still be the generic architecture for the FCGAN, since it is still desired to have weights for further feature extraction after the convolutional layers have extracted the "spatial" features. Another purpose of this is to add more weights to the network, since convolutional layers by themselves only consist of the kernel weights as explained in subsection 3.5.1. This architecture for the GAN will be referred to as a Deep Convolutional Generative Adversarial Network (DCGAN).

5.2.1 Discriminator architecture

As with the networks making up the FCGAN, it is desired to first establish the input- and output layer of the discriminator for the DCGAN.

Input layer

Unlike an FCNN it is possible for CNN to take inputs that are multidimensional. Therefore it is not necessary to concatenate the training data in order to use it as input. The data is though still transposed to yield an input shape to the network of 2×801.

Output layer

For the purpose of the output layer of the discriminator for the DCGAN, there will be made no changes to compared to the FCGAN. Thus this layer will still consist of a single node with a Softmax activation. This is due to the fact, that the output layer's purpose is still to only act as giving an interpretation of a probability of the input belong to the training data.

Hidden layers

As earlier stated, the DCGAN will be based on the architecture of the FCGAN. Therefore the generic discriminator architecture from Table 5.1 will be used as a basis for forming a generic architecture for the discriminator in the DCGAN. This means that for the number of hidden fully connected layers in the discriminator, there will be the N number of hidden layers chosen in the FCGAN, and these will still decrease with a factor of two each hidden layer, where the last hidden layer before the output layer will have a total of 256 nodes.

For the purpose of adding convolutional layers to this, it has been chosen to keep the architecture generic by not specifying the number of convolutional layers. It has though been chosen to keep a fixed number of filters in each convolutional layer to 64. This number has been deemed sufficiently large to extract initial features. This means that just like with the number of fully connected layers, it is also the number of convolutional layers that will determine the number of filters. The kernel size of the convolutional layers have been chosen to be 2×10 for the first convolutional layer and 1×10 for all subsequent layers. This is due to the fact, that dimensional reduction will occur from the convolution operation, and thus the two rows of data will be combined to one row after the first layer. The stride is fixed for all layers and is set to two. Leaky ReLU will be used as the activation function for all the convolutional and fully connected layers.

For the purpose of further dimensionality reduction and extracting most significant features, a max pooling layer is added after the last convolutional layer. This serves as a means of speeding up the training process but also has the purpose of extracting the most numerically significant measurements. The pool size is chosen to be 1×10 with a a stride of two. Having the kernel sizes and pool sizes fixed in the column direction and the strides fixed as well means, that the input will reduce half the size for each convolutional and pooling layer.

From these choices with regards to input, output-, and hidden layers, the following generic architecture for the CNN discriminator seen in Table 5.4 is made.

Specifications for the generic CNN discriminator architecture	
Layers	**Parameters**
Input layer	Input size = 2×801
K Convolutional layers	$K \cdot 64$ Filters, kernel size = 2×10 and 1×10, stride = 2
Max pooling layer	Pool size = 1×10, stride = 2
N fully connected layers	$\sum_{n=1}^{N} 256 \cdot 2^n$ total nodes, activation = leaky ReLU
Output layer	1 node, activation = Softmax

Table 5.4: Network specifications for the generic discriminator CNN architecture.

A graphical illustration of the generic DCGAN discriminator architecture can be seen in Figure 5.4.

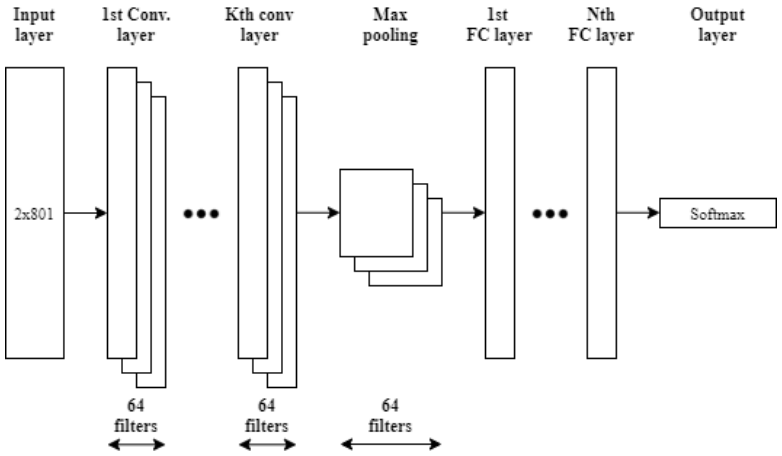


Figure 5.4: Illustration of the generic architecture for the discriminator of the DCGAN. Weights and biases are implicit.

5.2.2 Generator architecture

For the purpose of designing the generator of the DCGAN, the procedure is the same as with the discriminator. The basis of the design will be the FCGAN, and from this a generic architecture for a CNN implementation of the generator will be made.

Input layer

Similarly to the generator in the FCGAN the input to this generator is not the training data, but random noise. Just like with the input of the discriminator, the generator also takes a multidimensional input. The impact of the noise dimension on the performance of the network is not investigated for the

generator of the DCGAN either. It is therefore chosen arbitrarily to have the input noise dimension 100×100.

Output layer

Like the output layer of the discriminator, the output layer of the generator does not change either when compared to the FCGAN. It will still consist of 1602 nodes with no activation. There is though one different, which is that the output is reshaped into having the same dimensions as the input layer of the discriminator. This means that the output size of the output layer will be 2×801.

Hidden layers

For the design of number of hidden layers in the generator the FCGAN is also used as a basis. This means that for the number of hidden fully connected layers in the discriminator, there will be the N number of hidden layers chosen in the FCGAN. These will still be increasing with a factor of two for each hidden layer, where the first hidden layer will have a total of 256 nodes.

When it comes to the number of convolutional layers in the architecture, the same approach followed for designing the discriminator will be taken. This means that the number of convolutional layers is not specified, and a fixed number of filters of 64 in each convolutional layer is chosen. So just like the discriminator, the number convolutional layers will determine the number of filters in the generator. A pooling layer is not used for the generator, since all values are equally significant, when it comes to generating data similar to the training data. Therefore averages and maximums are not desirable. Leaky ReLU will be used as the activation function for all the convolutional and fully connected layers.

There is though one very important consideration to make with regards of the placement of the convolutional layers, that was not necessary for the discriminator. The discriminator designed in this project just takes the form of a standard CNN fir classification as described in Figure 3.8. In this architecture it is pretty straight forward to place the convolutional layers right after the input layer, since this is what is usually done for a CNN classifier [14]. What is though not so straight forward is where to place these in the generator. Through experiments it is seen that for the purpose of this project the best results where obtained, when the convolutional layers where before the other hidden layers. Thus the generic architecture for the CNN generator is made and can be seen in Table 5.5.

Specifications for the generic CNN generator architecture	
Layers	**Parameters**
Input layer	Input size = 100×100
K Convolutional layers	$K \cdot 64$ Filters, kernel size = 4×4, stride = 2, activation = leaky ReLU
N fully connected layers	$\sum_{n=1}^{N} 256 \cdot 2^n$ total nodes, activation = leaky ReLU
Output layer	1602 nodes
Reshape output	Output shape = 2×801

Table 5.5: Network specifications for the generic generator CNN architecture.

A graphical illustration of the generic DCGAN generator architecture can be seen in Figure 5.5.

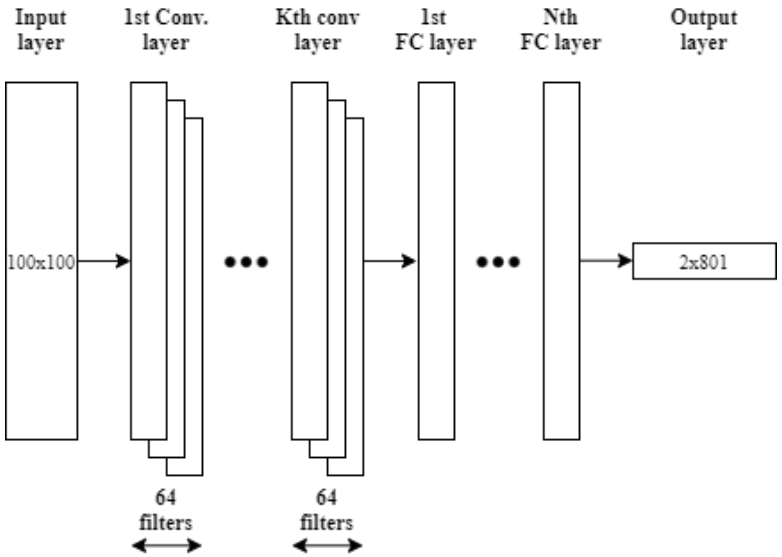


Figure 5.5: Illustration of the generic architecture for the generator of the DCGAN. Weights and biases are implicit.

5.2.3 Design of generic deep convolutional generative adversarial network

Now that both the discriminator and generator networks are designed, it is straight forward to combine these two to make up the GAN. As described in section 3.6 and as seen in Figure 5.3, the concept of a GAN is simply the process of linking the two networks.

The training process of the GAN will be conducted as explained in section 3.6. For the purpose of training and evaluating both the discriminator and the generator the loss function of the network is chosen to be binary cross entropy, which is described in section 3.4. For the choice of optimiser in the system, Adam is chosen.

These choices combined with the design of the generic architectures for both discriminator and generator yields the following generic architecture for the DCGAN, which can be seen in Table 5.6.

Specifications for the generic DCGAN architecture	
Generator layers	**Generator parameters**
Input layer	Input size = 100×100
K Convolutional layers	$K \cdot 64$ Filters, kernel size = 4×4, stride = 2
N fully connected layers	$\sum_{n=1}^{N} 256 \cdot 2^n$ total nodes, activation = leaky ReLU
Output layer	1602 nodes
Reshape output	Output shape = 2×801
Discriminator layers	**Discriminator parameters**
Input layer	Input size = 2×801
K Convolutional layers	$K \cdot 64$ Filters, kernel size = 2×10 and 1×10, stride = 2
Max pooling layer	Pool size = 1×10, stride = 2
N fully connected layers	$\sum_{n=1}^{N} 256 \cdot 2^n$ total nodes, activation = leaky ReLU
Output layer	1 node, activation = Softmax
Optimizer	**Loss function**
Adam	Binary Cross Entropy

Table 5.6: Network specifications for the generic DCGAN architecture.

Since the number of hidden layers and nodes is not of concern with regards to the scope of the project, it is not deemed necessary to fine tune these numbers. For the purpose of the project it is therefore chosen to not further investigate model parameters when satisfactory results are reached, though a more optimal choice of the number of parameters may exist.

Training and testing of GAN 6

In this chapter two different realisations of a GAN will be reviewed. Relevant choices with regards to the structure of the networks will be described. These choices include the type of input data and the pre-processing of this data, batch size of the input data, and the amount of epochs necessary for training the GAN. The number of epochs the GAN has trained will be compared to the loss and accuracy of the two networks combined in the GAN. These two realisations of the GAN will also be reviewed with regards to the statistics of the generated data as explained in subsection 4.2.1.

6.1 Training and initial testing of FCGAN

As explained in section 5.1 the first implementation of the GAN will be based on FCNNs. This means implementing an FCGAN. As earlier described the focus here will be to focus on the number of hidden layers in each of the FCNNs, where the number of neurons increase in a fixed manner as seen in Table 5.3.

6.1.1 Pre processing of input data

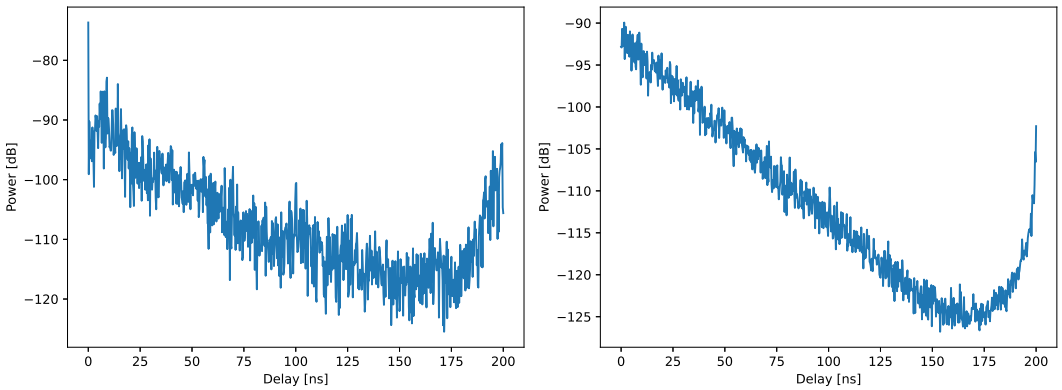
As described in section 5.1 the input size to the FCGAN is 1×1602. As described in chapter 4 normalisation of the input data is used to speed up training time and boost performance. Normalisation is applied as described in section 4.3. This training data input is generated from a frequency response of a stochastic channel model, and therefore the data is in the frequency domain. An initial investigation of the impact of transforming the training data to the time domain is made. This is transformation is applied by taking the inverse Fourier transform of the channel frequency response.

A brief investigation where only the PDP is used as reference for performance is made. In Figure 6.1 it can be seen that when using the time domain data, a much better match for the PDP of the training data is achieved.

Figure 6.1 is just an example of one of many tests that has shown, that for the purpose of this project, the performance of the FCGAN is significantly increased when using the time domain training data. This might not be true for other GAN architectures, but for the implementation of the FCGAN it is chosen to only work with time domain data.

6.1.2 Training time

As described in section 3.4 a balance between the number of epochs and the batch size is necessary for getting good performance. Through the process of training and testing the FCGAN, multiple batch sizes have been tried. Obviously increasing the batch size increases the training speed per epoch, but



(a) Average PDP of the generated data when frequency domain data is used for training.

(b) Average PDP of the generated data when time domain data is used for training.

Figure 6.1: Comparison of the PDPs of the generated data with either frequency or time domain data used for training.

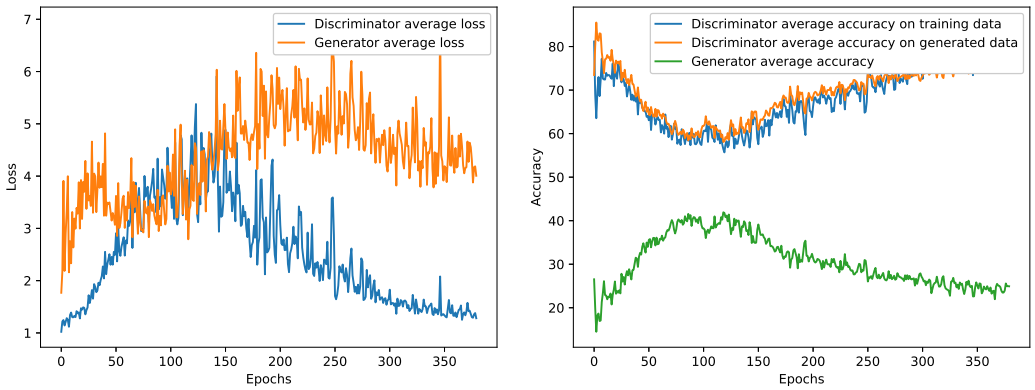
it also increases the number of epochs necessary for convergence of the FCGAN, since fewer gradient upgrades are applied each epoch.

For the purpose of training a GAN there are though some things, that vary from what is described in section 3.4. As explained the batch size describes how many samples from the training data is used before gradient updates are applied, but for GANs it is also a measure of how many samples are generated from the generator network. This means that the batch size in a GAN actually results in twice the amount of input samples for the discriminator network compared to a traditional neural network. This is because it is wanted to train the discriminator on both the fake and the real data, and then computing the loss based on this. From this the loss of the generator will be computed based on the discriminators performance on the fake data. This also means, that the discriminator will be trained on batch size·2 input samples before gradient updates are applied to it, where as the generator will only be evaluated based on the samples it generates.

For testing the affect of the choice of batch size, many different batch sizes have been assessed. This assessment is done with regards to training for an undefined number of epochs. The results are then inspected and a number of epochs is found to be the point, where no further improvement in performance was observed. For the implementation of the FCGAN it was found, that good performance was achieved with a batch size of 10 and a training time of 380 epochs.

It must be noted, that performance has not been evaluated solely on the loss function and accuracy of the FCGAN. This is due to the fact, that the information from these are not sufficient for an evaluation of the network. When looking at the training dynamics over these 380 epochs seen in Figure 6.2, it would seem that less than 380 epochs is sufficient for a good performance.

From this it can be concluded that the most times the generator network fooled the discriminator was achieved at around 100 epochs. This means that if the loss function and accuracy of the system was the only performance measure, the training should be stopped at around a 100 epochs. If on the other hand the statistics of the generated data is compared to the training data, it is seen that the performance is not good until around 380 epochs.



(a) Average losses of the discriminator and generator over epochs.

(b) Average accuracies of the discriminator on the training - and generated data over epochs and success rate of the generator over epochs.

Figure 6.2: Training dynamics of the FCGAN over a training time of 380 epochs with a batch size of 10.

This might be due to the discriminator not being sufficiently trained in the early training process, and therefore the generator can more easily fool it and thus getting a good performance for the accuracy and loss. Where as in the later stages of training both the discriminator and generator network are well trained and thus the improvements made here are more significant with regards to generating data more statistically similar to the training data. It should though be noted, that the loss and accuracy of the network still yields some insights to the training process for the later stages of training as well. Since the further along in the training process the less these change.

6.1.3 Final model for the FCGAN

Through the training and testing of different realisations of the FCGAN it was found, that good performance is achieved when each of the networks making up the GAN consist of three hidden layers. This model was trained and implemented with choices and findings described regarding the input layer and training time. From this a final network model for the FCGAN was obtained. A description of the model is seen in Table 6.1.

Specifications for the final model for the FCGAN		
Generator layers	**Generator filters**	**Number of parameters**
Input layer	0 filters, input size 1×100	0 parameters
Fully connected layer 1	256 filters	25856 parameters
Leaky ReLU 1	256 filters	0 parameters
Fully connected layer 2	512 filters	131584 parameters
Leaky ReLU 2	512 filters	0 parameters
Fully connected layer 3	1024 filters	525312 parameters
Leaky ReLU 3	1024 filters	0 parameters
Output layer	1602 filters	1642050 parameters
Discriminator layers	**Discriminator filters**	**Number of parameters**
Input layer	0 filters, input size 1×1602	0 parameters
Fully connected layer 1	1024 filters	1641472 parameters
Leaky ReLU 1	1024 filters	0 parameters
Fully connected layer 2	512 filters	524800 parameters
Leaky ReLU 2	512 filters	0 parameters
Fully connected layer 3	256 filters	131328 parameters
Leaky ReLU 3	256 filters	0 parameters
output layer	1 filters	257 parameters
Softmax activation function	1 filters	0 parameters
Total number of parameters		
The total number of parameters in the networks is 4622659		

Table 6.1: Summary of the layers, filters and parameters in each of the networks that the FCGAN consists of.

6.2 Training and initial testing of DCGAN

As explained in subsection 5.2.3 the DCGAN will be based on the design of the FCGAN. This means the that number of hidden fully connected layers in the DCGAN is already established. For the purpose of testing the

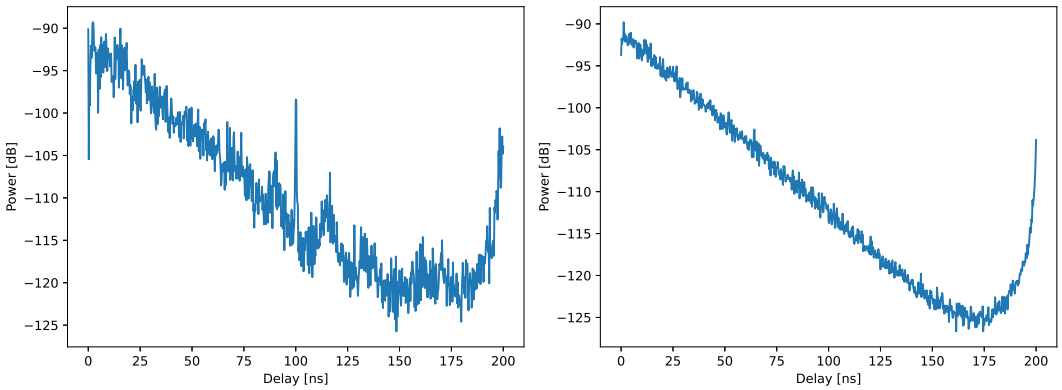
the first implementation of the GAN will be based on FCNNs. This means implementing an FCGAN. As earlier described the focus here will be to focus on the number of hidden layers in each of the FCNNs, where the number of neurons increase in a fixed manner as seen in Table 5.3.

6.2.1 Pre processing of input data

As described in section 5.2 the input size to the DCGAN is 2×801. As described in chapter 4 normalisation of the input data is used to speed up training time and boost performance. Normalisation is applied as described in section 4.3. This training data input is generated from a

frequency response of a stochastic channel model, and therefore the data is in the frequency domain. An initial investigation of the impact of transforming the training data to the time domain is made.

A brief investigation where only the PDP is used as reference for performance is made. In Figure 6.3 it can be seen that using the time domain data, yields a better match for the PDP of the training data is achieved. It can though be seen, that the increase in performance is not as significant as it was with the FCGAN.



(a) Average PDP of the generated data when frequency domain data is used for training.

(b) Average PDP of the generated data when time domain data is used for training.

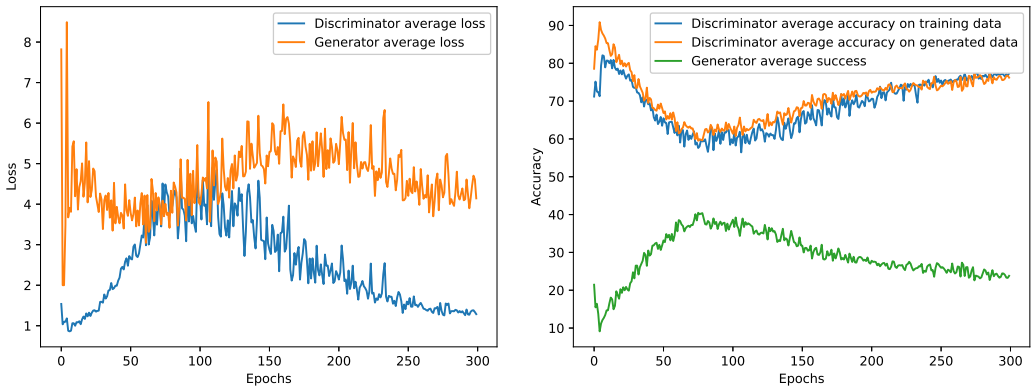
Figure 6.3: Comparison of the PDPs of the generated data with either frequency or time domain data used for training.

The figure is just an example of one of many test that shows the tendency, that the performance of the DCGAN is increased when the time domain training data is used. Therefore it is chosen from this point on to only work with the time domain data for the DCGAN as well.

6.2.2 Training time

Similarly to the FCGAN a balance between the number of epochs and the batch size is necessary for achieving good performance. The same test procedure for arriving at an ideal number of epochs and batch size described in subsection 6.1.2 is also applied to the DCGAN.

Through this test it has been seen, that the DCGAN shows the same training dynamics as the FCGAN. What is different between the two models is, that the DCGAN reaches convergence at around 300 epochs which is 80 epochs earlier than the FCGAN. A batch size of 10 was also here found to yield the best performance. The training dynamics of the DCGAN can be seen in Figure 6.4.



(a) Average losses of the discriminator and generator over epochs.

(b) Average accuracies of the discriminator on the training - and generated data over epochs and success rate of the generator over epochs.

Figure 6.4: Training dynamics of the DCGAN over a training time of 300 epochs with a batch size of 10.

From these training dynamics the same conclusion arrived at for the training dynamics of the FCGAN can be made. The loss and accuracy tend to show good performance of the network at around 100 epochs, but good results with regards to mimicking the statistics of the training data are not obtained until around 300 epochs.

6.2.3 Final model for the DCGAN

Through the training and testing of different realisations of the DCGAN it was found, that good performance is achieved when two convolutional layers where added for both the generator and the discriminator to the model already established in subsection 6.1.3. Addtionally a max pooling layer is introduced in the discriminator as described in subsection 5.1.3. This model was trained and implemented with choices and findings described regarding the input layer and training time. From this a final network model for the DCGAN was obtained. A description of the model is seen in Table 6.2.

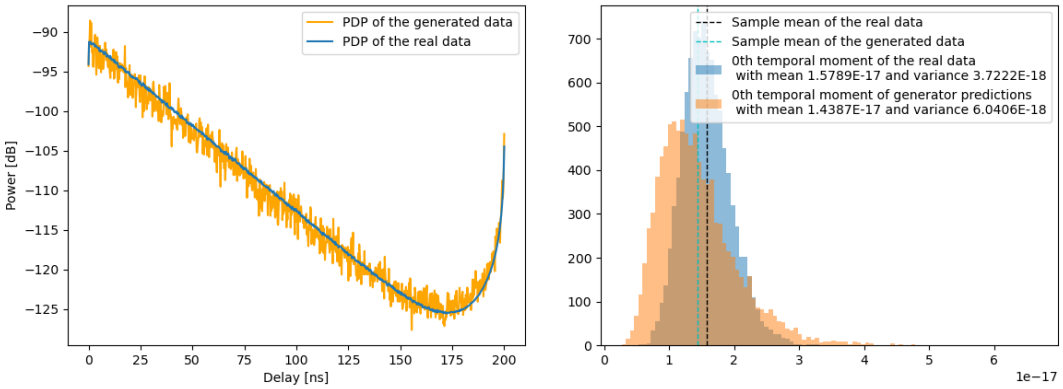
Specifications for the final model for the DCGAN		
Generator layers	**Generator filters**	**Number of parameters**
Input layer	0 filters, input size 100×100	0 parameters
Convolutional layer 1	64 filters	1088 parameters, kernel size 4×4, stride 2
Leaky ReLU 1	64 filters	0 parameters
Convolutional layer 2	64 filters	65600 parameters, kernel size 4×4, stride 2
Leaky ReLU 2	64 filters	0 parameters
Fully connected layer 1	256 filters	8667392 parameters
Leaky ReLU 3	256 filters	0 parameters
Fully connected layer 2	512 filters	131584 parameters
Leaky ReLU 4	512 filters	0 parameters
Fully connected layer 3	1024 filters	525312 parameters
Leaky ReLU 5	1024 filters	0 parameters
Output layer	1602 filters	1642050 parameters
Reshape output	0 filters, output shape 2×801	0 parameters
Discriminator layers	**Discriminator filters**	**Number of parameters**
Input layer	0 filters, input size 2×801	0 parameters
Convolutional layer 1	64 filters	1344 parameters, kernel size 2×10, stride 2
Leaky ReLU 1	64 filters	0 parameters
Convolutional layer 2	64 filters	41024 parameters, kernel size 1×10, stride 2
Leaky ReLU 2	64 filters	0 parameters
Max pooling layer 1	64 filters	0 parameters, pool size 1×10, stride 2
Fully connected layer 1	1024 filters	6095872 parameters
Leaky ReLU 3	1024 filters	0 parameters
Fully connected layer 2	512 filters	524800 parameters
Leaky ReLU 4	512 filters	0 parameters
Fully connected layer 3	256 filters	131328 parameters
Leaky ReLU 5	256 filters	0 parameters
Output layer	1 filters	257 parameters
Softmax activation function	1 filters	0 parameters
Total number of parameters		
The total number of parameters in the networks is 17827651		

Table 6.2: Summary of the layers, filters and parameters in each of the networks that the DCGAN consists of.

6.3 Comparison of GAN models

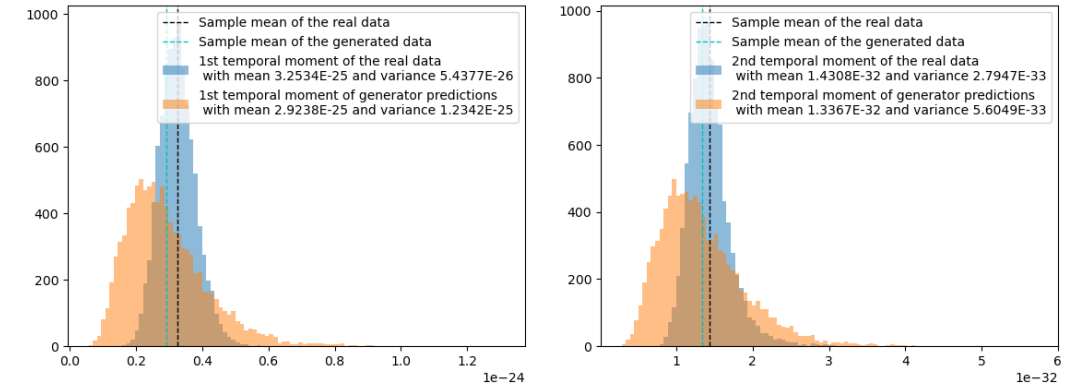
For a final evaluation of the GANs, the two are compared with regards to performance. As stated in subsection 4.2.1 the statistical properties of the training data set will serve as evaluation of the GANs. For this purpose a visual inspection of the data generated from the FCGAN and the DCGAN is conducted with regards to how similar the generated data is to the training data.

The data generated from FCGAN compared to the training data can be seen in Figure 6.5.



(a) Comparison of the average power delay profile of the training- and generated data.

(b) Comparison of the 0th temporal moments of the training- and generated data.



(c) Comparison of the 1st temporal moments of the training- and generated data.

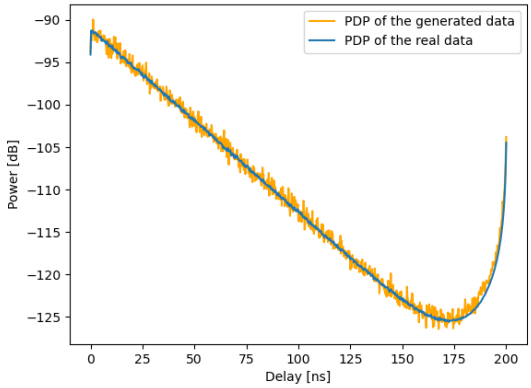
(d) Comparison of the 2nd temporal moments of the training- and generated data.

Figure 6.5: Overview of the comparison of statistics between the training data and the generated data for the FCGAN

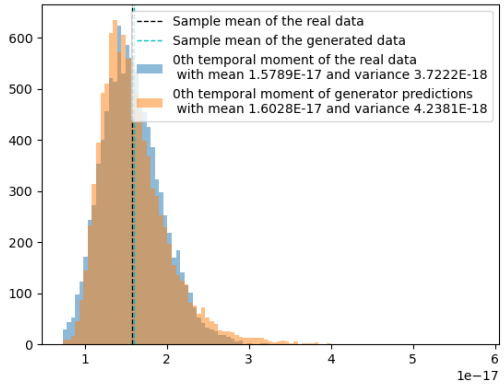
From this it can be seen, that the FCGAN gets a decent match of the training with regards to the PDP and the means of the temporal moments. It can though be seen, that there is a big difference between the variance of the temporal moments of the data generated from the FCGAN and the variance of the temporal moments of the real training data. It can also be seen, that the best match is made for the 0th temporal moment.

For the DCGAN the data can be seen in Figure 6.6

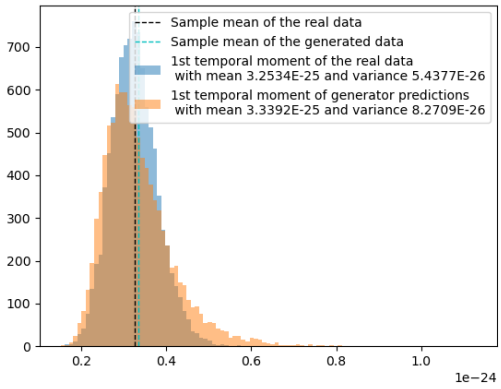
Comparing the data from the DCGAN to the real training data, it can be seen that a much better



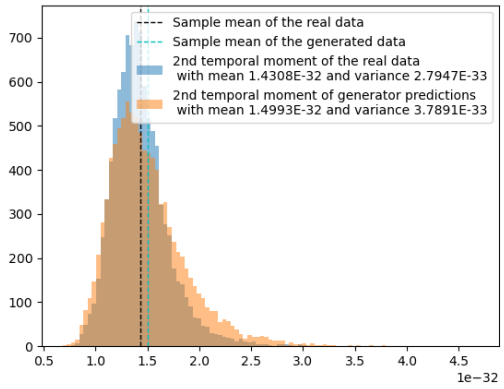
(a) Comparison of the average power delay profile of the training- and generated data.



(b) Comparison of the 0th temporal moments of the training- and generated data.



(c) Comparison of the 1st temporal moments of the training- and generated data.



(d) Comparison of the 2nd temporal moments of the training- and generated data.

Figure 6.6: Overview of the comparison of statistics between the training data and the generated data for the DCGAN

match for both PDP and all the temporal moments is achieved compared to the FCGAN. Good results for both the mean and variance of the temporal moments are achieved, but similarly to the FCGAN the best match is made for the 0th temporal moment. It should though be noted, that whereas the matching of the means of the temporal moments are very good, there is still not as good of a match for the variances. From these results it can be seen, that the best performance was obtained with the DCGAN. It is therefore chosen to not test the FCGAN further, where as further evaluation of the DCGAN will be made. It is also seen, that values for the mean and variance of the temporal moments is sufficient for comparing the GAN generated data to the training data.

6.4 Robustness of network

In this section the model of the DCGAN, which was found in section 6.3, will be tested with regards to robustness of the architecture. For the purpose of this test, two things will be changed compared to the original test.

Firstly the network architecture will be tested on the impact of adding noise to the training data. Secondly a test of the network where a limited amount of the data set is available for training. These two cases are considered to test the architecture's capabilities of generating data based on training sets that are more similar to that of real life measurements where fewer training samples are available and measurements have noise added to them.

6.4.1 Impact of noise on the model

In order to evaluate the robustness of the model with regards to noise, nine new data sets have been generated. These data sets have been generated as described in chapter 4. Noise has then been added to each measurement in the data set in the form of circular symmetric Gaussian noise. The first data set's noise has a variance of $\sigma^2 = 2.8 \cdot 10^{-10}$. This level of noise is chosen based on an estimate of noise in some real indoor channel response measurements [19]. This level of noise results in a noise floor in the PDP of the channel response at approximately $\frac{\sigma^2}{N_s} = -124.5$ dB, where N_s is the number of measurements in each channel realisation, which in this case is 801. For the noise added to the remaining data sets, it is chosen to double the variance of the added noise each time, resulting in adding 3 dB to the noise floor for each new data set. This results in the PDPs of the training data which are seen in Figure 6.7.

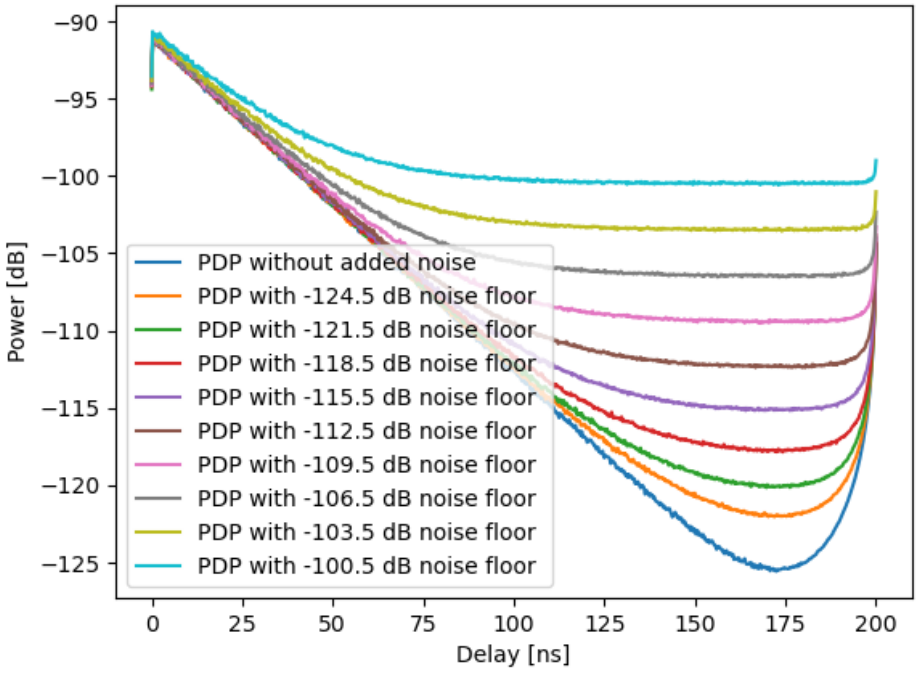


Figure 6.7: Average PDP of 10,000 channel realisations for nine training data sets with different noise floor.

From these data sets the temporal moments have also been computed and these will again be used as an evaluation statistic. As stated in section 6.3 an evaluation of the match of the temporal moments can be made by only considering the mean and variance of these. It has been seen through experiments, that the better the PDPs match, the better the temporal moments match. This is obvious, since the temporal moments are computed based on the PDPs. Because of this it is deemed sufficient for the purpose of this test to only compare the means and variances of the temporal moments.

6.4.2 Results with noisy data

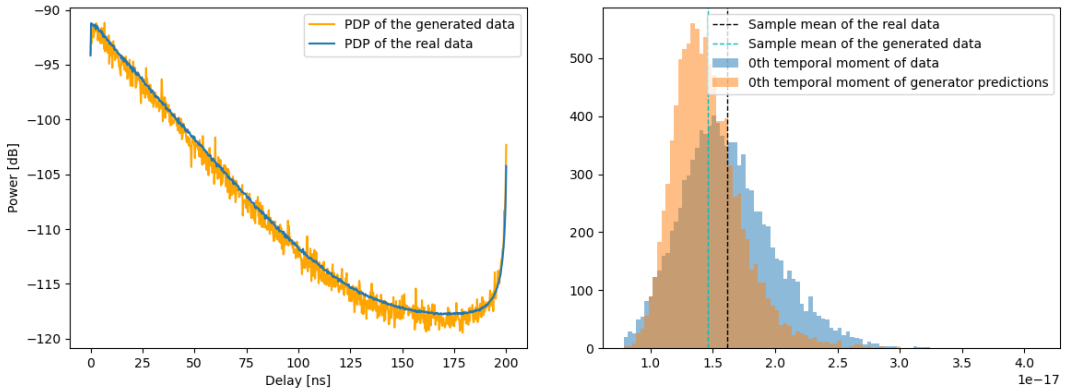
through the training process it has been seen that adding noise to the training data does affect performance of the network. The results are though still very promising and even for higher levels of added noise it can be seen, that the generator network does in fact performs decently with regards to generating data with similar temporal moments as the training data. As earlier stated it is possible to compare the temporal moments using only their mean and variance. Since many different results are being compared for this section, it is therefore chosen to compare the temporal moments based on this and not the visual/graphical inspection formerly used. A table containing the comparisons of means and variances of the noisy data sets and the training data can be seen in Table 6.3

Comparison of the GAN generated data and the training based on means and variances of the temporal moments computed from the data sets		
Approx. noise floor of data	**Means for temporal moments compared (generated $\hat{\mu}$, training μ)**	**variances for temporal moments compared (generated $\hat{\sigma^2}$, training σ^2)**
-124.5 dB	$\hat{\mu}_{m0} = 1.6687 \cdot 10^{-17}$, $\mu_{m0} = 1.5867 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 3.4908 \cdot 10^{-25}$, $\mu_{m1} = 3.3215 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 1.6257 \cdot 10^{-32}$, $\mu_{m2} = 1.5227 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 3.3708 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7783 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 6.7733 \cdot 10^{-26}$, $\sigma^2_{m1} = 5.5846 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 3.3115 \cdot 10^{-33}$, $\sigma^2_{m2} = 2.8960 \cdot 10^{-33}$
-121.5 dB	$\hat{\mu}_{m0} = 1.5527 \cdot 10^{-17}$, $\mu_{m0} = 1.5983 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 3.4101 \cdot 10^{-25}$, $\mu_{m1} = 3.3950 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 1.6901 \cdot 10^{-32}$, $\mu_{m2} = 1.6167 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 3.3771 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.8105 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 7.0165 \cdot 10^{-26}$, $\sigma^2_{m1} = 5.5534 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 3.6466 \cdot 10^{-33}$, $\sigma^2_{m2} = 2.8699 \cdot 10^{-33}$
-118.5 dB	$\hat{\mu}_{m0} = 1.4624 \cdot 10^{-17}$, $\mu_{m0} = 1.6098 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 3.2422 \cdot 10^{-25}$, $\mu_{m1} = 3.5293 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 1.7067 \cdot 10^{-32}$, $\mu_{m2} = 1.8001 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 2.8515 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7249 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 6.0661 \cdot 10^{-26}$, $\sigma^2_{m1} = 5.5025 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 3.3549 \cdot 10^{-33}$, $\sigma^2_{m2} = 2.8824 \cdot 10^{-33}$
-115.5 dB	$\hat{\mu}_{m0} = 1.5890 \cdot 10^{-17}$, $\mu_{m0} = 1.6315 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 3.7755 \cdot 10^{-25}$, $\mu_{m1} = 3.8119 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 2.1926 \cdot 10^{-32}$, $\mu_{m2} = 2.1813 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 3.3305 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7251 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 7.7143 \cdot 10^{-26}$, $\sigma^2_{m1} = 5.4484 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 4.5784 \cdot 10^{-33}$, $\sigma^2_{m2} = 2.9588 \cdot 10^{-33}$
-112.5 dB	$\hat{\mu}_{m0} = 1.4485 \cdot 10^{-17}$, $\mu_{m0} = 1.6853 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 3.7880 \cdot 10^{-25}$, $\mu_{m1} = 4.3621 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 2.5808 \cdot 10^{-32}$, $\mu_{m2} = 2.9232 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 3.1959 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7360 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 8.1704 \cdot 10^{-26}$, $\sigma^2_{m1} = 5.5081 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 5.6855 \cdot 10^{-33}$, $\sigma^2_{m2} = 3.0025 \cdot 10^{-33}$
-109.5 dB	$\hat{\mu}_{m0} = 1.6077 \cdot 10^{-17}$, $\mu_{m0} = 1.8085 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 4.8858 \cdot 10^{-25}$, $\mu_{m1} = 5.4916 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 3.9737 \cdot 10^{-32}$, $\mu_{m2} = 4.4205 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 3.1919 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7740 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 9.5118 \cdot 10^{-26}$, $\sigma^2_{m1} = 5.6703 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 7.8369 \cdot 10^{-33}$, $\sigma^2_{m2} = 3.2809 \cdot 10^{-33}$
-106.5 dB	$\hat{\mu}_{m0} = 1.9368 \cdot 10^{-17}$, $\mu_{m0} = 2.0318 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 7.4891 \cdot 10^{-25}$, $\mu_{m1} = 7.7424 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 7.2549 \cdot 10^{-32}$, $\mu_{m2} = 7.4251 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 3.2938 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7956 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 1.2081 \cdot 10^{-25}$, $\sigma^2_{m1} = 5.9643 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 1.1839 \cdot 10^{-32}$, $\sigma^2_{m2} = 4.2161 \cdot 10^{-33}$
-103.5 dB	$\hat{\mu}_{m0} = 2.4279 \cdot 10^{-17}$, $\mu_{m0} = 2.4769 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 1.2124 \cdot 10^{-24}$, $\mu_{m1} = 1.2227 \cdot 10^{-24}$ $\hat{\mu}_{m2} = 1.3309 \cdot 10^{-31}$, $\mu_{m2} = 1.3416 \cdot 10^{-31}$	$\hat{\sigma^2}_{m0} = 4.3638 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7860 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 2.1329 \cdot 10^{-25}$, $\sigma^2_{m1} = 6.8443 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 2.3834 \cdot 10^{-33}$, $\sigma^2_{m2} = 6.5625 \cdot 10^{-33}$
-100.5 dB	$\hat{\mu}_{m0} = 2.6695 \cdot 10^{-17}$, $\mu_{m0} = 3.3694 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 1.6847 \cdot 10^{-24}$, $\mu_{m1} = 2.1182 \cdot 10^{-24}$ $\hat{\mu}_{m2} = 2.0188 \cdot 10^{-31}$, $\mu_{m2} = 2.5384 \cdot 10^{-31}$	$\hat{\sigma^2}_{m0} = 5.5822 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.8968 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 3.4503 \cdot 10^{-25}$, $\sigma^2_{m1} = 9.5831 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 4.1706 \cdot 10^{-32}$, $\sigma^2_{m2} = 1.2005 \cdot 10^{-32}$

Table 6.3: Comparison of the GAN generated data and the training based on means and variances of the temporal moments computed from the data.

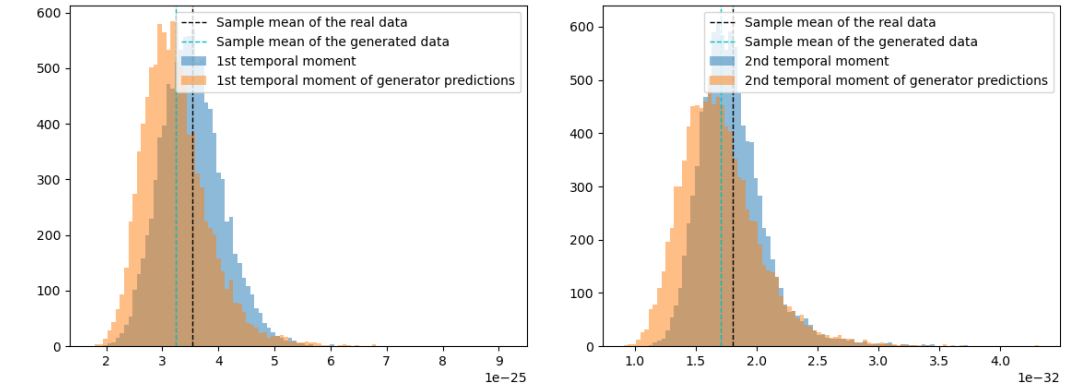
It is obvious from this test, that every time a higher level of noise is added to the training data, a decrease in the performance is observed. It can though also be seen, that the matching of the means of the temporal moments is relatively good even for the highest levels of added noise. This is though not the case for the matching of the variances. It is clear that a more noisy training data set results in the variance of the temporal moments computed from the generated data having a large numerical difference compared to those of the training data. What can also be seen from the results is, that the higher the order of the temporal moments the more fragile the matching is with regards to the added noise. The matching of the second temporal moment starts to have very poor performance when the training data has a noise floor of -112.5dB or higher. The matching of the second temporal moment does not show that much of a drop in performance before the training data has a noise floor of -103.5dB or higher. Finally the matching of the zeroth temporal moments does not show a significant drop in performance before the training data has a noise floor of -100.5dB.

An illustration of the statistics of one of the generated data sets compared to the training data can be seen in Figure 6.8. The GAN used for generating these have been trained using a training data with a noise floor of 118.5dB.



(a) Comparison of the average power delay profile of the training- and generated data.

(b) Comparison of the 0th temporal moments of the training- and generated data.



(c) Comparison of the 1st temporal moments of the training- and generated data.

(d) Comparison of the 2nd temporal moments of the training- and generated data.

Figure 6.8: Overview of the comparison of statistics between the training data with a noise floor of -118.5 dB and the generated data

It is quite obvious that the matching between the temporal moments is not as good as the ones with the noise less data set. This might be due to the added noise, but it also might be due to the fact that the specific architecture might not be suited for noisy data. It should also be noted that optimal training time and batch size have not been investigated. The parameters found through the initial testing of the network have been deemed sufficient to investigate the impact of noise, though these might not be optimal with regards to the performance measures. The tests do though still provide insights to the amount of noise that can be added to the measurements before performance starts to decrease dramatically.

6.4.3 Results with limited data

In order to evaluate the robustness of the model with regards to limiting the amount of training data, four different sizes of the training data set is reviewed. The size of the data set refers to the number of realisations of the channel response included in this. The four sizes reviewed are, 5000, 2500, 1250 and 625. These have been arbitrarily to just be half of the previous data set.

In contrast to the test of the models robustness to noise, the GAN trained with the limited training data sets will be evaluated based on its ability to generate data similarly to the full training data set of size 10000. This is done, since it is desired to obtain knowledge of the amount of training data necessary to generate channel responses that are statistically similar to the full data set.

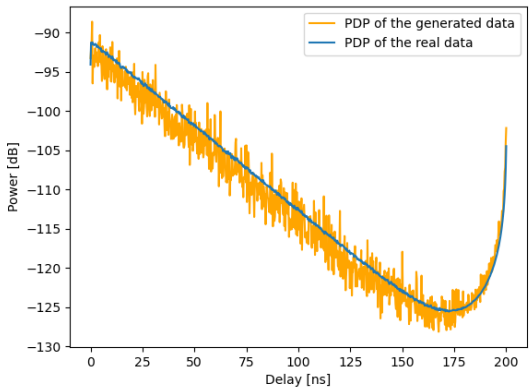
The DCGAN model has been trained with the four different data set sizes. From the trained models data has been generated and based on this the temporal moments and the means and variances of these have been computed. A table containing the comparisons of means and variances of the limited data sets and the training data can be seen in Table 6.4.

Comparison of the GAN generated data and the training based on means and variances of the temporal moments computed from the data sets		
Data set size	**Means for temporal moments compared (generated $\hat{\mu}$, training μ)**	**variances for temporal moments compared (generated $\hat{\sigma^2}$, training σ^2)**
5000	$\hat{\mu}_{m0} = 1.4477 \cdot 10^{-17}$, $\mu_{m0} = 1.5789 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 3.0442 \cdot 10^{-25}$, $\mu_{m1} = 3.2534 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 1.4221 \cdot 10^{-32}$, $\mu_{m2} = 1.4308 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 2.8983 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7222 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 5.7240 \cdot 10^{-26}$, $\sigma^2_{m1} = 5.4377 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 2.8974 \cdot 10^{-33}$, $\sigma^2_{m2} = 2.7947 \cdot 10^{-33}$
2500	$\hat{\mu}_{m0} = 1.5890 \cdot 10^{-17}$, $\mu_{m0} = 1.5789 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 3.7755 \cdot 10^{-25}$, $\mu_{m1} = 3.2534 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 2.1926 \cdot 10^{-32}$, $\mu_{m2} = 1.4308 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 3.3305 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7222 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 7.7143 \cdot 10^{-26}$, $\sigma^2_{m1} = 5.4377 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 4.5784 \cdot 10^{-33}$, $\sigma^2_{m2} = 2.7947 \cdot 10^{-33}$
1250	$\hat{\mu}_{m0} = 1.2408 \cdot 10^{-17}$, $\mu_{m0} = 1.5789 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 2.6088 \cdot 10^{-25}$, $\mu_{m1} = 3.2534 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 1.2585 \cdot 10^{-32}$, $\mu_{m2} = 1.4308 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 9.7702 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7222 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 2.0462 \cdot 10^{-25}$, $\sigma^2_{m1} = 5.4377 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 1.0073 \cdot 10^{-32}$, $\sigma^2_{m2} = 2.7947 \cdot 10^{-33}$
625	$\hat{\mu}_{m0} = 1.2449 \cdot 10^{-17}$, $\mu_{m0} = 1.5789 \cdot 10^{-17}$ $\hat{\mu}_{m1} = 2.5770 \cdot 10^{-25}$, $\mu_{m1} = 3.2534 \cdot 10^{-25}$ $\hat{\mu}_{m2} = 1.1510 \cdot 10^{-32}$, $\mu_{m2} = 1.4308 \cdot 10^{-32}$	$\hat{\sigma^2}_{m0} = 7.5124 \cdot 10^{-18}$, $\sigma^2_{m0} = 3.7222 \cdot 10^{-18}$ $\hat{\sigma^2}_{m1} = 1.5163 \cdot 10^{-25}$, $\sigma^2_{m1} = 5.4377 \cdot 10^{-26}$ $\hat{\sigma^2}_{m2} = 7.1174 \cdot 10^{-33}$, $\sigma^2_{m2} = 2.7947 \cdot 10^{-33}$

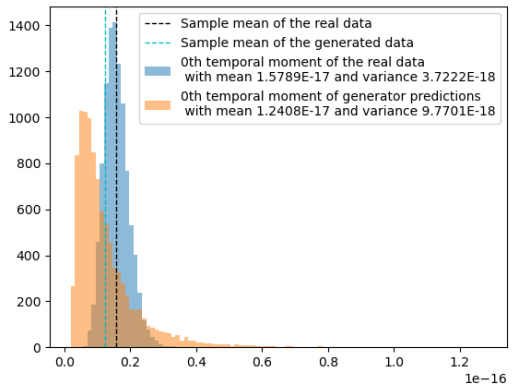
Table 6.4: Comparison of the GAN generated data and the training based on means and variances of the temporal moments computed from the data.

From this it can be seen, that a good match with regards to the temporal moments is maintained

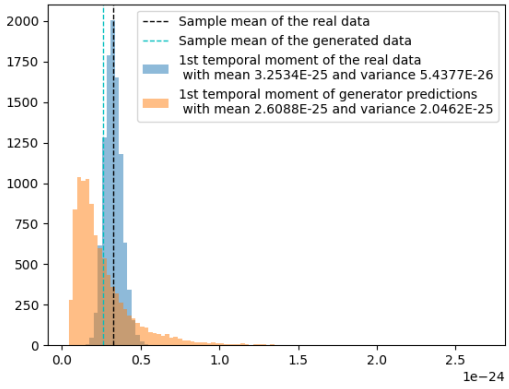
when the data set size is 2500. At this size it is though seen, that a difference in the variance for the second temporal moment is already apparent. For the data set sizes below 2500 it can be seen that the matching of the variances starts to become worse. Similarly to the noisy data sets it is though seen, that the matching of the means is pretty consistent for all the data set sizes. Similarly to the tests with the noisy data set, it is seen that the higher the order of the temporal moment, the quicker they are to lose their matching. It does though seem that reducing the size of the training set has a bigger impact on the performance of the GAN than adding noise to the training data. It can though not be concluded whether another architecture for the GAN could yield better results, and whether or not a change of training time and batch size could increase performance has not been investigated. An illustration of the statistics of one of the generated data sets compared to the training data can be seen in Figure 6.9. The GAN used for generating these have been trained using a training data set of size 1250.



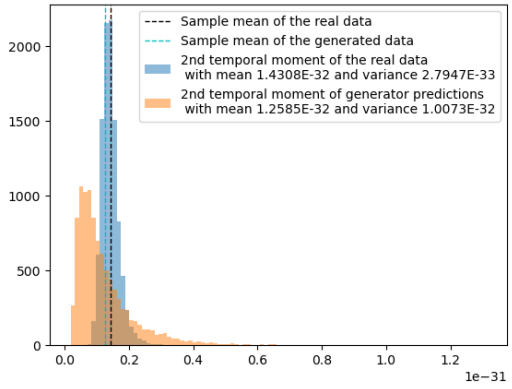
(a) Comparison of the average power delay profile of the training- and generated data.



(b) Comparison of the 0th temporal moments of the training- and generated data.



(c) Comparison of the 1st temporal moments of the training- and generated data.



(d) Comparison of the 2nd temporal moments of the training- and generated data.

Figure 6.9: Overview of the comparison of statistics between the training data limited to a size of 1250 and the generated data

Just like with the test for the noisy data sets, the parameters found through the initial testing of the network have been deemed sufficient to investigate the impact of limiting the data set, though these might not be optimal with regards to the performance measures. The tests do though still provide insights to the affect of limiting the amount of training data available for the GAN.

Evaluation with measured channel data

7

In this chapter further testing of the chosen DCGAN model will be conducted. For this purpose a data set consisting of real channel response measurements done at Lund University in Sweden has been provided.

7.1 Data set description

The data set used is the one described in [21]. This data set consists of channel transfer functions that have been measured using a vector network analyser. The measurements have been made in a room with the dimensions, $3m \times 4m \times 3m$. The measurements are performed in SISO mode using a planar array of 25×25. This results in a total of 625 realisations of the channel. The bandwidth of the measured signal is 4 GHz with a number of measurements per realisation, $N_s = 801$. This means 801 samples in each channel measurement, which results in a frequency spread of $\Delta f = 5$ MHz and a maximum delay, $t_{max} = 200$ ns [21]. The delay of the first peak is found visually through inspection of the data, and is found to be $t_0 = 6$ ns [19].

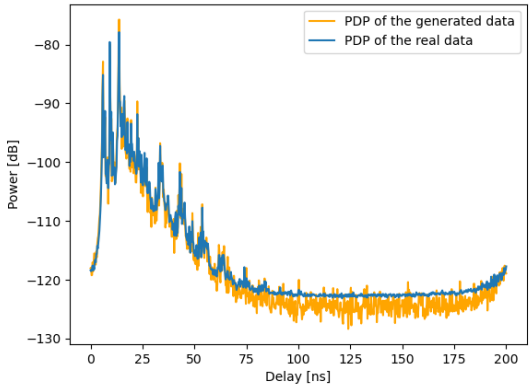
7.2 Evaluation of DCGAN on the Lund data set

For the purpose of testing the DCGAN architecture on the Lund data set, the PDP and the temporal moments of the data set are computed as described in subsection 4.2.1. These will then be compared to the GAN generated data.

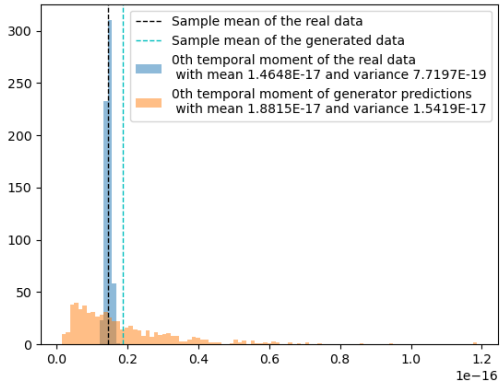
as described in section 6.4 both the amount of training examples available and the amount of noise in the measurements have an impact on the performance. Though the noise level in the Lund data set is fairly low the number of channel realisations are very limited compared to the size of the synthetic training data set required for achieving good performance of the GAN.

Through training the DCGAN using the Lund training data, it is seen that as stated above, that either the noise level in the measurements or the size of the data set has an impact of the performance. The PDP and the temporal moments of the data generated from the GAN have been computed, and these are then compared to the Lund data set. The comparison of the statistics can be seen in Figure 7.1.

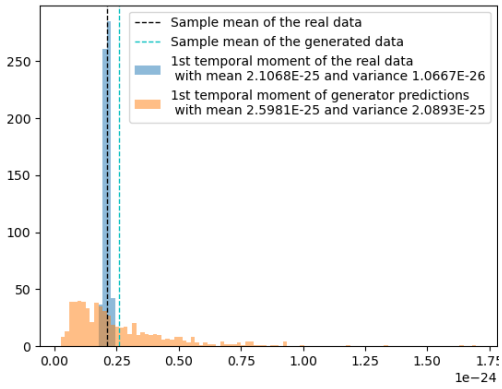
From these results it is fairly sure that the data set is too limited, since the GAN generated data seems to exhibit the same tendencies as what was seen in subsection 6.4.3. Though the results are not very good, there is still some promising aspects to the GAN generated data. When looking at the PDP it can be seen, that the GAN does in fact capture some of the features of the training data. Both some of the peaks and the exponentially decaying behaviour of the PDP are apparent. When expecting the temporal moments, it can also be seen, that though the variances differ a lot, the means are still fairly



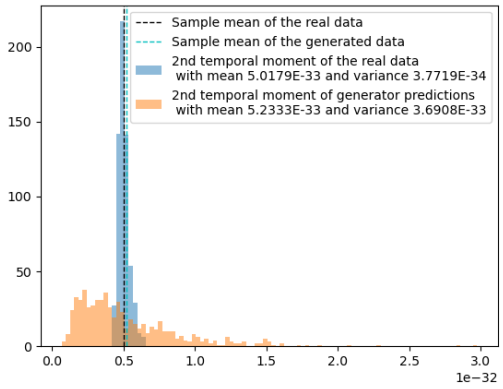
(a) Comparison of the average power delay profile of the training- and generated data.



(b) Comparison of the 0th temporal moments of the training- and generated data.



(c) Comparison of the 1st temporal moments of the training- and generated data.



(d) Comparison of the 2nd temporal moments of the training- and generated data.

Figure 7.1: Overview of the comparison of statistics between the Lund training data and the generated data.

similar. Overall it does seem that it might be possible to obtain better performance if a larger data set was available.

Conclusions 8

In this chapter, conclusions with regards to the problem statement and the additional research questions associated to it will be made. Additionally a discussion regarding future works based on the results in this project is made.

Throughout this report different realisations of a GAN have been implemented, trained and evaluated based on performance measures established in the report. The scope of this project has been to study the feasibility of using GANs as a means of generating stochastic channel responses. It has been sought to establish a fixed point of evaluation for the performance of the GAN, and through this evaluation it has been possible to improve the performance of the GAN, both in a heuristic experimental manner, but also through relevant choices made based on the theory associated with neural networks and specifically GANs.

In this project it has been seen that many aspects of the training process of the GAN have a big influence of the final results and performance. On such effect was observed in the choice of pre-processing the training data with regards to using the time domain or frequency domain data in the training of the GAN. It was seen how both the training time of the networks but also the performance were greatly impacted by this. A brief investigation quickly determined that using the time domain training data was far superior with regards to performance for the scope of this project. This fact was established for all the architectures reviewed.

Another aspect that has shown a big impact on the performance was the choices made regarding training time, specifically the choice of number of epochs and batch size. Though many different constellations of the two may yield the same amount of gradient updates throughout the training run, it was seen that widely different results were achieved. It has been seen that for the specific architectures a balance between the number of epochs and the batch size is important for reaching good performance. Specifically it was shown, that for the purpose of the synthetic data set used in the training process, that a batch size of 10 was ideal for both the implementation of the FCGAN and the DCGAN. It was though also seen, that a decrease in the number of epochs trained was a achieved for the implementation of the DCGAN. The initial FCGAN implementation needed 380 epochs of training before convergence, whereas the DCGAN cut this down to 300 epochs.

When evaluating these two networks based on the statistical properties of the data generated from these compared to those from the training data, it was clear that the DCGAN reached better results for all performance measures. As stated the convergence time for the DCGAN was faster, but it also showed much better results for mimicking the statistics of the training data. When evaluating these statistics of the DCGAN generated data it is apparent that a complete match is not obtained. The results are though seen as being sufficiently matching for the scope of the project.

As stated before the pre-processing of the training data has shown to have a big impact on the

performance of the GAN. This is though not the only impact of the training data. With regards to robustness of the network model, the impact of noise and limited training data has been reviewed. From this it has be seen that both of these measures have a big impact on the performance of the network. When adding noise or limiting the amount of training a drop in performance is immediately observed, and for large amount of noise and very limited data training data the performance increasingly worse. It is though seen from these experiments that despite all this the model still replicates some of the desired statistics from the data though not all of them. Through the investigation of the robustness of the model, it can also be included that for the two metrics of robustness, limiting the training data has the biggest impact on the performance of the model.

After the robustness of the model has been evaluated, the model is also evaluated on a data set consisting of real measurements performed in an indoor environment. For this case the model was limited with regards to both the size of the data set and the fact that the measurements in this data set contained noise. When using the real measurements as training data it was seen, that the model did not have much success with regards to generating statistically similar data. Though the noise in the measurements might also affect the performance it is believed that the biggest factor is the limited training data available.

When considering all the results obtained throughout this report it seems that GAN have potential for generating channel responses from a stochastic radio channel model. This is though mostly based on the results seen from the synthetic channel response data. The Lund data set was deemed too small to be able to properly train the GAN, but even this did show some promising results. Further development of the models implemented in this report with regards to tuning of parameters might yield better performance, and other architectures may prove more ideal for the purpose of this project, but from a feasibility point of view it can though still be concluded, that the GAN concept does show promising results with regards to generating radio channel responses.

future works

The purpose of this project has been, as earlier stated, to study the feasibility of using GANs for generating radio channel responses. The study has though been limited to only consider one type of channel, which is that of the Turin model and data similar to this. The study has also furthermore only considered time invariant channels. Additional investigations into whether or not the results would generalise to other types of channel models as well will therefore be an obvious next step. This would be the case for both other time-invariant channels but also for time-varying channels. Another aspect of the channel models considered in the project is that only SISO has been investigated. Therefore further expanding models to a MIMO setup should be considered. Finally another point of investigation would be to test how a system behaves over GAN generated channel responses compared to model generated channels, measured channels or actual operation in the field. From this the matching of the system performance could be investigated when evaluating the GAN generated channel responses.

Bibliography

[1] Troels Pedersen. *Contributions in Radio Channel Sounding, Modeling, and Estimation.* PhD thesis, Aalborg University, 2009.

[2] Morten Lomholt Jakobsen. *Modeling and Analysis of Stochastic Radio Channels: An Application of the Theory of Spatial Point Processes.* PhD thesis, Aalborg University, 2013.

[3] Michal Uřičář, Pavel Křížek, David Hurych, Ibrahim Sobh, Senthil Yogamani, and Patrick Denny. Yes, we GAN: Applying adversarial techniques for autonomous driving. *Electronic Imaging*, 2019(15):48–1–48–17, January 2019.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[5] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. 10 2017.

[6] Y. Li, S. Liu, J. Yang, and M. Yang. Generative face completion. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5892–5900, 2017.

[7] Tamoghna Roy, Tim O'Shea, and Nathan West. Generative adversarial radio spectrum networks. pages 12–15, 05 2019.

[8] Hao Ye, Le Liang, Geoffrey Li, and Biing-Hwang Juang. Deep learning based end-to-end wireless communication systems with conditional gan as unknown channel. *IEEE Transactions on Wireless Communications*, PP:1–1, 02 2020.

[9] T. J. O'Shea, T. Roy, and N. West. Approximating the void: Learning stochastic channel models from observation with variational generative adversarial networks. In *2019 International Conference on Computing, Networking and Communications (ICNC)*, pages 681–686, 2019.

[10] Y. Yang, Y. Li, W. Zhang, F. Qin, P. Zhu, and C. Wang. Generative-adversarial-network-based wireless channel modeling: Challenges and opportunities. *IEEE Communications Magazine*, 57(3):22–27, 2019.

[11] David McLean. Kevin gurney, an introduction to neural networks, university college london (ucl) press, 1997. isbn 1-85728-673-1 hb. £14.95. xi+234 pages. *Nat. Lang. Eng.*, 6(2):203–204, June 2000.

[12] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[16] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[18] G. L. Turin, F. D. Clapp, T. L. Johnston, S. B. Fine, and D. Lavry. A statistical model of urban multipath propagation. *IEEE Transactions on Vehicular Technology*, 21(1):1–9, Feb 1972.

[19] Ayush Bharti, Ramoni Adeogun, and Troels Pedersen. Estimator for stochastic channel model without multipath extraction using temporal moments. In *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), United States, August 2019. IEEE. 20th IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC) 2019, SPAWC ; Conference date: 02-07-2019 Through 05-07-2019.

[20] Ayush Bharti, Laurent Clavier, and Troels Pedersen. Joint statistical modeling of received power, mean delay, and delay spread for indoor wideband radio channels. In *European Conference on Antennas and Propagation*, 2020.

[21] C. Gustafson, D. Bolin, and F. Tufvesson. Modeling the polarimetric mm-wave propagation channel using censored measurements. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2016.