Master's thesis

Master's Programme in Data Science

Detecting Anomalies in GNSS Signals with Complex-valued LSTM Networks

Savolainen Outi

December 8, 2022

Supervisor(s):	Associate Prof. Laura Ruotsalainen
	Dr. Arul Elango
Examiner(s):	Associate Prof. Laura Ruotsalainen
	Dr. Arul Elango

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Koulutusohjelma — Utbildningsprogram — Degree programme Master's Programme in Data Science

Tekijä — Författare — Author

Savolainen Outi

Työn nimi — Arbetets titel — Title

Detecting Anomalies in GNSS Signals with Complex-valued LSTM Networks

Työn laji — Arbetets art — Level Master's thesis	Aika — Datum — Month and year December 8, 2022	Sivumäärä — Sidantal — Number of pages 55

Tiivistelmä — Referat — Abstract

Today, Global Navigation Satellite Systems (GNSS) provide services that many critical systems [1] as well as normal users, need in everyday life. These signals are threatened by unintentional and intentional interference. The received satellite signals are complex-valued by nature, however, state-of-the-art anomaly detection approaches operate in the real domain. Changing the anomaly detection into the complex domain allows for preserving the phase component of the signal data.

In this thesis, I developed and tested a fully complex-valued Long Short-Term Memory (LSTM) based autoencoder for anomaly detection. I also developed a method for scaling of complex-numbers that forces both real and imaginary units into the range [-1,1] and does not change the direction of a complex vector. The model is trained and tested both in the time and frequency domains, and the frequency domain is divided into two parts: real and complex domain. The developed model's training data consists only of clean sample data, and the output of the model is the reconstruction of the model's input. In testing, it can be determined whether the output is clean or anomalous based on the reconstruction error and the computed threshold value.

The results show that the autoencoder model in the real domain outperforms the model trained in the complex domain. This does not indicate that the anomaly detection in the complex domain does not work; rather, the model's architecture needs improvements, and the amount of training data must be increased to reduce the overfitting of the complex domain and thus improve the anomaly detection capability. It was also investigated that some anomalous sample sequences contain a few large valued spikes while other values in the same data snapshot are smaller. After scaling, the values other than in the spikes get closer to zero. This phenomenon causes small reconstruction errors in the model and yields false predictions in the complex domain.

ACM Computing Classification System (CCS):
Computing methodologies → Machine learning → Machine learning approaches → Neural networks

Avainsanat — Nyckelord — Keywords

GNSS, anomaly detection, complex-valued LSTM

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Life begins at the end of your comfort zone.
— Neale Donald Walsch

Contents

1. Introduction

The roots of the idea of using satellites for navigation are from the 1960s, and the first satellite for the Global Positioning System (GPS) was launched in 1978 [25]. The number of satellites and the different satellite constellations is increased over the years, and today, many critical infrastructures are built to rely heavily on accurate positioning, navigation, and timing (PNT) [1]. For example, financial transactions, communications, and electricity transmission systems need accurate timing for synchronization, while aviation, rescue operations, and logistics need accurate positioning services for safe vehicle operations.

The satellites are located in space, far from the earth's surface. Hence the signal power is weak at the time of reaching the receivers on the ground level [33]. Moreover, the Global Navigation Satellite System (GNSS) was initially designed to be used outdoors, especially in open areas [13]. Thus, artificial and natural constructions can block the signal entirely or reflect the signal resulting in multipath signals. In addition, solar activity may cause disruptions in the ionosphere and thus interfere with the satellite signals. These unintentional interference types can lead to inaccuracies in estimating the position or location of a receiver. In addition, unintentional disruption may happen due to incorrectly installed antennas or broken devices that interfere with the satellite signals.

The unintentional interference is only part of the problem. Intentional interference is an existing threat, and it can be divided into two main types: jamming and spoofing. Jamming exists, for example, when someone wants to hide their location for privacy reasons [26]. For this purpose, a jammer is used to send a signal to suppress or block the original signal sent by satellites [13]. Because the signal is sent near the earth's surface it has a higher power level that overpowers the original satellite signal. Thus, the receiver's ability to track the satellite signal is reduced, or it is blocked completely. This event is analogous to the phenomenon in a group of people in which the loud voices overpower the quiet ones. Instead of trying to block the satellite signal, spoofers try to trick the receivers to calculate a false position or timing by sending either clean but delayed signal or a signal containing false information but otherwise resembling the clean one.

The weak power level of GNSS signals as they reach the receivers makes them sensitive to both intentional and unintentional interference signals [6]. Interference, regardless of its type, produces multiple threats in our society. In urban areas, the multipath signals and other disruptions cause positioning errors, and in the future of autonomous vehicles, those errors lead to fatal accidents in the worst case. Already today, the rescue forces utilize navigation to find their destination [1]. The probability of higher material or bodily injuries increases if the navigation fails and the rescue services arrive delayed at the target destination. The calculated errors in timing cause problems in electricity grids as well as in banking transactions. Hence, robust interference detection systems are needed to react to the different types of detected anomalies accordingly and as soon as possible.

In recent years, anomaly detection in GNSS signals has been researched using different approaches; traditional methods, such as regression-based [17] or deep learning-based methods, like those mentioned in [12, 14]. A more detailed introduction to state-of-the-art anomaly detection can be found in Chapter 2. The common characteristic of all the studies is that they use real-valued parameters, and if complex values are involved, the real and imaginary parts are handled separately [2]. Radio frequency (RF) signals are complex-valued by nature, and transforming them into the real domain loses some crucial information about the signal. The existing threats encourage us to find good and accurate solutions for detecting anomalies in GNSS signals. Thus far, there are no developed models available that use a fully complex-valued neural network for this problem.

In my thesis, I developed a fully complex-valued Long Short-Term Memory (LSTM) based autoencoder for anomaly detection in GNSS to differentiate my work from the previous research. The input and the output values are in the complex domain, and the autoencoder uses only complex-valued parameters in its hidden layers. Only the output of the loss function must be in the real domain to solve the optimization problem. The developed model is tested with anomalous and clean data and compared to the real-valued counterpart. My research questions are: Is anomaly detection improved by using deep neural networks directly in the complex domain? Does a complex-valued neural network model perform better in terms of accuracy? Does it serve better in terms of computational time, e.g., are fewer epochs needed for training, and are there any significant differences in used time in each epoch? Are there significant differences in the computational time in the predicting phase? Does the usage of the memory differ from model to model?

The rest of this work is structured as follows: Chapter 2 presents related work and the state of the art of the current anomaly detection in GNSS signals. Chapter 3 introduces the basic properties of complex values and briefly introduces the effects of interference on GNSS signals. Chapter 4 explains the basic structure of a complex-valued long short-term memory network. In Chapter 5, I will introduce a novel complex valued autoencoder model I developed for anomaly detection. Chapter 6 presents the experiments done with the model, and the results are discussed in detail. Finally, in Chapter 7 I will present my conclusions and possible future works related to this topic.

2. Related work

This section will introduce the state-of-the-art anomaly detection approaches in GNSS signals. In recent years deep learning based methods for anomaly detection and classifying has become increasingly popular in the research field. Because of the existing threats, GNSS signal interference detection, classification, and mitigation have been studied vastly. In my thesis, I will focus merely on anomaly detection. Thus, pure classification and mitigation related works are left out.

Even though the popularity of deep learning based approaches is increased, traditional methods are still used. An integrated moving average (ARIMA) prediction algorithm is used in [19] to find the satellite time outliers and determine whether the signal is spoofed. Another statistical analysis based approach used for detecting spoofing with the help of pseudo-error at the time is introduced in [3]. The distribution of multiple directional antennas in phasor measurement units (PMUs) is used to detect the presence of antenna-specific timing anomalies caused by the spoofing and to correct the timing errors to ensure that reliable timing is provided to the PMUs. The PMUs record the voltage and current phasors in power network systems. This approach is specialized to work only in PMUs.

Another device-specific jamming and spoofing detection system was developed for devices using the Android operating system [30]. The anomalies were detected by comparing together the location and time measurements received from the GNSS signals and the mobile network, and the classification was done based on the given thresholds. One more approach takes advantage of external devices [5]. Spoofed signals can be detected by using a network of low-cost spectrum sensors, while in spoofed areas, sensors deliver different readings than in non-spoofed regions. The benefit of using this type of approach is that the affected region can be estimated. However, the network of sensors must be vast enough that some subset of the sensors is undoubtedly outside of the possibly spoofed region. Spoofing is also detected in [29] with the help of support vector machines by using the cross-correlation among statistically significant GNSS observables and measurements, for example, the Carrier-to-Noise power density ratio (C/N_0). C/N_0 is the ratio of carrier signal power C presented in watts to the noise density N_0 in a 1-Hz bandwidth [13]. However, the main limitation is that the models presented in this and the previous paragraph can detect only one kind of anomaly type or are device or operating system specific.

Contrary to previous approaches, different kinds of anomalies are detected in [17]. A regression-based anomaly detection model is developed to determine whether the satellite signal is anomalous or clean. The model was trained with clean data from previous days. Based on the statistical rule, the deviation between the predicted and true values was flagged as either clean or anomalous. The approach is simple and provides good results. However, before any new predictions, the model must be trained with the data from previous days, and the collected data must be cleaned from the anomalous data shown on those days before the training can be started.

As mentioned, deep learning based approaches have become popular in recent years. Deep learning is claimed to overcome generalization difficulties when working with high-dimensional data [10]. In this context, generalization means that the model's prediction capability is good in situations where new data samples from the same distribution are introduced. The deep learning methods allow for finding the clean or anomalous signal characteristics of high dimensional data spaces. Convolutional neural network (CNN) is a deep learning approach that learns to find underlying patterns in the given data. For example, CNN was used to detect and classify different kinds of chirp signals in the time-frequency domain from spectrum image data [18]. Chirp signals belong to the family of jamming. Another approach used pre-trained CNN for detecting and classifying different signal disruptions from scalogram image data [7]. The process transforms complex-valued signal data into the real domain in both cases. Convolutional neural networks are also used for multipath signal detection: the network output is either clean, or it contains multipath effects [22]. The most significant difference from previous approaches is that in the network's input data, the intensity of pixels is present for both in-phase (I) and in-quadrature (Q) components of the signal. The problem with using CNN and especially 2D to 3D data in anomaly detection and classification is that the image tensors must be created, which is time-consuming. Additionally, a large number of labeled data is needed to avoid overfitting and train the model to recognize all kinds of anomalies.

For anomaly detection, often autoencoder-based solutions are meaningful, especially if the model is trained with clean data. These models detect anomalies based on the reconstruction error between the input and the output sequence. For example, in [12], a Long Short-term Memory (LSTM) network is trained with normal data, and prediction errors over a given threshold are flagged as anomalies. The data consists of spectral time series received from a satellite transponder. This approach is an example of an unsupervised learning task, of which the benefits are that there is no need for labeled data. On the other hand, to make any conclusions about the model's goodness, we have to know which sequences contain anomalies and which are clean of interference. Autoencoders that use only clean data in the training phase provide good results also in other fields: in [32] LSTM-based autoencoder was used for anomaly detection in CO_2 time series data.

However, the related works for anomaly detection use only real-valued signal measurements or characteristics as the model's input values. In [22], both I and Q components are considered, albeit divided into two separate channels. Phase information is lost when the satellite signal data is transformed from the complex domain to the real domain. Complex-valued neural networks have already been studied for years. For example, the requirements to develop fully complex-valued neural networks are studied in [15] and in [20]. However, it was not possible to use machine learning libraries, such as Keras or PyTorch, for training models fully in the complex domain until recently. That is most likely why many approaches, such as [31] or the approaches mentioned in [2], split the complex-valued data into real and imaginary components before forwarding the data to the neural network. Or the input and output values are kept in the complex domain like in [11], but the values are split into two components in each model layer to do the multiplications according to the computation rules defined for complex numbers. Moreover, I was not able to find any fully complex-valued neural network architectures for anomaly detection in GNSS signals.

To differentiate my work from the previous works, I developed a fully complex-neural network for anomaly detection in GNSS data. Compared to most of the related works in anomaly detection in satellite signals, the developed model can detect different kinds of anomalies and flag the sample either as clean or anomalous based on the input sequence.

3. Preliminaries

This chapter introduces the basics of complex numbers. Additionally, I will briefly introduce GNSS and explain the different kinds of GNSS interference types and their effects on positioning and timing.

3.1 Properties of Complex Numbers

This section introduces the essential characteristics of Complex numbers.

Definition 3.1.1 (Complex numbers)**.** The domain of complex numbers consists of pairs of two real numbers [21]

$$\mathbb{C} := \{(x, y) \colon x, y \in \mathbb{R}\}.$$

The common presentation of the complex numbers is $z = x + yi$, where i is the root of $i^2 = -1$, x is the real unit, and y represents the imaginary unit. Absolute value r is commonly known as *magnitude* and is calculated by using both real and imaginary parts as follows

$$r = |z| = \sqrt{x^2 + y^2} \tag{3.1}$$

The definition of addition and multiplication in the complex domain is formulated as follows:

$$(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2) \tag{3.2}$$

$$(x_1, y_1) \cdot (x_2, y_2) = (x_1x_2 - y_1y_2, x_1y_2 + y_1x_2) \tag{3.3}$$

The addition is straightforward, and only two computations are needed for two variables that form the complex variable. Multiplication, however, increases the number of calculations required to get the final product. Complex conjugate is an important feature in the complex-valued neural networks, as explained in Chapter 4, and it is derived by changing the sign of the imaginary part of the original number.

$$z^* = (x + iy)^* = x - iy \tag{3.4}$$

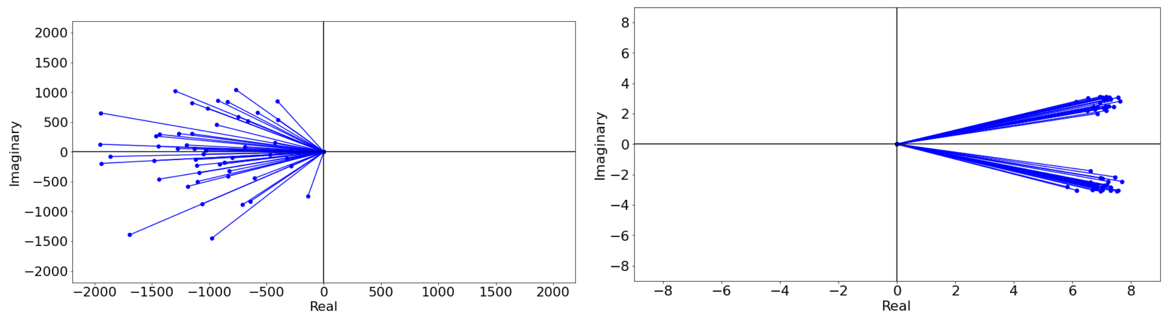
In addition to the regular presentation of complex numbers, they can be presented in *polar* form

$$re^{i\theta} = r(\cos\theta + i\sin\theta), \tag{3.5}$$

where r is the magnitude and θ is the phase. The magnitude presents the length of the complex vector starting from the origin, whereas the phase $\theta = \arg\frac{y}{x} = \tan^{-1}\frac{y}{x}$ is the angle between the real axis and the vector. The polar form presentation is useful when calculating the natural logarithm of a complex number. The natural logarithm or the principal branch of the logarithm is defined as follows

$$\ln z = \ln re^{i\theta} = \ln r + i\theta \tag{3.6}$$

for nonzero $z \in \mathbb{C}$ such that $\theta \in (-\pi, \pi]$. Computing the natural logarithm changes the direction of the complex vector that can be seen in 3.1.



(a) 50 different complex values mapped to Cartesian coordinates.

(b) Same 50 complex numbers as shown in 3.1a after taking the natural logarithm mapped to Cartesian coordinates.

Figure 3.1: Both figures contains 50 different complex values. In 3.1b natural logarithm is taken of the numbers before plotting them. The Cartesian coordinates consist of the real and imaginary axis.

Division of a complex number with a positive real value does not change the angle θ of the complex number as the relation between real and imaginary parts does not change. If we are given a complex value $z = x + yi$ and c, where $x, y \in \mathbb{R}$ and $c \in \mathbb{R}^+$, the angle $\theta = \tan^{-1}(y/x)$ of the given value z persists also for the scaled complex value z/c.

Proof. Assume that $z = x+yi$ with arbitrary values $x, y \in \mathbb{R}$ exists. The angle between the complex vector z and real axis is defined as $\theta = \tan^{-1}(y/x)$. Now, we want to prove that the equation $\tan^{-1}(y/x) = \tan^{-1}((y/c)/(x/c))$ holds for any $c \in \mathbb{R}^+$. Next, we choose a multiplier $c/c = 1$, and according to the calculation rules, we get

$$\theta = \tan^{-1}(y/x) = \tan^{-1}\left(\frac{cy}{cx}\right) = \tan^{-1}\left(\frac{\frac{y}{c}}{\frac{x}{c}}\right)$$

and

$$\theta = \tan^{-1}\left(\frac{\frac{y}{c}}{\frac{x}{c}}\right) = \tan^{-1}\left(\frac{cy}{cx}\right) = \tan^{-1}\left(\frac{y}{x}\right)$$

Thus, the assumption $\tan^{-1}(y/x) = \tan^{-1}((y/c)/(x/c))$ holds. $\square$

3.2 Global Navigation Satellite System

Global Navigation Satellite System (GNSS) consists of four different systems: Global Positioning Systems (GPS) from the United States, European Galileo, GLONASS from Russia, and Chinese BeiDou [13]. In addition, there exist other regional satellite navigation systems operating, such as the Japanese Quasi-Zenith Satellite System (QZSS) and Indian NavIC. For a three-dimensional position of a receiver and the correct timing, signal information from a minimum number of four satellites is needed. To simplify the mathematics used for the computations, we can say that position is computed from the traversed time between a satellite and a receiver.

Satellites transmit navigation signals on different carrier frequencies [13]. For example, GPS bands L1 and L2 have their central frequencies at 1575.42 MHz and 1227.6 MHz, whereas Galileo's E1 and E6 bands have their corresponding central frequencies at 1575.42 MHz and 1278.75 MHz. As a result, there is overlapping in some central frequencies. The different interference types can affect one or more bands [6]. Interference can be classified into three categories: narrowband interference (NBI), wideband interference (WBI), and continuous-wave interference (CWI). This categorization is done based on the signal characteristics in the frequency domain; that is, each category differs by how much their spectral occupation is with respect to the GNSS signal bandwidth. In NBI, the occupation is smaller than the GNSS signal bandwidth, whereas WBI occupies approximately the whole bandwidth. In CWI, the occupations are the smallest; it appears as a single tone in the frequency domain. Interference categories can be thought of as how the signal looks like when examined in the frequency domain. In contrast, the interference types can be considered as how the interference happens or is executed.

The interference types can cause inaccuracies in calculating the position and timing or even prevent these events completely [13]. For example, unintended interference can be caused by other RF transmitters out of the band frequencies placed near receiver antennas. The band frequencies are the frequencies in which the different satellite systems operate. The signal harmonics may collide with the satellite signal bandwidths, and if this phenomenon occurs, the signals are disrupted. Irregularities in the ionospheric layer can cause a signal distortion phenomenon, which can reduce the tracking capability of a receiver for short time periods. Satellite signals may be blocked or reflected because of artificial or natural constructions in urban areas, mountainous regions, or near water [6]. Multipath occurs if the signal's echoes are received in addition to the line-of-sight signal. The line-of-sight (LOS) means the direct signal between the satellite and the receiver. The multipath can be divided into two categories; specular and diffuse depending on the surface on which the signal is reflected.

As mentioned before, spoofing is a general term for one type of intentional interference. Information about message content, frame structure, and signal characteristics are publicly available for civilian signals, and they can be used to create fake signals [6]. In general, spoofing is the act in which counterfeit GNSS-like signals are broadcast. A spoofer generates the signals autonomously and sends them to the target receivers. Because of the higher signal power level, the user device receives a combination of both authentic and spoofed signals, which yields a calculation of position and timing based on incorrect information. The satellite signals can also be intercepted and delayed before being sent further. This type of spoofing is called meaconing. The only differences to the correct transmitted signals are now the positive delay and amplitude of the signal. In spoofing, the target group is often known beforehand.

It is more challenging to organize a spoofing attack than a jamming attack. This is because developing and constructing a spoofing device requires deep knowledge and understanding of the domain [6]. In contrast, a jammer, a piece of equipment used to transmit the jamming signal, can be purchased online relatively cheaply. With jammers, high-power interference signals are used to suppress or shield the spectrum of legitimate satellite signals. For a receiver under a jamming attack, it is difficult to track the clean signals, or in the worst case, it can not track them at all. Different jammers have their own spectrum characteristics, which can be used for categorizing the different jamming types. For example, chirp signals are mapped into WBI because of their temporal linear variation characteristics over the frequency band. Pulse signals can be categorized as their own group because they have an on-off status of several microseconds that alternates in time, which is characteristic of this type of signal.

4. Complex neural networks

Complex-valued neural networks have the same characteristics as their real-valued counterparts. The familiar functionality in the real domain – sorting numbers by their magnitude – does not work in the complex domain. This is clearly seen in an example, where four complex parameters are defined as follows $z_1 = 1 + 1i, z_2 = 1 - 1i, z_3 = -1 + 1i$, and $z_4 = -1 - 1i$. Now, with the rules of calculating the modulus in (3.1), we have $|z_1| = |z_2| = |z_3| = |z_4| = \sqrt{2}$. This means that the choice of activation functions that handle only complex numbers is limited. Moreover, the loss optimization process needs a real-valued output of a loss function because of the impossibility of minimizing the loss in the complex domain.

In many previous approaches, the complex numbers are not used directly as a whole, but real and imaginary units are handled separately [2]. If the numbers are split into real and imaginary units, a neural network's input size and parameter space are doubled. To keep the size of the input and the model's parameters at the same level as in the real-valued networks, the neural network parameters and operations must be switched entirely into the complex domain.

A deep neural network is a generic term for construction that have one or more hidden layers between the input and output layer [10]. In the forward phase, the neural network finds characteristic features of the input based on the model's weights and bias term. During backpropagation, the model's parameters are updated to minimize the loss, or in other words, the error between the model's predicted and the expected value. Activation functions are used to introduce non-linearity for the model. Based on the value from the activation function, a neuron either sends the output into the next layer of neurons or holds it back.

In this chapter, I first describe the LSTM cell structure, followed by an explanation of the basic components in the LSTM-based neural network architecture from the perspective of the complex domain.

4.1 LSTM Properties

Long Short-Term Memory (LSTM) is one of the modifications of the recurrent neural network (RNN). Unlike the basic feedforward neural networks, LSTM has recurrent connections to the previous hidden units of the network [10]. This character makes this type of network suitable for handling time series data, such as GNSS signals. This is because the information from the previous time step of an input sequence can be used for calculating the output of the current time step. The basic unit of an LSTM

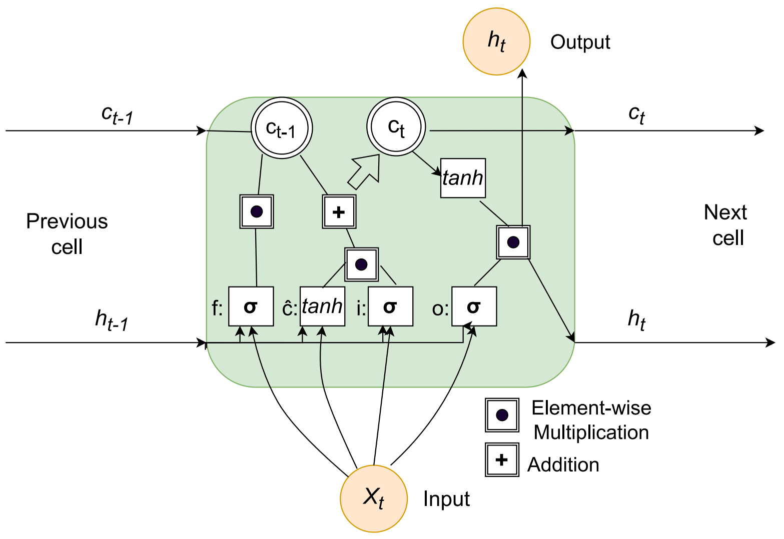


Figure 4.1: LSTM cell. Cell state and hidden state, which is also the output of the cell, are marked with c and h, respectively. The values received from the forget f and input i gates, estimated cell state $\hat{c}$, and the previous cell state are used to calculate the current cell state. The results from the output gate o and the *tanh* activation function are used for calculating the current hidden state and the cell output.

layer is an LSTM cell. This cell contains hidden state h, cell state c, estimated cell state $\hat{c}$ and three different types of gates: input i, forget f, and output o [10]. The main purpose of the gates is to decide and learn which input information should accumulate and when the old network state should be set to zero. The structure of the cell is illustrated in Figure 4.1.

The flow in the LSTM cell is simple: the input of a cell consists of the data at the current time step and the output from the previous time step. Next, input is fed to all three gates, in which both input tensors are separately multiplied with weights, and a bias term b is added to both products. Finally, before calculating the sigmoid activation, the tensors were added together. Additionally, estimated cell state $\hat{c}$ is calculated similarly to the gate values with one exception: this time, a hyperbolic tangent (tanh) activation function is used. The product of the element-wise multiplication of $\hat{c}$ and the input gate is added to the results of the multiplication of the forget gate and the previous cell state. This addition results in the current cell state. The hidden state is obtained by calculating the element-wise product of the output gate and the result from the tanh activation of the current cell state. There are different approaches to defining the LSTM cell calculus. For example, only one bias term can be used in each gate equation and the multiplication can be replaced with convolution [31]. In this thesis, the equations follow the implementation used in PyTorch library [23] as they are used in experiments.

The equations for activation and update formulas for the time step t for each gate and the states are the following [10, 23]

$$\begin{aligned}
i_t &= \sigma(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}) && (4.1)\\
f_t &= \sigma(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}) && (4.2)\\
\hat{c} &= \tanh(W_{xc}x_t + b_{xc} + W_{hc}h_{t-1} + b_{hc}) && (4.3)\\
c_t &= f_t \circ c_{t-1} + i_t \circ \hat{c} && (4.4)\\
o_t &= \sigma(W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}) && (4.5)\\
h_t &= o_t \circ \tanh(c_t), && (4.6)
\end{aligned}$$

where i_t, o_t, and f_t represent the input, output, and the forget gates, c_t and h_t represents the cell and hidden state, and the layer's input is marked as x_t at time step t. The biases are marked as $b_{xi,xo,xf,xc}$ for input and $b_{hi,ho,hf,hc}$ for the hidden state. Additionally, $W_{xi,xo,xf,xc}$ and $W_{hi,ho,hf,hc}$ are the weights for the input value and previous hidden state, respectively. Hadamard-product or the element-wise multiplication is marked with $\circ$.

An LSTM layer contains one or more layers of cells (Figure 4.2). Each layer contains a cell for each time step in the input sequence. First, weights and bias terms are initialized based on equations explained in Section 4.3. Both hidden and cell states are initialized to zero. In the case of a stacked cell layer structure, the input of the hidden cell layers is the output sequence from the previous layer. Either the final hidden state or output sequence can be used as input for the following layers. However, the size of the output tensor differs from the hidden state tensor. This is because the batch size and the sequence length of the output are the same as the input sequence, whereas the hidden state size is formed from batch size and the final hidden states from previous cell layers. The output tensor has to be resized and repeated to get the correct sequence length for the next layer in the network.

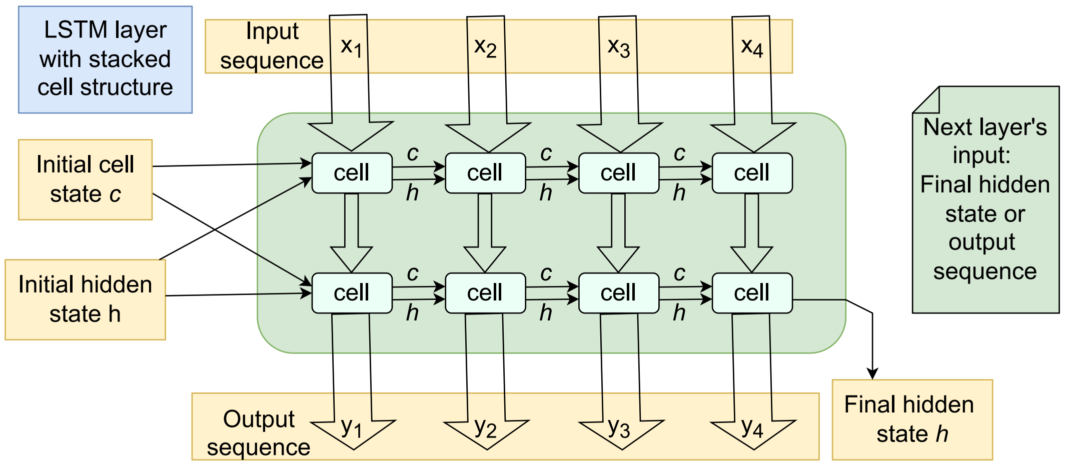


Figure 4.2: Each LSTM layer can have one or more stacked cell layers. In the picture, a structure of two stacked cell layers is presented. Both cell and hidden states are initialized to zero for the first instance of the input sequence. The input of the next cell layer is the output from the previous cell layer. The final hidden state or output from the last cell layer can be used as an input for the next layer in the neural network.

4.2 Activation Functions

In the neural network architecture, activation functions introduce non-linearity for the model. That is, they are used to decide whether a neuron should be activated or not. In the real domain, there are multiple choices for the activation functions for different neural network architectures. Some complex characteristics must be considered when changing the neural networks into the complex domain. For example, one restriction to complex numbers is that they cannot be sorted by their magnitude. Thus, for example, Rectified Linear Unit (ReLU) function defined as $\max(0, z)$, where z is a complex number, cannot be directly used in the complex domain. One of the previous approaches explained in [31, 2] is to split a complex number into its real and imaginary units and compute any given activation function f for each unit separately.

$$f(z) = f(Re(z)) + if(Im(z)), \quad \text{where } z \in \mathbb{C}. \tag{4.7}$$

However, in the approaches that use split-wise activation functions, the real and imaginary parts of the complex values are often split into separate real-valued input tensors [31, 2]. Another approach is to separate the tensors into real and imaginary units inside the network and combine the result of the calculations as a complex-valued output tensor [11]. Both approaches use the complex multiplication rules (3.3) to obtain the correct real and imaginary values from the network layers. This doubles the number of needed parameters, as both units need their own weights. This increases the complexity of the neural network. Thus, fully complex activation functions are to be used to reduce the number of network parameters to the same level as in real-valued networks.

Generally, no group of activation functions exists that has been deemed the best for both real and complex neural networks [2]. However, some specific neural network architectures, such as LSTM, are typically implemented to use sigmoid (4.8) and tanh (4.9) activation. These two functions can be used directly in the complex domain [28, 23].

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{4.8}$$

$$\tanh(z) = \frac{\sinh z}{\cosh z} = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{4.9}$$

where $z \in \mathbb{C}$ and the sigmoid function is marked as σ. Both functions have singularity points. For tanh, those points are found when the real part is zero, and the imaginary part is $i(1/2+n)\pi$, where $n \in \mathbb{N}$ [28]. For the sigmoid function, the points are defined for complex numbers $i(1+n)\pi$, and this can be proven with the help of (3.5). Assume $z = x + yi$, where $x = 0$ and $y = n\pi$. Now

$$\begin{aligned}\sigma =& \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(x+yi)}} = \frac{1}{1+e^{-yi}} = \frac{1}{1+(\cos n\pi + i\sin n\pi)^{-1}} \\ =& \frac{1}{1+(-1+0)^{-1}} = \frac{1}{1-1} \Rightarrow \text{undefined}\end{aligned}$$

Thus, it is important to initialize the model's parameters correctly, but also the complex input variables must be scaled to avoid singularities during the computation. The scaling of the complex numbers is discussed in detail in 5.2.

4.3 Model's Parameter Initialization

The model's parameters can be initialized in different ways, but there are two main rules to consider of. First, it is not recommended to initialize parameters, especially weights, with a constant. Assume that weight tensors $\mathbf{W}$ and bias tensors $\mathbf{b}$ are initialized with zero. Now, by examining gate Equations (4.1, 4.2, 4.5), we get, based on the definition of the sigma function (4.8), the output tensors consists of constant 1/2 because $\sigma(0) = 1/2$. Following up Equations (4.9) and (4.3), we see the output tensor for $\hat{c}$ is a zero tensor. Hidden and cell states are initialized to be zero tensors, and thus the cell state update yields zero (4.4). Finally, taking the Hadamard product from the output gate and $\tanh(\hat{c})$, we get zero tensors again because $\tanh(0) = 0$. This denotes that the forwarded cell and hidden states c and h, and thus the final output is zero tensors, and the network cannot learn.

Secondly, it is advisable to initialize the parameters with small random initial values to avoid exploding gradients and to help the model's changes to find a good minimum point during the backpropagation [24]. Different types of weight initialization exist. Technically bias can be initialized to be a constant, but for example, default weight and bias initialization executed in the PyTorch library draw these parameters from the uniform distribution, e.g. $\theta \propto \mathcal{U} \sim (-\sqrt{1/(k)}, \sqrt{1/(k)})$, where $k = \frac{1}{hidden\_dim\_size}$ [23]. Using Rayleigh distribution for drawing the modulus and uniform distribution for drawing the phase and initializing the weights with the help of these values is proposed in [31]. Mathematically this weight initialization can be formulated as follows.

$$\begin{aligned} r &\propto \text{Rayleigh} \sim (std) \\ \theta &\propto \mathcal{U} \sim (-\pi, \pi) \\ \text{weights} &= r\cos(\theta) + i\, r\sin(\theta) \end{aligned} \tag{4.10}$$

where standard deviation $std = 1/(n_{input} + n_{output})$, where n_{input} and n_{output} are the numbers of the input and output units, and finally variables r and θ are the modulus and the phase, respectively. The main rule by initialization is that the resulting weights are in reasonable scale to avoid singularities or exploding gradients during the model's training.

4.4 Backpropagation

In the neural networks, backpropagation is used to adjust the model parameters based on the difference between the predicted, and the true value [10]. That is, we want to optimize the model's parameters to make better predictions in the future. The actual learning does happen in this phase of the learning process. We use a loss function to evaluate the current state of the model's prediction ability. The model's parameters are updated based on the computed gradients from the current epoch's loss and the gradients of activation functions. As already mentioned, we can apply neither minimization nor maximization to complex values because their magnitudes cannot be compared.

In general, the functions in the neural network designed for the complex domain can be either holomorphic or non-holomorphic. Holomorphicity means that the function is differentiable in all points in open set $U \in \mathrm{C}$ [21]. However, functions that are differentiable at all points in the real domain might not be entire in the complex domain. For example, both sigmoid and tanh functions have singularities in the imaginary axis at points $i\{2n + 1\}\pi$ and $i\{n + 1/2\}\pi$, respectively as mentioned in

4.2. Well-defined derivation rules exist, but the functions have to be holomorphic and satisfy the Cauchy-Riemann conditions

$$\frac{\partial u(x,y)}{\partial x} = \frac{\partial v(x,y)}{\partial y} \quad \text{and} \quad \frac{\partial v(x,y)}{\partial x} = -\frac{\partial u(x,y)}{\partial y},$$

where u and v are the real-valued functions, and parameters x and y are the real and imaginary units of a complex variable, respectively. However, the output of a loss function used in the complex-valued neural network must be in the real domain. Otherwise, the minimization task cannot be done. This yields the situation in which we obtain $v(x,y) = 0$ for all points in the complex domain, and thus the Cauchy Riemann conditions are not satisfied [28]. Hence, the well-defined complex derivation rules cannot be used in the backpropagation phase for the loss function.

Even though we cannot take advantage of basic complex derivation rules, the solution to the problem exists. Mathematician Wilhelm Wirtinger derived calculus named by him that can be used for both holomorphic and non-holomorphic functions [9]. This calculus can be applied for the backpropagation phase for computing the gradients of loss and activation functions. The Wirtinger derivative is defined as two derivatives, both for any arbitrary complex value z and its conjugate z^* (3.4) as follows

$$\frac{\partial f}{\partial z} = \frac{1}{2}\left(\frac{\partial}{\partial x} + i\frac{\partial}{\partial y}\right) \tag{4.11}$$

$$\frac{\partial f}{\partial z^*} = \frac{1}{2}\left(\frac{\partial}{\partial x} - i\frac{\partial}{\partial y}\right). \tag{4.12}$$

Both of the derivatives (4.11) and (4.12) can be applied to any arbitrary function in the complex domain, including the holomorphic functions. However, in the specific case of holomorphic functions, it is important to recognize that they do not depend on z^*, which yields the results $\frac{\partial f}{\partial z^*} = 0$ [9]. The gradients can directly be derived from the Wirtinger derivative and are defined as follows

$$\frac{\partial f}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{z}_1} \\ \frac{\partial f}{\partial \mathbf{z}_2} \\ \vdots \\ \frac{\partial f}{\partial \mathbf{z}_n} \end{bmatrix} \quad \text{and} \quad \frac{\partial f}{\partial \mathbf{z}^*} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{z}_1^*} \\ \frac{\partial f}{\partial \mathbf{z}_2^*} \\ \vdots \\ \frac{\partial f}{\partial \mathbf{z}_n^*} \end{bmatrix} \tag{4.13}$$

Choosing a loss function is an important part of neural network design. Many loss functions cannot be used directly in complex-valued networks. This is because they do not deliver real-valued output for a complex-valued input by design. *Mean absolute error* (MAE) loss function can be selected straight because it calculates the absolute value of the difference between two tensors. This procedure converts the complex-valued tensors into the real domain. Some of the existing loss functions need only minor altering to convert them suitable for complex networks. For example, in *mean squared error* (MSE) function, before taking the square root of the complex-valued difference, computation of absolute value is added.

$$\begin{aligned} \text{MAE} =& \mathcal{L}(\mathbf{x}, \mathbf{y}) = (l_1 + \cdots + l_N)/N, \quad \text{where } l_n = |x_n - y_n| && (4.14) \\ \text{MSE}_{modified} =& \mathcal{L}(\mathbf{x}, \mathbf{y}) = (l_1 + \cdots + l_N)/N, \quad \text{where } l_n = (|x_n - y_n|)^2 && (4.15) \end{aligned}$$

In the equations (4.14) and (4.15), a tensor $\mathbf{x}$ corresponds to predicted values of the network, whereas the tensor $\mathbf{y}$ is the true values and N is the total number of the calculated error values between each variable x_n and y_n of the tensors, where $n \in \{1, \dots N\}$.

As mentioned earlier, the output of the loss function is in the real domain even though all other network parameters are in the complex domain. The update rule that resembles the real-valued counterpart must be updated to process complex numbers. The updated rule is defined for stochastic gradient descent optimizer [10] as in Equation (4.16), but this can be expanded to different optimizer types such as Adam.

$$\theta = \theta - \eta g, \tag{4.16}$$

where θ are the model parameters, η learning rate, and g are the gradients of the model parameters in the previous step. The optimization step that updates the model's parameters in the domain $\mathbb{C}$ takes advantage of the Wirtinger calculus. In this case, both real and imaginary units of a complex-valued parameter must be updated. We can derive the update rule for stochastic gradient descent with the help of (4.12) as follows:

$$\begin{aligned} z_{t+1} =& x_t - \eta \cdot \frac{1}{2}\frac{\partial \mathcal{L}}{\partial x} + i \cdot \left(y_t - \eta \cdot \frac{1}{2}\frac{\partial \mathcal{L}}{\partial y} \right) \\ =& z_t - \eta \cdot \frac{1}{2} \left(\frac{\partial \mathcal{L}}{\partial x} + i \frac{\partial \mathcal{L}}{\partial y} \right) \\ =& z_t - \eta \frac{\partial \mathcal{L}}{\partial z^*} \end{aligned} \tag{4.17}$$

where η is the learning rate, $z_t = x_t + iy_t$ is one of the model's parameters in the previous step, and $\mathcal{L}$ is the loss function to be optimized. The updated rules of different optimization algorithms, such as Adam, differ from those in Equation (4.17), but those differences do not affect how the gradient is calculated. The derivative of the loss $\frac{\partial \mathcal{L}}{\partial z^*}$ can be rewritten as

$$\frac{\partial \mathcal{L}}{\partial z^*} = \left(\frac{\partial \mathcal{L}}{\partial s} \right)^* \frac{\partial s}{\partial z^*} + \frac{\partial \mathcal{L}}{\partial s} \left(\frac{\partial s}{\partial z} \right)^* \tag{4.18}$$

where s is the real-valued output of the loss function and z is the complex-valued input obtained from the linear layer [23]. The derivative of the loss function is not the only gradient that has to be computed. In backpropagation, the network is propagated from the loss back to the start of the network. Basically, gradients are calculated for all the activation functions. The difference compared to the forward pass is that now the directions are reversed. While computing the gradient the input of the function comes from the next network layer or cell.

5. Methodology

The first idea for approaching the problem was to develop a complex-valued neural network for detecting and classifying anomalies of which input values are the spectrograms of the given signals. This idea was based on different papers: complex-valued convolutional LSTM approaches in [31] for speech spectrum prediction, and in [8] polarimetric synthetic aperture radar (PolSAR) image classification, the convolutional neural network approaches for anomaly classification of different jamming types in [18] and classification of different kind of interference types in [7]. However, I abandoned this idea mainly because each of the data files includes sample signals with lengths of 200-250 ms or later a length of 1500 ms. Creating, saving, and reading 200 or 250 spectrogram images for training and testing purposes would have been computationally heavy. Additionally, this would have meant that in case of anomaly detection from streamed signal data, there would have been 1000 created spectrograms for each second if each millisecond had been checked for anomalies. Taking a snapshot, for example, every second would decrease the computational load but increase the delay of the detection.

The signal data is complex-valued, and the main signal characteristics can be found efficiently with the Fast Fourier transform (FFT) algorithm. This reduces the computational time because the network's input size decreases. Calculating the FFT from the I/Q data result in complex values, hence, we do not lose the information coded in the phase element of the signal. The good results in [32] in which autoencoders were used for anomaly detection successfully, encouraged to development of a complex-valued LSTM-based autoencoder for anomaly detection in GNSS signals. A large number of existing interference types was another reason to approach this anomaly detection problem with an autoencoder. Thus, if the model is used for binary classification – a sample is clean or anomalous – only clean data is needed for training, and the anomalies can be detected based on the reconstruction error computed in the prediction phase. Compared to other classification models, there is no need to collect and label a large number of data for training and testing. Because of the novelty of fully complex-valued neural networks, I was not able to find proper scaling methods for complex numbers. Therefore, I developed a scaling method for complex numbers in order to avoid singularity values in used activation functions. In this chapter, I will describe the data, the developed scaling method, the developed model architecture, and the workflow of the experiments.

5.1 Experiment Data

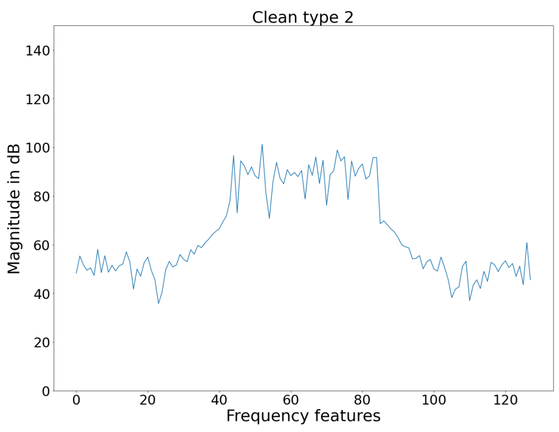
As jamming and spoofing are both illegal methods to interfere with the GNSS signals and collecting such data has its difficulties, I used generated data in the experiments in this thesis. Moreover, in this way, different kinds of possible interference scenarios can be provided for testing the trained model. The data was generated with Orolia's signal simulator from GSG-8 Series. Each interference file consists of in-phase in-quadrature (I/Q) signal data, of which in-phase represents the real unit and in-quadrature is the imaginary unit of the complex number. Each file contains data for 200, 250, or 1500 milliseconds. Each snapshot of one ms second long contains the sampling rate $16224 \cdot 2$ fixed in the GNSS simulator during the recording of the data.

The interference types used in the experiments were multipath, spoofing, and jamming. The six different types of jamming signals or their combinations are continuous wave interference (CWI), multi-continuous wave interference (MCWI), multi-continuous wave interference with three spikes in the central frequency region (TM-CWI), pulse, CWI with chirp interference (CWI-CI), and two types of CWI-pulse-CI.

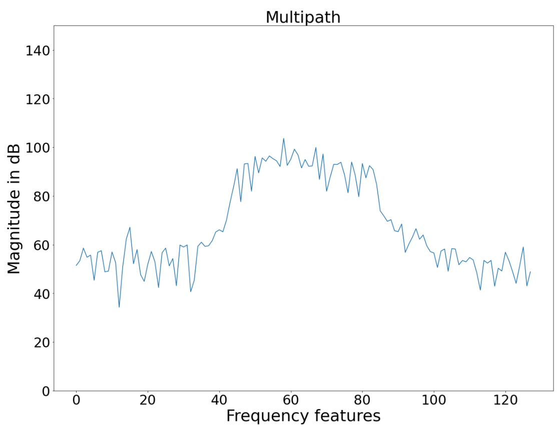
In time domain presentation, the number of signal frequency features for each millisecond is large: 32448 complex values. Using I/Q data directly as input is possible but has some limitations that are explained later on in Chapter 6. Hence, Fast Fourier transform (FFT) is used to transform the signal snapshots from the time domain to the frequency domain. The PyTorch library used for computing the FFT computes the Discrete Fourier Transform (DFT), which is defined as

$$\begin{aligned} F(k) &= \sum_{n=0}^{N-1} f(n) e^{\frac{-i2\pi kn}{N}} \\ &= \sum_{n=0}^{N-1} f(n)_{real} \cos\frac{2\pi kn}{N} + f(n)_{imag} \sin\frac{2\pi kn}{N} \\ &\quad - i\left[\sum_{n=0}^{N-1} f(n)_{real} \sin\frac{2\pi kn}{N} - f(n)_{imag} \cos\frac{2\pi kn}{N}\right] \end{aligned} \tag{5.1}$$

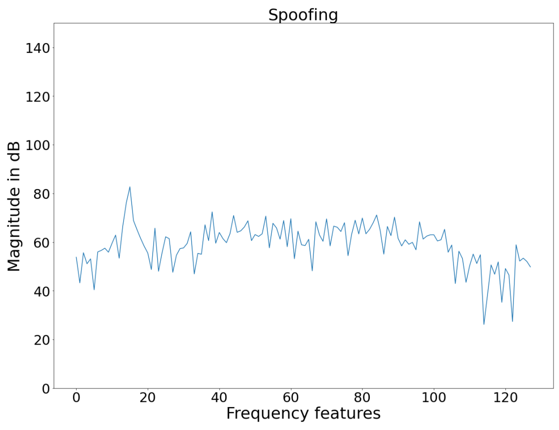
where $f(n) = f(n)_{real} + if(n)_{imag}$ is the complex time function, N represents the length of the array produced by FFT, which in our case is set to 128 [4]. The resulting 1-dimensional array from the FFT function contains the positive frequency terms first. The fftshift method, also provided by PyTorch, is used to rearrange the vector to have the center frequency in the middle and the negative frequency terms first.



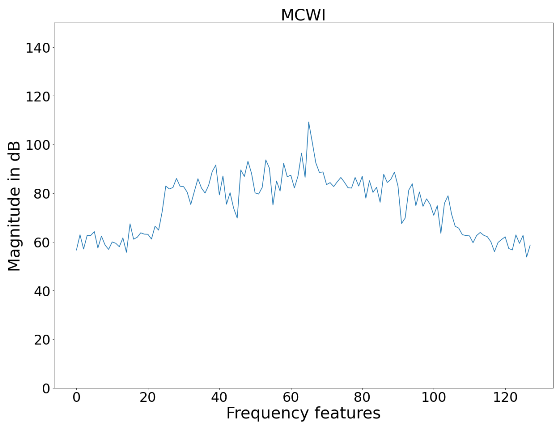
(a) Clean type 2



(b) Multipath



(c) Spoofing



(d) MCWI

Figure 5.1: Samples of clean type 2 used in training, multipath, spoofing, and MCWI (jamming) after calculating FFT and fftshift. Each sample is created from a 1 ms snapshot and scaled into dB by taking the base ten logarithms of the absolute values and multiplying them with a scalar 10. We can see that different types of signals provide different kinds of magnitude profiles. The illustrated values are used in the real-valued autoencoder that is used in the comparison.

The transform is shown in Figures 5.2 and 5.3 in which we have together five sample snapshots of the length 1 ms both in I/Q format and after computing FFT and fftshift. Especially, clean type 1 (5.3a, 5.3b) and multipath (5.2a, 5.2b) signals resemble each other when viewing the I/Q data, but changing from the time domain to frequency one, the differences are easier to see. However, the newer clean signal data, clean type 2 presented in Figures 5.3c and 5.3d resembles the multipath signal also in the frequency domain. The difference between both clean types and spoofed (5.2c, 5.2d) signal is clear already in the time domain; in this generated spoofed signal, the values in real and imaginary units fluctuate between -250 and 250, whereas in the clean sample the range is approximately between -1500 and 1500. Between both clean types and MCWI (5.2e, 5.2f), the difference is visible in the time domain, but especially it can be observed in the frequency domain due to the spikes in the central frequency region. For training purposes in the real domain, we have to map the data into real values. This is done by transforming the unit of measure from amplitude to magnitude in decibels with the formula

$$data\_in\_dB = 10\log_{10}|z|$$

where $|z|$ is the absolute value of the complex parameter z. This transformation for the same four samples discussed in the previous paragraph is illustrated in Figure 5.1. When comparing the clean sample type 2 (5.1a) and MCWI (5.1d) signals together, it is clear to see that MCWI appears as single tones in the frequency domain. In multipath (5.1b), the power level in dB and the shape of the figure are close to the clean type 2. The spoofed sample represented in Figure 5.1c stays mainly below the magnitude decibel levels and has a wider occupation on the frequency band compared to the clean type 2. The figures for the rest of the used anomaly types can be found in Appendix A.

5.2 Scaling of Complex Values

Because of the activation functions used in the LSTM layers, data needs to be scaled to avoid singularities and, thus, invalid outputs of the activation functions. For example, min-max scaling can be used in the real domain to fit the values between given intervals, typically between 0 and 1 or between -1 and 1. However, when scaling the complex values, this approach can not be used directly without changing the complex vector's direction and angle. Thus, I developed a method for scaling complex numbers.

Dividing a complex value with a scalar preserves the angle as mentioned in Section 3.1. I tested two different scalar types for scaling. The first approach was to use the maximum absolute value of each complex tensor which values were to be scaled. The second approach uses a maximum of maximum absolute value of the real or imaginary unit of the tensor that is to be scaled. The scaling functions are defined as

$$scaling_1 = \mathbf{Z}/\max(\text{abs}(\mathbf{Z})) \tag{5.2}$$

$$scaling_2 = \mathbf{Z}/\max(\max(\text{abs}(\mathbf{Z}_{real})), \max(\text{abs}(\mathbf{Z}_{imaginary}))) \tag{5.3}$$

where $\mathbf{Z} \in \mathbb{C}^{mx1}$. The first type scales the complex values so that absolute values of the complex numbers are mapped between -1 and 1. However, real and imaginary units are often centered closer to zero than by using the scaling_2. Based on this, it seems that the magnitude of the scalar and, thus, the results of scaled numbers can affect the training and, thus, the prediction ability of the model as explained in Section 6.2. The latter scaling type assures that both units range between -1 and 1, forcing at least one of the complex vectors to have either the imaginary or real part at the rand of the interval. The results of the two functions are shown in Figure 5.4. We can see that using maximum absolute value as denominator can sometimes give almost the same result as by using the function scaling$_2$.

5.3 Complex valued LSTM Autoencoder

Autoencoders consist of two parts: encoder and decoder [10]. First, the input data is given to the encoder, which seeks the essential patterns in the input sequences. Then, the decoder unit tries to predict the sequence given to the encoder unit. Ideally, the output of the autoencoder closely resembles the given input. For anomaly detection, the network is often trained only with clean data [32, 12] to get the reconstruction error as high as possible with anomalous data. The high-level architecture of the developed autoencoder is illustrated in Figure 5.5. In my thesis, I tested the autoencoder model in real and complex domains. The complex domain can be divided into two subcategories: Time and frequency. These three variants and their results are described in Chapter 6.

The workflow of a basic autoencoder can be described as follows: Input sequence that contains signal spectrum or I/Q data for all time steps of the sequence is given to the first LSTM layer in form *(batch size, sequence length, features)*. The first layer's purpose is to encode its input's main features. The spectrum data at the first time step is processed in an LSTM cell (Figure 4.1). The hidden and the cell state of the first cell is forwarded to the next cell, in which the input data from the second time step is processed. This procedure continues until the whole sequence is processed. The encoder tries to find a specified number of signal characteristics of which the signal can be decoded. The number of extracted features is set during the model initialization by the user. Before moving to the decoder unit, the decoder input must be decided. Both output and the final hidden state can be used as input.

If the final hidden state is used as an output, it must be reshaped into a form of (*batch size, hidden dimension*). The sequence length defines how often the reshaped tensor is repeated. The repeated tensor is again reshaped into form *(batch size, sequence length, hidden features)* before feeding the obtained tensor to the decoder unit that starts with the second LSTM layer. If the output sequence is chosen, it can be used as an input of the decoder directly.

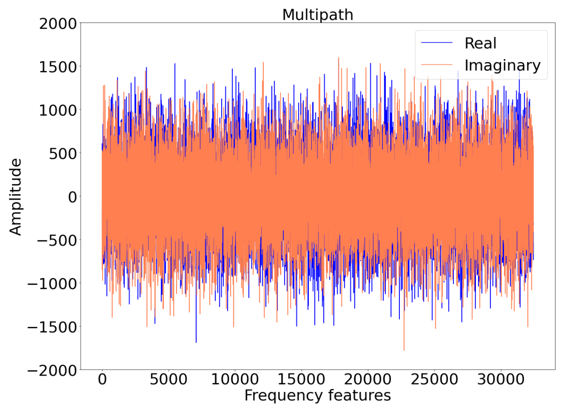
In the decoder unit, handling the time steps in the LSTM cells is equal to the first LSTM layer, but the output feature size will be increased to the size of the sample features. Finally, the output of the LSTM layer is processed through a linear layer which produces the final output of the model that can be compared to the input sequence.

Mean absolute error (L1Loss in PyTorch) is chosen as the criterion. The purpose of MAE is to measure the reconstruction error between the true and predicted values. Instead of a traditional stochastic gradient descent optimizer, we chose the Adam optimizer. Adam was chosen because is claimed to have only little memory requirements and to be computationally efficient [16]. The optimizer is used for updating the model's parameters based on the calculated gradient.

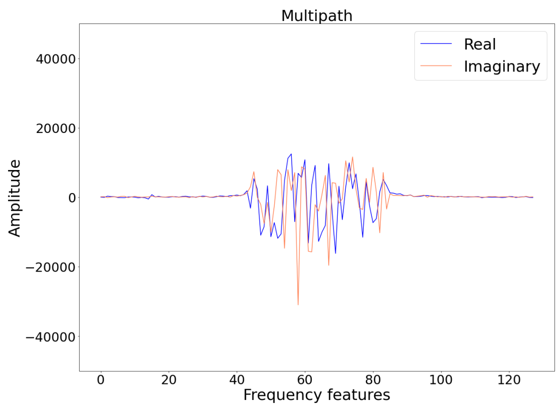
The model itself is trained over several epochs. The order of the sample sequences is shuffled before each epoch. Validation data is used to evaluate the correctness of the model during the training. As the training is completed, the test sequences, including clean samples, are tested and classified as an anomaly or clean based on the given threshold value. This value is determined to be the maximum reconstruction error in validation data from the last epoch.

The workflow of the whole algorithm is illustrated in Figure 5.6. The user first defines the FFT length, batch and hidden dimension's size, number of cell layers and epochs, and the learning rate. The data is transformed from the time domain to the frequency domain and scaled. If training is done with I/Q data, only scaling of the values is executed. For the anomalous data and clean data type 1, only test sequences are created. For clean data type 2, training, validation, and test sequences are created so that the model has not seen the test data during the training, and the validation data does not improve the model. The model is initialized using the used-defined hyperparameters. The weights are created either with default distribution, which is the uniform distribution or using Rayleigh and uniform distribution as described in 4.3. The bias term used in each layer is drawn from the uniform distribution. The training takes n epochs, and on each epoch first, the training set is iterated through with backpropagation, after which the model is evaluated with a separate validation set.

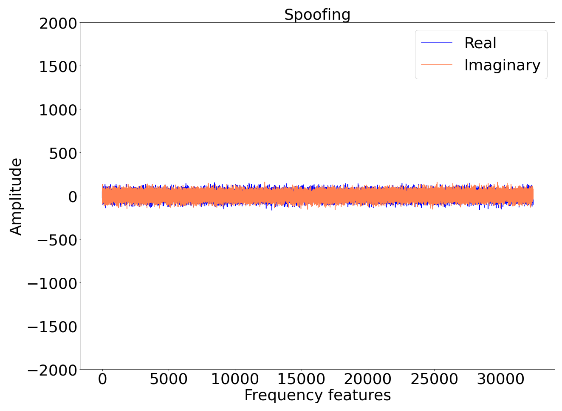
When the training is finished, the threshold that classifies the samples as clean or anomalous is computed from the last training epoch, and it is the maximum of the validation loss. The model is tested with both clean and anomalous samples that are unseen for the model. Finally, the totals of the predicted values, clean or anomalous, are counted for each sample to see the prediction capability of the model.



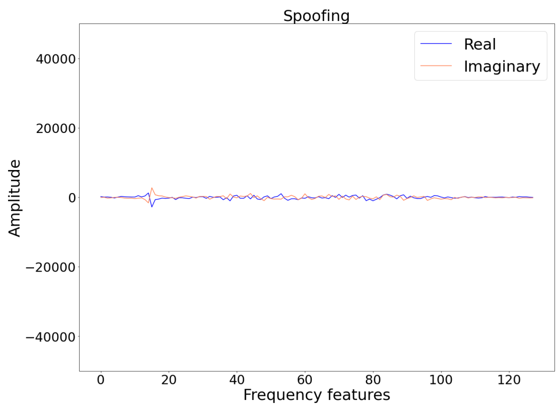
(a) Multipath I/Q data for 1 ms



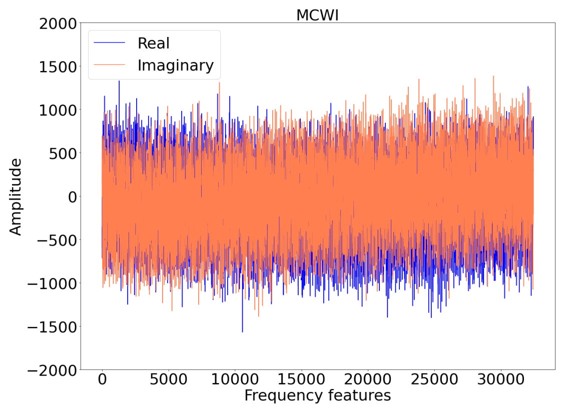
(b) Multipath after FFT and fftshift



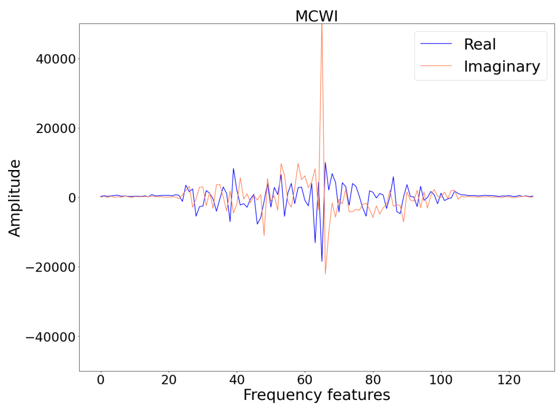
(c) Spoofing I/Q data for 1 ms



(d) Spoofing after FFT and fftshift

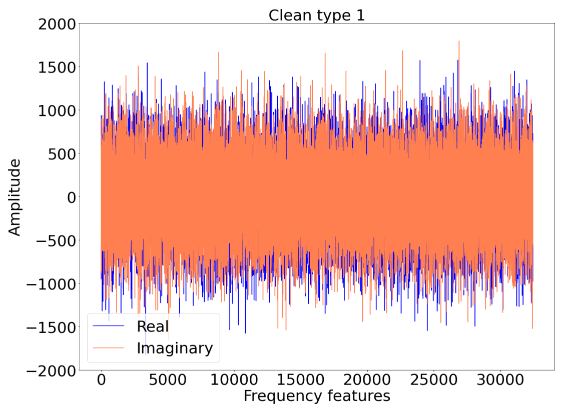


(e) MCWI I/Q data for 1 ms

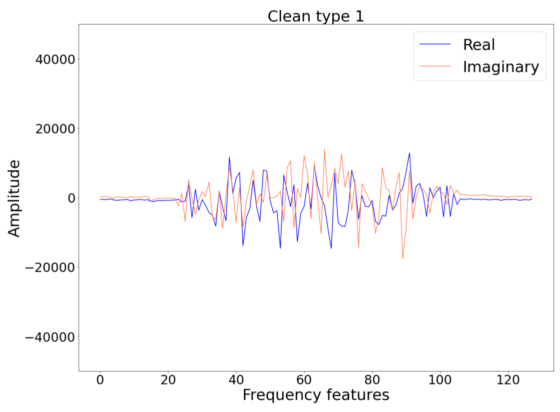


(f) MCWI after FFT and fftshift

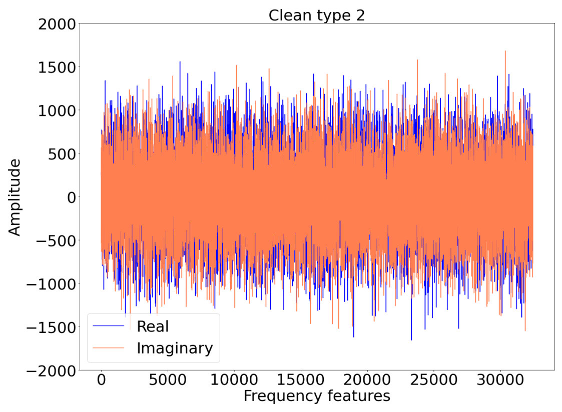
Figure 5.2: Samples of multipath, spoofing, and MCWI (jamming). We have a snapshot of 1 ms of I/Q data on the left column. On the right column, the same snapshot is after calculating FFT with a length of 128 and fftshift. Based on the I/Q data, it is difficult to say the difference between the samples except the spoofing, but as turned into the frequency domain, more signal characteristics are shown, and the differences between the samples are easier to see.



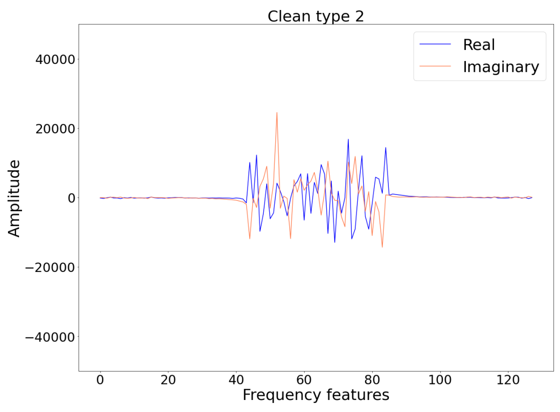
(a) Clean type 1 I/Q data for 1 ms



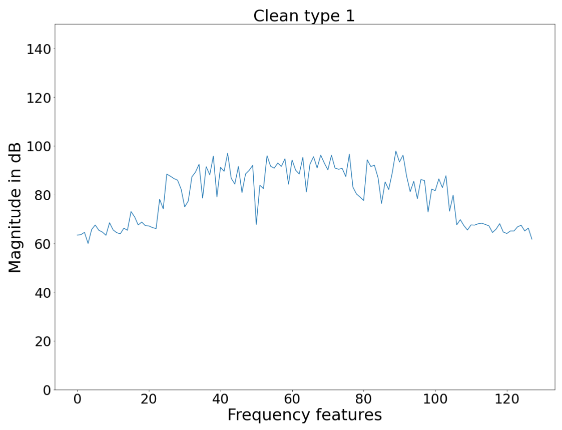
(b) Clean type 1 after FFT and fftshift



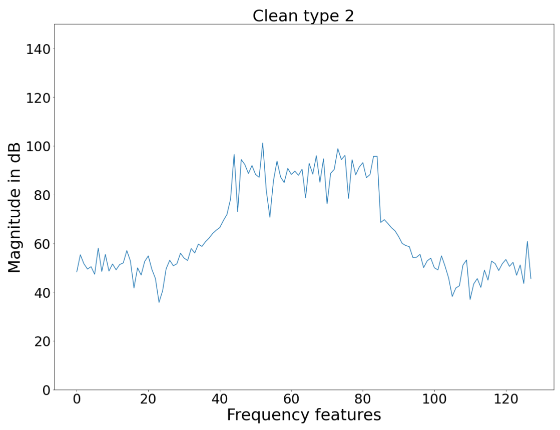
(c) Clean type 2 I/Q data for 1 ms



(d) Clean type 2 after FFT and fftshift

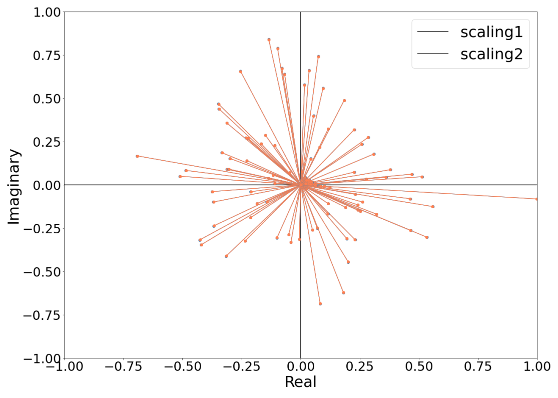


(e) Clean type 1 after FFT and fftshift and changing the signal into real domain

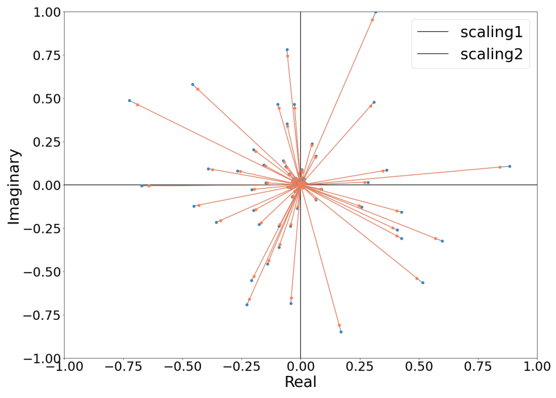


(f) Clean type 2 after FFT and fftshift and changing the signal into real domain

Figure 5.3: Samples of both clean types presented in the testing. Clean type 2 is used for the training. The difference between these types is the occupation on the frequency band. We have a snapshot of 1 ms of I/Q data on the left column. On the right column, the same snapshot is after calculating FFT with a length of 128 and fftshift. In Figures *e* and *f*, both clean data samples are presented after taking the logarithm base ten from absolute values and multiplying by 10.



(a) Results of both types of scaling functions in clean data type 1 (1 ms).



(b) Results of both types of scaling functions in clean data type 2 (1 ms).

Figure 5.4: Examples of results of the scaled values presented in the Cartesian coordinate system, where the y-axis and x-axis represent the imaginary and the real unit, respectively. In (a), we can see that scaling with maximum absolute value can produce nearly the same values as by using the function scaling$_2$. A clearer difference between those two methods is visible in (b).

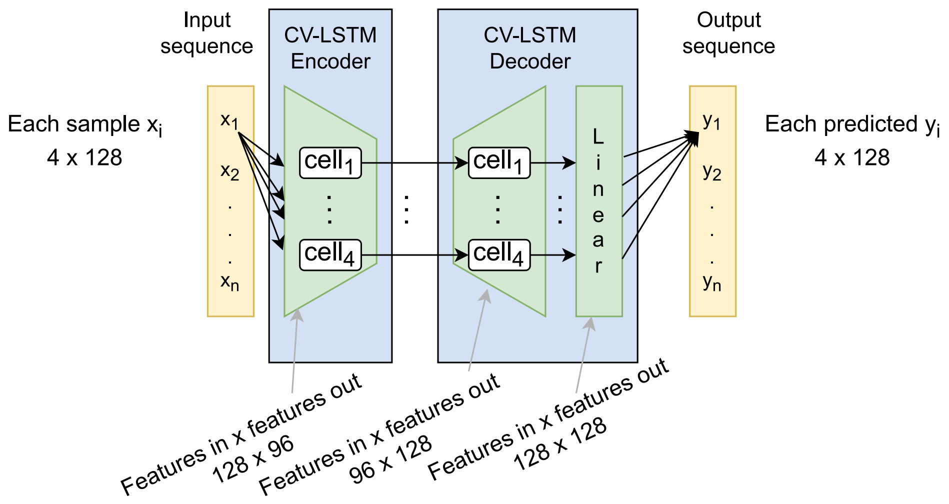


Figure 5.5: The final CVAutoencoder architecture. Here to simplify the explanation, it is assumed that the batch size is one. The model consists of two units: encoder and decoder. The encoder has only one LSTM layer, of which high-level dimensions are 128 x 96 representing the number of features in and out, respectively. The output features are fed into the decoder unit as input, and the output feature dimension from the LSTM layer is increased back to 128. The decoder unit includes a linear layer in which input and output dimensions are 128. The output of the linear layer consists of the predicted sequence of size 4 x 128. The only activation functions used in the network are included at the cell level in LSTM layers.

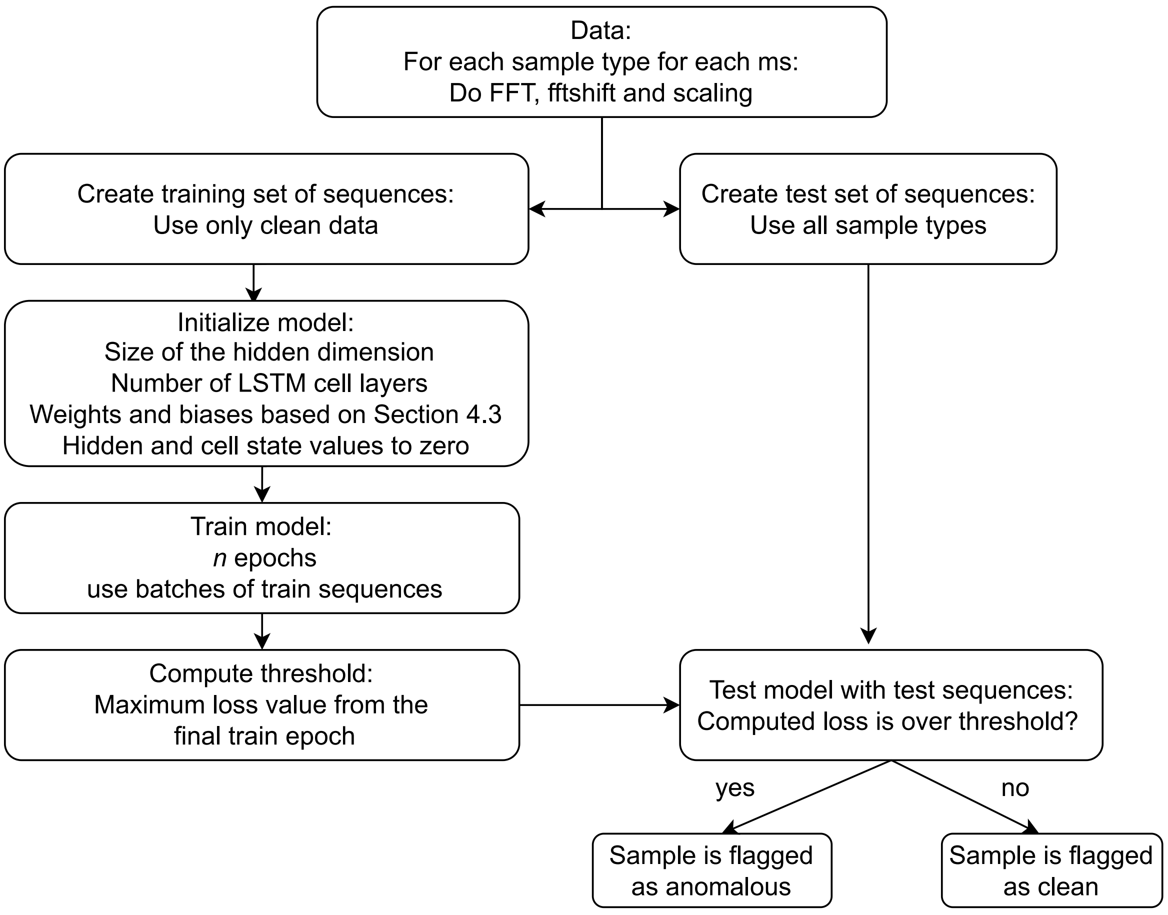

Data:
For each sample type for each ms:
Do FFT, fftshift and scaling
Create training set of sequences:
Use only clean data
Create test set of sequences:
Use all sample types
Initialize model:
Size of the hidden dimension
Number of LSTM cell layers
Weights and biases based on Section 4.3
Hidden and cell state values to zero
Train model:
n epochs
use batches of train sequences
Compute threshold:
Maximum loss value from the
final train epoch
Test model with test sequences:
Computed loss is over threshold?
yes
no
Sample is flagged
as anomalous
Sample is flagged
as clean

Figure 5.6: The workflow of the algorithm. First, the data is read into memory, FFT is calculated, and fftshift is applied for each millisecond for each sample type. In addition, every complex number is scaled. Training set consisting is created by only using clean data, and the test set contains all types of samples. Sequence length – how many milliseconds a sequence consists of – is predefined by the user. Next, the model is initialized with the pre-defined size of the hidden dimensions and the number of stacked LSTM cell layers. Hidden and cell state tensors are initialized to contain zeros, and the other model's parameters are initialized based on Section 4.3. On each epoch, the model is trained with the available data given as batches into the model. The user predefines both batch size and the number of epochs. The threshold value is received by taking the maximum loss value of the validation set from the final training epoch. Finally, the model is tested with the test samples, of which sequence length is the same as used in the training phase. The algorithm can determine whether the tested sample is clean or anomalous by comparing the prediction error and the threshold values.

6. Experiments

PyTorch library [23] version 1.12.1 with CPU is used for the experiments, and while writing this thesis, it was the latest stable release. Unfortunately, CUDA implementation for complex-valued LSTM cells was not yet been implemented. Therefore, the CPU was the only option for training and testing the developed model.

The signal sequences are split into sequences of length 4. Other lengths were tested, but with the number of available clean data samples, this length provided the best results. Each sequence is considered an independent sample. The number of clean train samples in the final tests in the frequency domain is 854, and the number of test samples is 692, of which 144 clean sequences are from the same distribution as the training sequences. The validation set used in the frequency domain contains 64 sample sequences. The learning rate is set to 0.0001 for the complex-valued network in the frequency domain and to 0.001 in all other final model tests. As the tests were done in the earlier phase of the project, a learning rate of 0.01 was also tried. In some cases, this resulted in a sudden jump in loss values after a few epochs, and the loss did not converge as expected. This chapter first presents training in the time domain and in the frequency domain for real and complex values. Thereafter, the results of complex and real-valued models are compared.

6.1 Training in Time Domain

Training and prediction can be made directly in the time domain. This, however, involves a vast parameter space compared to other approaches. Therefore, the input of the model was set to 4 x 4 x 32448 (batches, sequence length, features), and the size of the hidden dimension was set to 32. Now, the training time for each epoch was on average 6 hours, and the prediction time for each sample was 30 seconds. Significantly, the prediction time per sample sequence is too high if fast anomaly detection at an early stage is needed. Due to the time consumption, the number of epochs was reduced to 4, and the reconstruction error did not start to diminish during this time.

Another approach in the time domain was to use only every second frequency feature of a millisecond. Now, the input size (8,4,16224) with the size of the hidden

	Clean	Anomaly
Clean	33	17
Anomaly	195	253

Table 6.1: The best results obtained in time domain experiments. The number of frequency features was reduced to 16224 for each ms, and the number of epochs was set to 10. At the time of these test runs, the larger clean data set was not yet in use. Thus, the high number of correct predicted values may be an effect of overfitting.

dimension being 64. The training epoch time was reduced to an average of 54 minutes and the prediction time was on average 5 seconds for each sample sequence. With this reduced approach, the behavior of the reconstruction error was similar to the approach with all I/Q data. The loss did not start to diminish with a small number of epochs, and on the other hand, if the number of epochs was over ten, the loss began slowly to grow. The confusion matrix for the best model trained with I/Q data can be seen in Table 6.1. For clean data, the prediction was good, with only 17 wrongly classified samples. However, from the anomalous samples, around 44% were misclassified as clean. Moreover, during the time the experiments were done in the time domain, the new clean data file did not exist yet. Hence, the correctly predicted samples may be caused by overfitting.

Another issue with this approach was the amount of memory needed. Data and the training parameters are kept in the memory for the calculations. For the first I/Q approach with all frequency features, the memory utilization was 506 GB, and for the latter one, the utilization was 128 GB. Because of the vast parameter size of the models and the computations needed in order to obtain the reconstruction error, the prediction in environments other than high-performance computers could cause memory issues. Based on the memory usage, the time needed for the prediction, and the prediction capability, these approaches in the time domain are unsuitable for fast and accurate anomaly detection.

6.2 Training in Frequency Domain with Complex values

As mentioned in Section 5.2, the scalar used in scaling seems to affect the training results. At first, the scaling method $scaling_1$ (5.2) was chosen. This scaling yields a situation where the training and prediction results are terrible: all anomalous samples are detected to be clean, and half of the clean samples are classified as anomalies based on the computed threshold value. The solution used for this problem was to apply a

	Clean	Anomaly
Clean	121	73
Anomaly	138	360

Table 6.2: The results of the final complex-valued Autoencoder model. From the misclassified clean sample sequences, 50 samples belonged to clean data from a different distribution leaving 23 samples misclassified from the same distribution. 360 anomalous samples were classified correctly, of which all sample sequences included in Spoofing, MCWI, CWI, and CWI-CI were classified as anomalous. From Pulse, there were only three, and from TMCWI, only one sample was classified as clean.

natural logarithm (3.6) for complex tensors. The resulting complex numbers are the follows: the real part is the magnitude of a complex parameter, and the imaginary part is the phase. This approach provided better results: All anomalies were classified correctly, and 38 out of 50 clean samples were classified as clean. Computing the logarithm changes a complex vector's direction and angle, which can be seen in Figures 3.1a and 3.1b, which can hide the information that a signal could provide.

Moreover, the changed structure of the complex vectors yields fast overfitting. After a few epochs, the model starts to predict the same output regardless of the model's input. This phenomenon can yield good results, particularly for this data set, but fail if a new clean sample sequence from the same distribution is used. The overfitting phenomenon is undesirable because the model works well for only a subset of the data – the used data set. Still, the model's performance would most likely increase with a large amount of data from the same receiver and frequency band.

Because of those issues in the model, new possibilities for scaling were thought of. The second method, $scaling_2$, was developed and tested. This new scaling approach provides better results without any tricks to modify the results returned from the Fast Fourier algorithm. Because the number of training samples is relatively small, increasing the cell layers to more than one results in overfitting. This phenomenon also happens if more than one LSTM layer is used in encoder and decoder units.

However, even though it was thought that the overfitting problem was suppressed with the new scaling method, it turned out that the problem was still there. The first attempts used the clean data type 1 (Figure 5.3), of which the sample file contained a length of 200 ms of data. Because of the lack of clean data, the validation set was not separated, and also, the test set contained samples that were already seen by the model. This was against all the best practices of training and testing deep learning models and, thus, a big mistake. This approach still hid the true problem of the model, namely the overfitting that yielded a situation in which the model could not predict the clean samples not used in the training phase from the same distribution at all as clean.

	Clean	Anomaly
Clean	133	61
Anomaly	94	404

Table 6.3: Final results from the real-valued Autoencoder model. Almost all sample sequences of clean type 2 were flagged as clean; that is, 133 out of 144 samples were correctly classified. All the multipath samples were wrongly flagged as clean, and all the clean samples of type 1 were flagged as an anomaly.

A sample of 1500 ms clean data was created, but this time it came from a slightly different distribution compared to previous clean data (Figure 5.3). At the beginning of the experiments, the FFT length was chosen to be 2048. However, with this length, only the train data started to converge, yielding a situation in which the mean absolute loss value of the validation set stayed high. That is, the model still could not predict the unseen data, although the number of sample sequences was increased. After running several attempts, the FFT length was decreased to 128, the batch size was decreased from 64 to 4, and the number of epochs was increased from 250 to 600. Also, the learning rate was decreased from 0.001 to 0.0001, and the hidden dimension was set to 96 instead of 64. These adjustments provided the results seen in Table 6.2 and in Figure 6.1. To compare, the model with the same hyperparameter values was also trained with default weights drawn from the uniform distribution. The results were slightly worse, having ten more anomalous samples flagged as clean. The final model's user-defined parameters can be found in Table 6.6. With the proper values, the model still seems slightly overfit but keeps the ability to adjust itself based on the input. More discussion about the results is found in 6.4.

6.3 Training in Frequency Domain with Real Values

The same autoencoder structure was used for experiments in the real domain to make the comparison. In this case, the default weights provided by PyTorch were used, and the model parameter space was changed to use floats instead of complex numbers. In the data preparation phase, a base ten logarithm is taken, and the values are multiplied by a scalar ten to obtain the dB scale before scaling the values with a min-max scaler. The same criterion chose the threshold as in the complex domain, which is the maximum validation loss value from the final training epoch.

At the very beginning, it was seen that the small amount of training data caused the model to overfit in a manner that starts to predict the same output for each

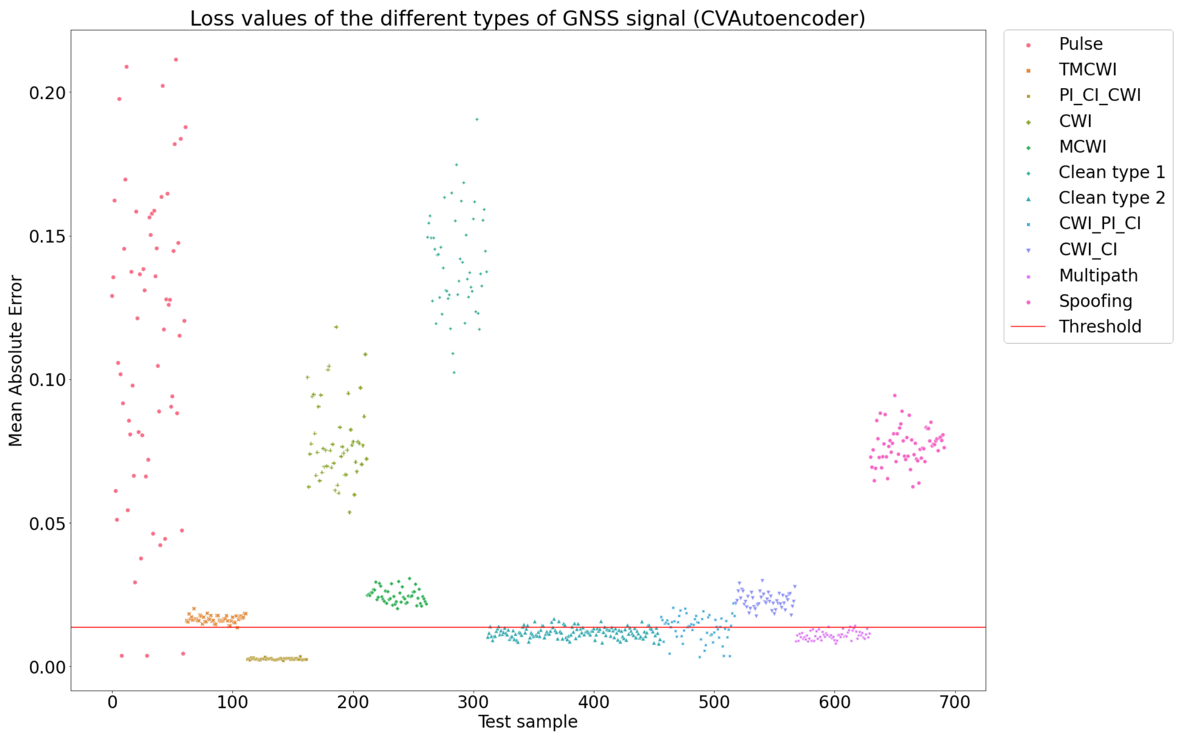


Figure 6.1: The distribution of the loss values for complex-valued autoencoder. The computed threshold is marked as a red horizontal line. To emphasize, the model cannot predict clean data from a different distribution as clean, but the model must be trained and used for the (clean) data that comes from the same distribution, e.g. have the same frequency band occupation. The model has the most difficulties with the samples containing more than one anomaly type but has the capability to label the samples containing only one interference type almost with 100% accuracy as anomalous. The only exception is the multipath signal: only three samples were classified as an anomaly.

sample regardless of the given input. This phenomenon was partly solved when the new generated clean sample file was used, and the length of the FFT algorithm was reduced. Also, reducing the number of hidden dimensions, batch size, and epochs improved the prediction performance of the real-valued autoencoder. As a higher number of epochs were tested, the model's capability to predict clean type 2 samples as clean was improved. However, at the same time, more anomalous samples were flagged wrongly as clean. The results from the final run are seen in Table 6.3 and 6.2 and are discussed in detail in the next section.

6.4 Discussion of the Results

In terms of anomaly detection accuracy (Table 6.1), and in terms of computational time and the used memory (Table 6.7), the models tested in the time domain can be discarded immediately. If only the numbers are looked at, the model trained in the real domain performs better in terms of the number of epochs needed to get reasonable

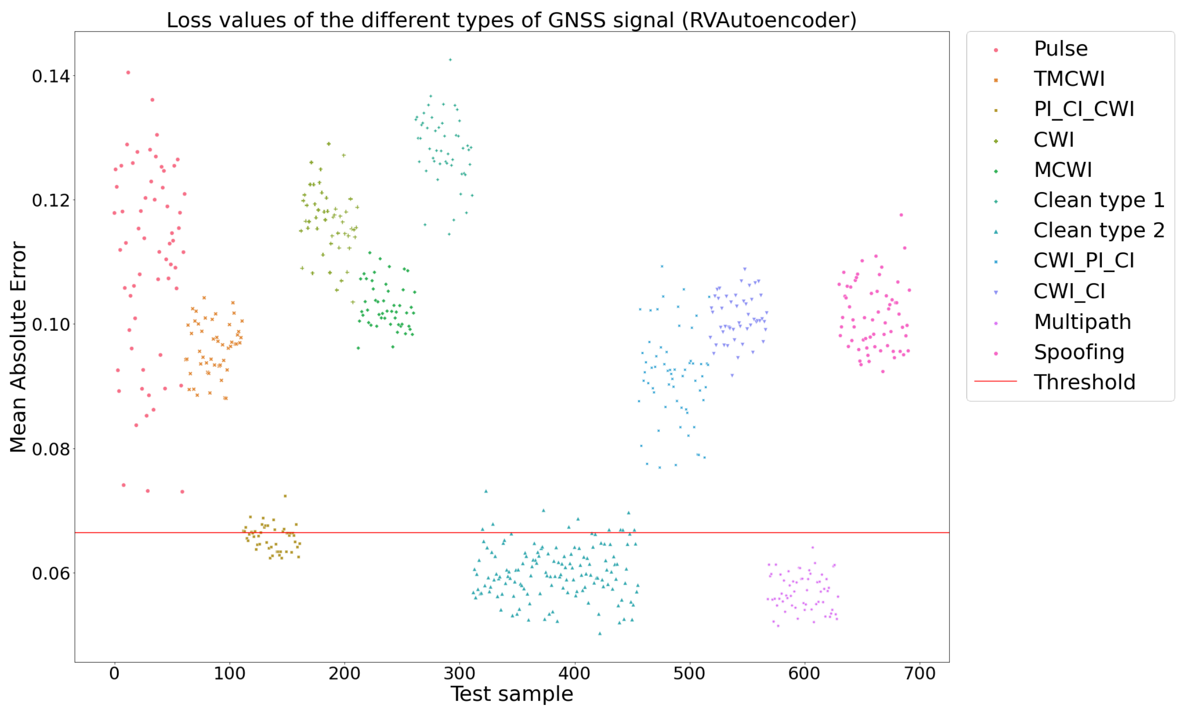


Figure 6.2: Real-valued Autoencoder: The distribution of the loss values received from the prediction. The computed threshold is marked as a red horizontal line. In the real domain, almost all anomaly sample groups were classified correctly as anomalous. However, samples from Pulse-Chirp-Continuous interference were partly flagged as clean, and all the multipath samples were classified as clean.

results. Also, the number of samples classified correctly (Table 6.3) is higher than it is with a complex-valued model in the frequency domain (Table 6.2). The prediction time is, in both cases, around 1 millisecond, and the training time for each epoch is 3.09 seconds for complex-valued and 1.31 seconds for real-valued autoencoder. Thus, in terms of overall time, the real-valued training is clearly better, but when examining the epoch level and sample-specific time duration, the models are reminiscent. The slight increase in time is due to the complex multiplication and slightly larger weight and bias tensors because of the used parameter value for the hidden dimension.

The user-defined parameters for real and complex-valued models are shown in Table 6.6. Only the hidden-dim parameter differs between the models. During the test runs, it came visible that a smaller hidden dimension size improved the real-valued model, whereas a higher one improved the complex-valued model. This resulted in all weight and bias tensors in the complex-valued encoder and the input-hidden weight tensor in the complex-valued decoder being bigger than the tensors used in the real domain (Tables 6.4 and 6.5). The utilized memory during the loading of the data, creating all three types of sequence sets, and training was for the real-valued model 5.29 GB and for the complex-valued model 5.30 GB. Even though only the training and validation tests were used to get the memory utilization numbers in the frequency domain, the test set was also loaded and kept in memory to see more clearly, do the complex numbers affect memory utilization. The utilization for the complex-valued model is slightly higher (Table 6.7). However, this can be explained by the size of the tensors used in the complex-valued Autoencoder. Also, the representation of the values may have an effect: the complex values are presented as a type of complex128, which means that both real and imaginary units are presented as float64, which is also the presentation of the values in real-valued autoencoder.

The prediction capability of the models was examined with common metrics: accuracy, precision, recall, and F1-score, the harmonic mean of precision and recall. The formulas of these metrics are the following [27]

$$\begin{aligned}\text{Accuracy } &= \frac{TP+TN}{TP+TN+FP+FN}\\ \text{Precision } &= \frac{TP}{TP+FP}\\ \text{Recall } &= \frac{TP}{TP+FN}\\ \text{F1-score } &= 2\frac{Precision \cdot Recall}{Precision + Recall}\end{aligned}$$

where TP, FP, TN, and FN represent the values of true positives, false positives, true negatives, and false negatives, respectively. The metrics of both real and complex-valued models are presented in Table 6.8. It is clearly visible that the real-valued model outperforms the complex-valued one in terms of accuracy and other measurement metrics. I left out the clean samples of type 1 as computing the accuracy metrics. This decision was done because the samples were from a different distribution and expected to be flagged as anomalous. Nevertheless, except for the precision values and the real-valued model's accuracy, the measurement metrics are below 80 percent. Based on the results, neither of the trained models is suitable for accurate anomaly detection.

The computed metrics indicate that the model architecture could be improved and that there are issues with the data. If, for example, Figures A.1b and A.6e are examined more carefully, we can see a few spikes with high real or imaginary values in the central region. As the complex vectors are scaled, the values significantly smaller than in spikes get close to zero. This means that during the training, the tensor consisting mostly of values of near-zero is multiplied with weight tensors. Thus, the resulting output tensors are also consisting mostly of near-zero values producing small reconstruction errors and hence misclassifications. Thus, to improve the prediction capability a possibility of a new scaling method must be considered.

As mentioned earlier, the validation set in the complex-valued model did not start to converge during the training, which indicated the overfitting of the model. To

Parameter type	LSTM (encoder)	LSTM (decoder)	Linear (decoder)
Input-hidden weights	4 x 32 x 128	4 x 128 x 32	-
Hidden-hidden weights	4 x 32 x 32	4 x 128 x 128	-
Input-hidden bias	4 x 32	4 x 128	-
Hidden-hidden bias	4 x 32	4 x 128	-
Weights	-	-	128 x 128
Bias	-	-	128

Table 6.4: The parameter space of the final model for the real-valued autoencoder. For the LSTM, the weights and biases are initialized for all three gates and $\hat{c}$ independently. Here, 32 is the hidden dimension parameter, which is the features out from the encoder and the number of features in the decoder. 128 is the number of features in and features out of the autoencoder.

overcome this problem, the length of the FFT was reduced from 2048 to 128. This reduction was done step-wise so that FFT lengths 1024 and 256 were also tested, but either the validation set did not converge, or the results were bad. However, the resolution of the computed result is lower with a smaller FFT length, and thus some information may get lost. While some values are dropped out due to shorter FFT length, some fine learnable patterns may diminish, which can affect the prediction accuracy negatively. Moreover, when the learning curves of both models are examined (Figure 6.3) we can see that the behavior of the loss function curves for the validation set is not what is expected. That is, the validation loss curve follows the loss curve of the training set closely for 50 first epochs, but after this, there is a cap of 0.01 units between the loss values. This behavior indicates that the model's generalization capability could be improved.

To summarize the discussion of the results, we can say that the real-valued model outperforms the complex-valued one by the measured metrics. Both models are even when comparing the epoch and prediction times and the utilized memory. However, the complex-valued autoencoder was able to classify 2 multipath samples as anomalous, whereas the real-valued autoencoder claimed all of those samples as clean. Also, the loss curve of the complex-valued autoencoder indicates that there is still a problem with the amount of data or the model may need changes in the architecture.

Parameter type	LSTM (encoder)	LSTM (decoder)	Linear (decoder)
Input-hidden weights	4 x 96 x 128	4 x 128 x 96	-
Hidden-hidden weights	4 x 96 x 96	4 x 128 x 128	-
Input-hidden bias	4 x 96	4 x 128	-
Hidden-hidden bias	4 x 96	4 x 128	-
Weights	-	-	128 x 128
Bias	-	-	128

Table 6.5: The parameter space of the final model in the frequency domain with a complex-valued autoencoder. For the LSTM, the weights and biases are initialized for all three gates and $\hat{c}$ independently. Here, 96 is the hidden dimension parameter, which is the features out from the encoder and the number of features in the decoder. 128 is the number of features in and features out of the autoencoder.

Model	Batch size	Hidden dim	Cell layers	Sequence length
CVAutoencoder	4	96	1	4
RVAutoencoder	4	32	1	4

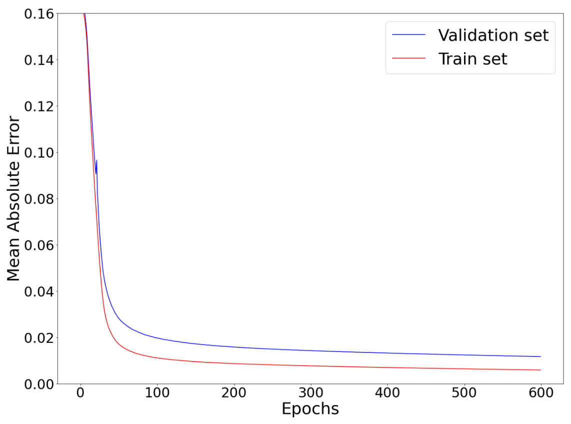
Table 6.6: Final models both in real and complex domain. Both models have otherwise the same parameter settings, but the hidden dim was set differently. Whilst the lower hidden dim reduced the overfitting phenomena in the real-valued model, the increased hidden dimension value was needed in the complex domain to predict the unseen clean data correctly.

Model	Epochs used	Epoch time (avg)	Prediction time (avg)	Memory Utilized
CVAutoencoder	600	3.09 s	0.0015 s	5.30 GB
RVAutoencoder	15	1.314 s	0.0007 s	5.29 GB
CVAutoencoder IQ/2	10	54 min	5 s	128 GB
CVAutoencoder IQ	4	370 min	30 s	506 GB

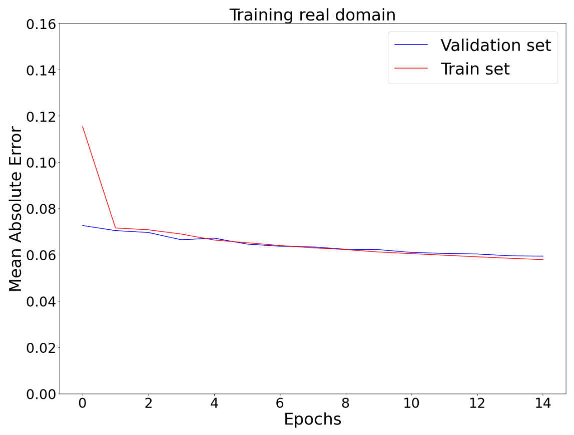
Table 6.7: The complex-valued autoencoders in the time domain had the worst performance. Here the model CVAutoencoder IQ is the model trained with the feature size of 32448, and the model CVAutoencoder IQ/2 represents the model in which the previous feature size was reduced to half. The values in all the columns except the number of epochs are manifold compared to models used in the frequency domain. The number of epochs used in the complex-valued CVAutoencoder was 40 times bigger than the number of epochs used in training in the real-valued RVAutoencoder. However, the time used for each epoch and predicted sample is comparable and close to each other. The memory utilized for creating the datasets and training the models is very close. The difference can be explained with slightly bigger parameter space for CVAutoencoder and the fact that numbers were of type complex128, meaning that both real and imaginary units are the type of float64. In RVAutoencoder, the numbers were the type of float64.

Model	Accuracy	Precision	Recall	F1-score
CVAutoencoder	0.749	0.841	0.467	0.600
RVAutoencoder	0.836	0.924	0.514	0.717

Table 6.8: Accuracy metrics for complex-valuer and real-valued autoencoders. Here the clean samples of type 1 were left out of the calculations. In all cases, the real-valued model outperforms the complex-valued one. When looking at the numbers, we can see that the 90 % mark is crossed only in one metric in the real-valued model. In both domains, most of the metrics are low. That is, neither of the models can accurately detect anomalies.



(a) The loss curve in the complex domain.



(b) The loss curve in the real domain.

Figure 6.3: The final loss curves from the experiments run in the frequency domain. The loss values in the real-valued autoencoder stay high compared to the complex-valued autoencoder. However, the losses from these two types of autoencoders are not comparable to each other. This is because of how the data is prepared for the model to use. The behavior of the loss, though, is comparable. The wanted behavior is seen in (b). That is, the loss of the validation set is close to the training set. Based on the loss of the training set, we can see that the model is still overfitting, even though both lines are converging.

7. Conclusions

In this thesis, a fully complex-valued autoencoder was implemented to detect anomalies in GNSS signals. The current libraries, such as PyTorch, do not provide all the functionalities in the complex domain, and those existing functionalities do not provide CUDA compatibility that is required if GPU is used. This set limitations of possible usage of functionalities without making changes to the existing code base. This is about to change slowly, and new releases contain more functionalities that can be applied to complex-valued network architectures.

The layers and activation and loss functions used in the model's architecture have full support for complex numbers if operated on CPUs. The developed model was tested in both the time and frequency domains, and the frequency domain was further divided into two separate test domains: real and complex-valued. It was shown that the time domain is unsuitable for fast and effective anomaly detection in GNSS signals because of the computation time in the prediction phase and the high memory utilization.

The experiments done in the frequency domain show that with this amount of data and the model's architecture, the real-valued model performs better in terms of accuracy metrics. It also needs fewer epochs compared to its complex-valued counterpart. The complex-valued autoencoder needs approximately 1 ms more time in prediction for each sample and 1.5 seconds more time for each epoch. On the other hand, these differences are not significant. Furthermore, for the complex-valued autoencoder, the hidden dimension was set to 96, whereas the value of the same parameter was set to 32 in the real-valued counterpart. This resulted in larger weight and bias tensors, especially in the encoder unit of the complex-valued autoencoder compared to the autoencoder used in the real domain. Nevertheless, the difference between the utilized memory for this data set during the data preparation and training was only 0.01 GB.

The numbers in the previous paragraph show that the real-valued autoencoder outperforms the complex-valued counterpart at this stage of development. On the other hand, from the produced loss curves, we can see that some overfitting is still happening in the autoencoder used in the complex domain. Another current issue is the length of FFT. As the size was reduced from 2048 to 128, the concern of losing information due to the decreasing resolution of FFT was raised. However, this was the only possibility to make the model recognize unseen data and detect it as clean. In future work, more clean data is needed for the training to see whether the FFT length could be decreased while keeping the model recognizing the unseen clean data. Especially if decreasing the FFT length is not possible, a new scaling method must be developed to overcome the issue that some of the anomalous data are getting significantly small reconstruction errors regardless of the trained model's parameters.

As said, one of the biggest problems throughout the project was the limited number of clean samples. After introducing the newly generated clean data set and reducing the FFT length, an improvement in the model's learning capability was visible. Before the longer clean data sample file, the overfitting prevented the model predict the unseen clean data as clean. On the other hand, generating data with the simulator is easy, but collecting real-world data and filtering out anomalous samples before training is time-consuming. Thus, there can always be a lack of training data. In future research, the architecture of the model can be further improved. For example, a 1-dimensional convolutional layer could extract the frequency patterns familiar for clean data better than using only LSTM layers.

Based on the results of this work, the complex-valued autoencoder used in anomaly detection in GNSS signals needs to be improved to outperform the real-valued one. In the complex domain, decreasing the learning rate from 0.001 to 0.0001 provided better results; however, the number of training epochs needed to be increased because of the minor update per each epoch. An adaptive learning rate could reduce the number of used epochs. Also, more research is needed to see if the learning rate could be complex-valued.

The most important research question: "Is anomaly detection improved by using deep learning networks directly in the complex domain?" cannot be answered based on the results and faults found in the architecture and the scaling method. Nevertheless, the experiments show that using a fully complex-valued autoencoder for anomaly detection is possible and can provide good results if the issues with architecture and data can be fixed.

Bibliography

[1] *Resilient Positioning, Navigation, and Timing for Critical Infrastructure. U.S Department of Homeland Security.* https://www.dhs.gov/publication/st-resilient-pnt-for-critical-infrastructure-fact-sheet. Accessed: 4.7.2022.

[2] J. Bassey, L. Qian, and X. Li. *A Survey of Complex-Valued Neural Networks.* 2021. DOI: 10.48550/ARXIV.2101.12249. URL: https://arxiv.org/abs/2101.12249.

[3] S. Bhamidipati, K. J. Kim, H. Sun, and P. V. Orlik. "GPS Spoofing Detection and Mitigation in PMUs using Distributed Multiple Directional Antennas". In: *Proceedings of the ICC 2019 - 2019 IEEE International Conference on Communications (ICC).* 2019, pp. 1–7. DOI: 10.1109/ICC.2019.8761208.

[4] O. E. Brigham. *The fast Fourier transform.* Prentice-Hall, Inc., Engelwood Cliffs, N.J., 1974. ISBN: 0-13-307496-X.

[5] R. Calvo-Palomino, A. Bhattacharya, G. Bovet, and D. Giustiniano. "Short: LSTM-based GNSS Spoofing Detection Using Low-cost Spectrum Sensors". In: *Proceedings of the 2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM).* 2020, pp. 273–276. DOI: 10.1109/WoWMoM49955.2020.00055.

[6] F. Dovis. *GNSS Interference Threats and Countermeasures.* Artech House, 2015. ISBN: 978-1-60807-810-3.

[7] A. Elango, S. Ujan, and L. Ruotsalainen. "Disruptive GNSS Signal detection and classification at different Power levels Using Advanced Deep-Learning Approach". In: *Proceedings of the 2022 International Conference on Localization and GNSS (ICL-GNSS).* 2022, pp. 1–7. DOI: 10.1109/ICL-GNSS54081.2022.9797026.

[8] Z. Fang, G. Zhang, Q. Dai, and B. Xue. "PolSAR Image Classification Based on Complex-Valued Convolutional Long Short-Term Memory Network". In: *IEEE Geoscience and Remote Sensing Letters* 19 (2022), pp. 1–5. DOI: 10.1109/LGRS.2022.3146928.

[9] R. F. Fischer. “Appendix A: Wirtinger Calculus”. In: *Precoding and Signal Shaping for Digital Transmission*. John Wiley & Sons, Ltd, 2002, pp. 405–413. ISBN: 9780471439004. DOI: https://doi.org/10.1002/0471439002.app1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471439002.app1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471439002.app1.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[11] R. Gu, S.-X. Zhang, Y. Zou, and D. Yu. “Complex Neural Spatial Filter: Enhancing Multi-Channel Target Speech Separation in Complex Domain”. In: *IEEE Signal Processing Letters* 28 (2021), pp. 1370–1374. DOI: 10.1109/LSP.2021.3076374.

[12] L. Gunn, P. Smet, E. Arbon, and M. D. McDonnell. “Anomaly Detection in Satellite Communications Systems using LSTM Networks”. In: *Proceedings of the 2018 Military Communications and Information Systems Conference (MilCIS)*. 2018, pp. 1–6. DOI: 10.1109/MilCIS.2018.8574109.

[13] E. Kaplan and C. Hegarty. *Understanding GPS/GNSS Principles and Applications*. Artech House, 2017. Chap. GNSS Disruptions Authors.

[14] D. R. Kartchner, R. Palmer, and S. K. Jayaweera. “Satellite Navigation Anti-Spoofing Using Deep Learning on a Receiver Network”. In: *Proceedings of the 2021 IEEE Cognitive Communications for Aerospace Applications Workshop (CCAAW)*. 2021, pp. 1–5. DOI: 10.1109/CCAAW50069.2021.9527295.

[15] T. Kim and A. Tülay. “Approximation by Fully Complex Multilayer Perceptrons”. In: *Neural Computation* 15.7 (2003), pp. 1641–1666. DOI: 10.1162/089976603321891846.

[16] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[17] S. Lebrun, S. Kaloustian, R. Rollier, and C. Barschel. “GNSS Positioning Security: Automatic Anomaly Detection on Reference Stations”. In: *Proceedings of the Critical Information Infrastructures Security*. Ed. by D. Percia David, A. Mermoud, and T. Maillart. Cham: Springer International Publishing, 2021, pp. 60–76. ISBN: 978-3-030-93200-8.

[18] I. E. Mehr and F. Dovis. “Detection and Classification of GNSS Jammers Using Convolutional Neural Networks”. In: *Proceedings of the 2022 International Conference on Localization and GNSS (ICL-GNSS)*. 2022, pp. 01–06. DOI: 10.1109/ICL-GNSS54081.2022.9797030.

[19] X. Miao, J. Zhao, and Y. Qiao. "Research on Application of ARIMA Clock Error Prediction Algorithm in Satellite Anti-spoofing". In: *Proceedings of the 2021 IEEE/CIC International Conference on Communications in China (ICCC)*. 2021, pp. 990–994. DOI: 10.1109/ICCC52777.2021.9580412.

[20] A. Minin, A. Knoll, and H.-G. Zimmermann. "Complex Valued Recurrent Neural Network: From Architecture to Training". In: *Journal of Signal and Information Processing* 03 (Jan. 2012), pp. 192–197. DOI: 10.4236/jsip.2012.32026.

[21] J. Muir and J. j. Muir. *Complex Analysis : A Modern First Course in Function Theory*. John Wiley & Sons, Incorporated, 2015.

[22] E. Munin, A. Blais, and N. Couellan. "Convolutional Neural Network for Multipath Detection in GNSS Receivers". In: *Proceedings of the 2020 International Conference on Artificial Intelligence and Data Analytics for Air Transportation (AIDA-AT)*. 2020, pp. 1–10. DOI: 10.1109/AIDA-AT48540.2020.9049188.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[24] P. Patel, M. Nandu, and P. Raut. "Initialization of Weights in Neural Networks". In: *International Journal of Scientific Development and Research* Volume 4 | Issue 2 (Feb. 2019), pp. 73–79.

[25] M. Poutanen. *Satelliittipaikannus*. Tähtitieteellinen yhdistys Ursa ry, 2016, pp. 18–29. ISBN: 978-952-5985-41-2.

[26] L. Ruotsalainen, M. Z. H. Bhuiyan, S. Thombre, S. Söderholm, and H. Kuusniemi. "Impact of cheap commercial jammer on BeiDou signal". In: *Coordinates* X (Nov. 2014), pp. 22–28.

[27] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2016. ISBN: 9781292153964.

[28] S. Scardapane, S. Van Vaerenbergh, A. Hussain, and A. Uncini. "Complex-Valued Neural Networks With Nonparametric Activation Functions". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 4.2 (2020), pp. 140–150. DOI: 10.1109/TETCI.2018.2872600.

[29] S. Semanjski, A. Muls, I. Semanjski, and W. De Wilde. "Use and Validation of Supervised Machine Learning Approach for Detection of GNSS Signal Spoofing". In: *Proceedings of the 2019 International Conference on Localization and GNSS (ICL-GNSS)*. 2019, pp. 1–6. DOI: 10.1109/ICL-GNSS.2019.8752775.

[30] N. Spens, D.-K. Lee, F. Nedelkov, and D. Akos. "Detecting GNSS Jamming and Spoofing on Android Devices". In: *NAVIGATION: Journal of the Institute of Navigation* 69.3 (2022). ISSN: 0028-1522. DOI: 10.33012/navi.537. eprint: https://navi.ion.org/content/69/3/navi.537.full.pdf. URL: https://navi.ion.org/content/69/3/navi.537.

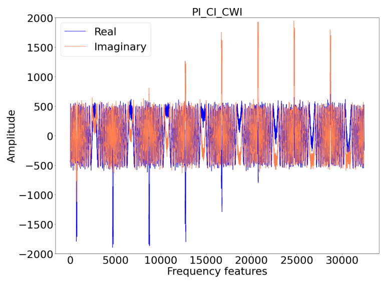
[31] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal. *Deep Complex Networks*. 2017. DOI: 10.48550/ARXIV.1705.09792. URL: https://arxiv.org/abs/1705.09792.

[32] Y. Wei, J. Jang-Jaccard, W. Xu, F. Sabrina, S. Camtepe, and M. Boulic. *LSTM-Autoencoder based Anomaly Detection for Indoor Air Quality Time Series Data*. 2022. DOI: 10.48550/ARXIV.2204.06701. URL: https://arxiv.org/abs/2204.06701.

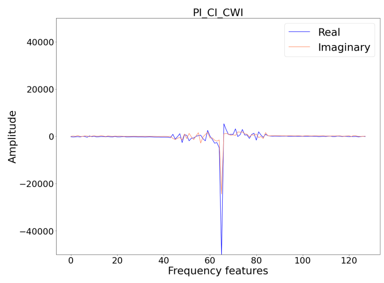
[33] Z. Wu, Y. Zhang, Y. Yang, C. Liang, and R. Liu. "Spoofing and Anti-Spoofing Technologies of Global Navigation Satellite System: A Survey". In: *IEEE Access* 8 (2020), pp. 165444–165496. DOI: 10.1109/ACCESS.2020.3022294.

Appendix A. Data

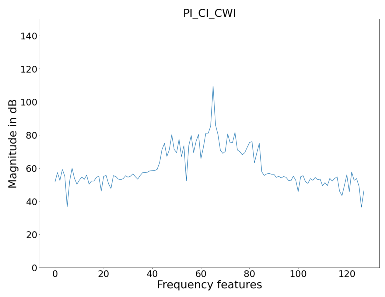
In this appendix chapter, the rest of the data is illustrated to give the reader the possibility to get familiar with the different kinds of interference types used in this thesis. There are small fluctuations between each millisecond because of the noise, but they resemble each other closely. However, to see the differences I have added here two samples of length 1 ms. The samples that are not shown in the main text are presented in Figures A.1, A.2, A.3, A.4, A.5 and A.6. For the samples shown in the main text, data for the second millisecond is illustrated in Figures A.7, A.8, A.9, A.10, and A.11.



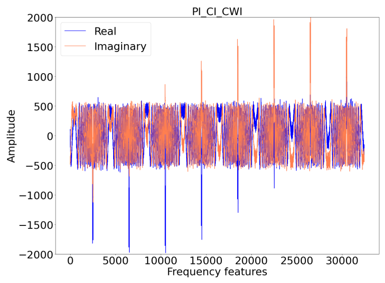
(a) I/Q data for the first ms.



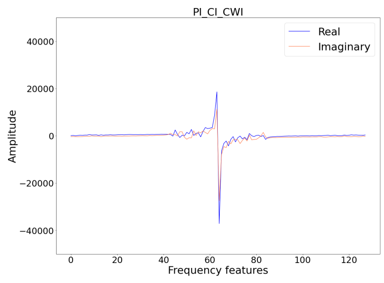
(b) Data in the complex domain.



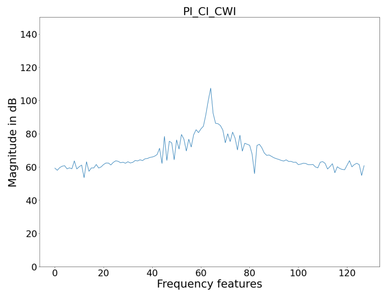
(c) Data in the real domain.



(d) I/Q data for the second ms.

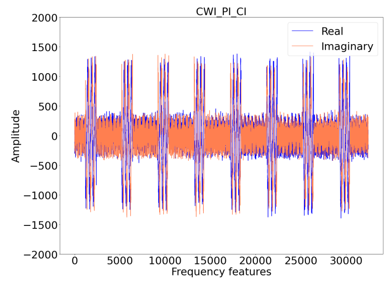


(e) Data in the complex domain.

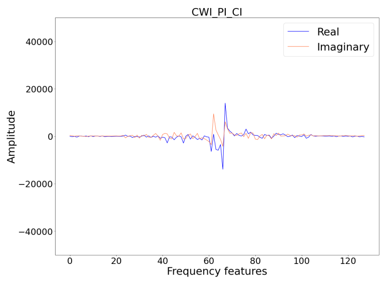


(f) Data in the real domain.

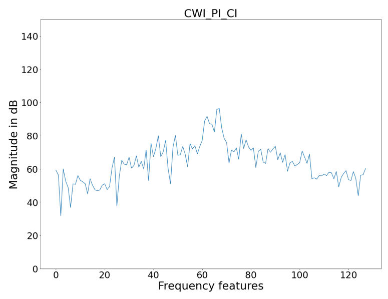
Figure A.1: Examples of Pulse-Chirp-Continuous wave interference in I/Q-format, after computing FFT, and after transforming the data representation into magnitude in dB. All samples in the complex domain and some of the samples in the real domain were classified as clean. When examining the figures we can see that the shape of data in the real domain (c) resembles the shape of the clean data type 2, which can confuse the model to predict wrongly. In the complex domain, the values are scaled with the maximum value of either the absolute value of a real or imaginary unit, depending on which one is bigger. In this case, all the values around the spikes are scaled to be close to zero. This means that because of the input tensor, the outputs of the cell computations producing tensors close to zero, yielding the reconstruction error to be close to zero.



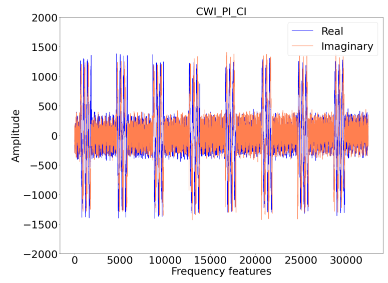
(a) I/Q data for the first ms.



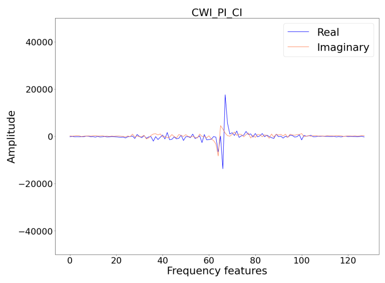
(b) Data in the complex domain.



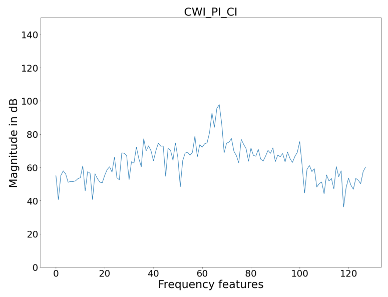
(c) Data in the real domain.



(d) I/Q data for the second ms.

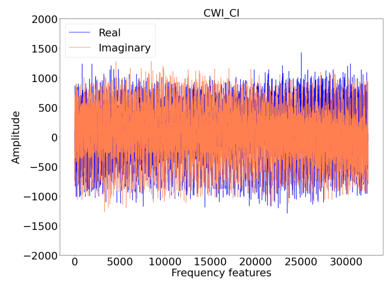


(e) Data in the complex domain.

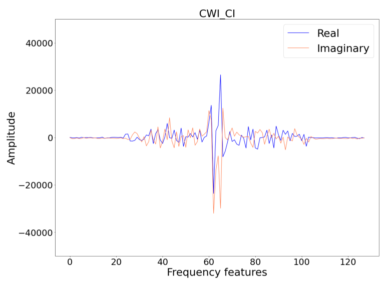


(f) Data in the real domain.

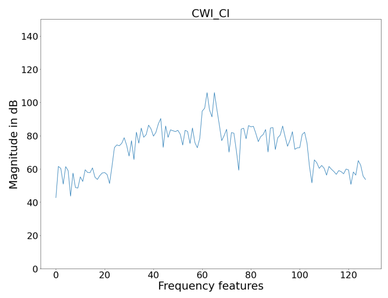
Figure A.2: Examples of Continuous wave-Pulse-Chirp-interference (a, d) in I/Q-format, (b, e) after computing FFT, and (c, f) after transforming the data representation into magnitude in dB.



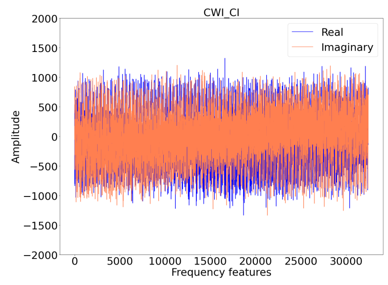
(a) I/Q data for the first ms.



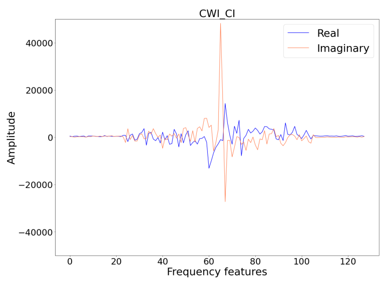
(b) Data in the complex domain.



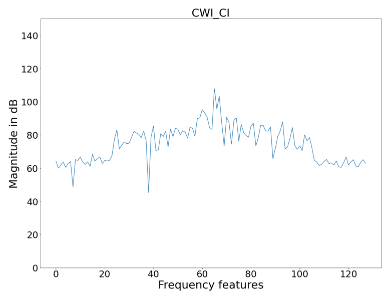
(c) Data in the real domain.



(d) I/Q data for the second ms.

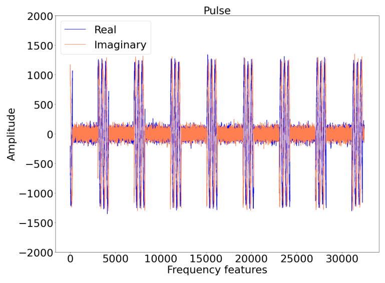


(e) Data in the complex domain.

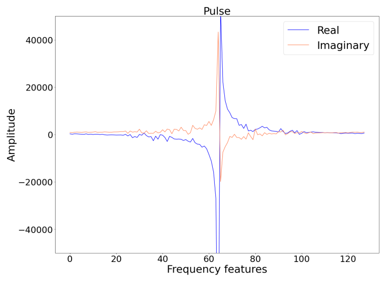


(f) Data in the real domain.

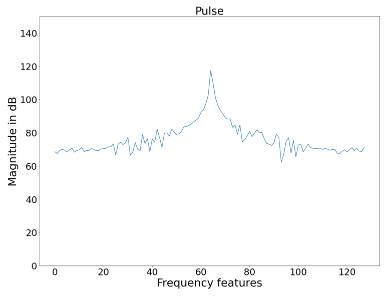
Figure A.3: Examples of Continuous wave-Chirp interference (a, d) in I/Q-format, (b, e) after computing FFT, and (c, f) after transforming the data representation into magnitude in dB.



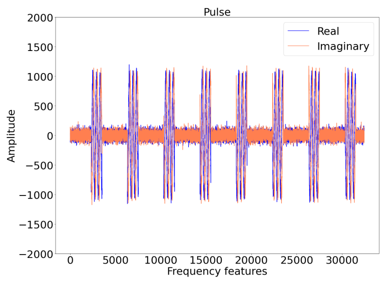
(a) I/Q data for the first ms.



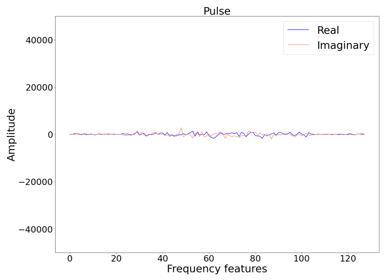
(b) Data in the complex domain.



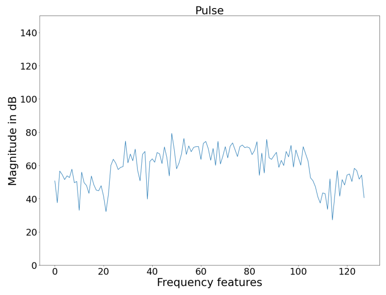
(c) Data in the real domain.



(d) I/Q data for the second ms.

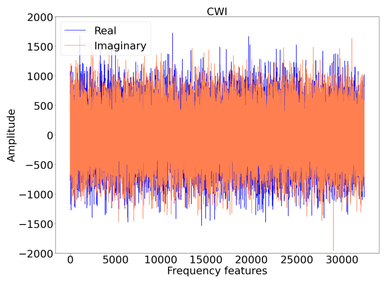


(e) Data in the complex domain.

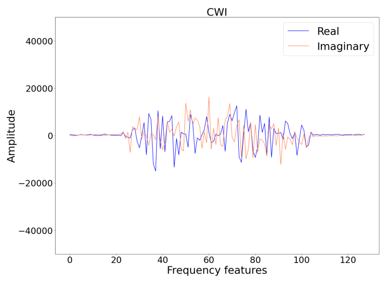


(f) Data in the real domain.

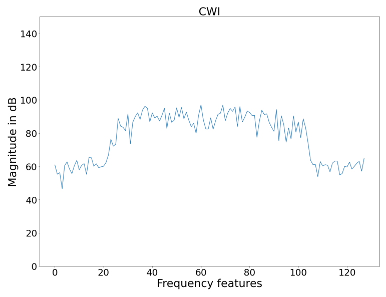
Figure A.4: Examples of pulse jamming interference (a, d) in I/Q-format, (b, e) after computing FFT, and (c, f) after transforming the data representation into magnitude in dB.



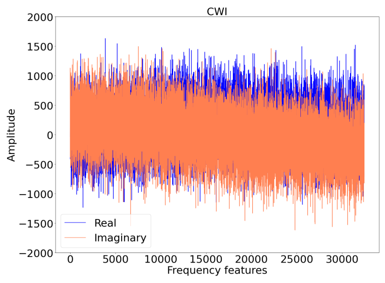
(a) I/Q data for the first ms.



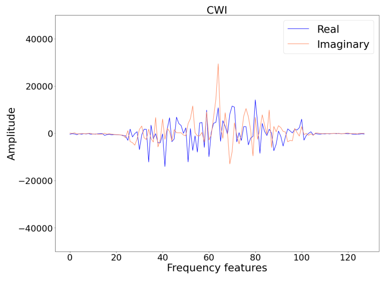
(b) Data in the complex domain.



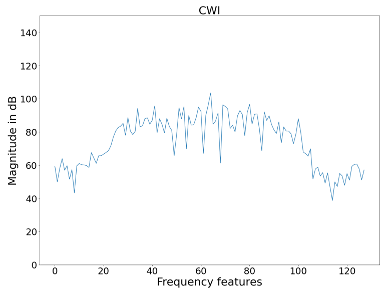
(c) Data in the real domain.



(d) I/Q data for the second ms.

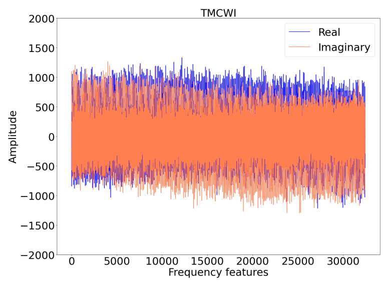


(e) Data in the complex domain.

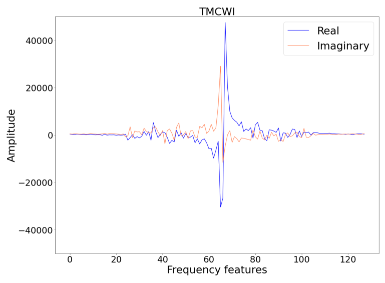


(f) Data in the real domain.

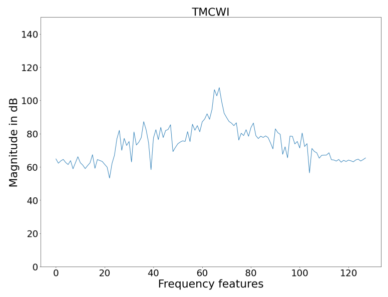
Figure A.5: Examples of Continuous wave interference (a, d) in I/Q-format, (b, e) after computing FFT, and (c, f) after transforming the data representation into magnitude in dB.



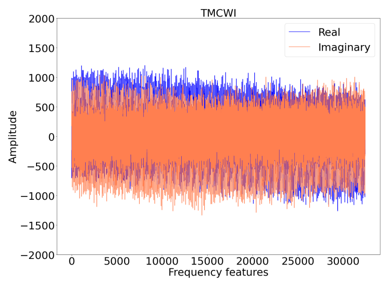
(a) I/Q data for the first ms.



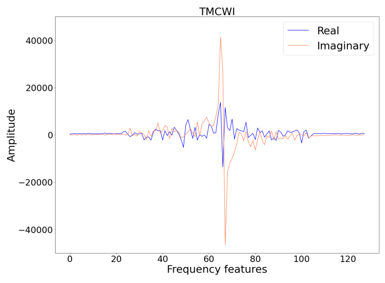
(b) Data in the complex domain.



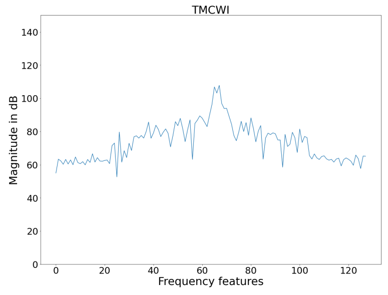
(c) Data in the real domain.



(d) I/Q data for the second ms.

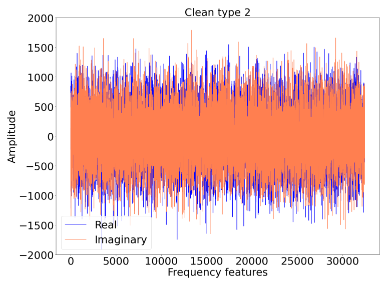


(e) Data in the complex domain.

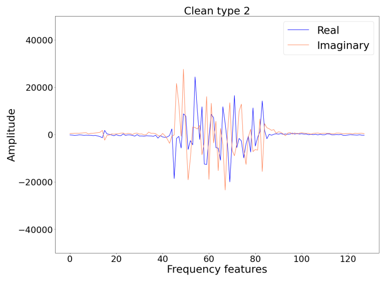


(f) Data in the real domain.

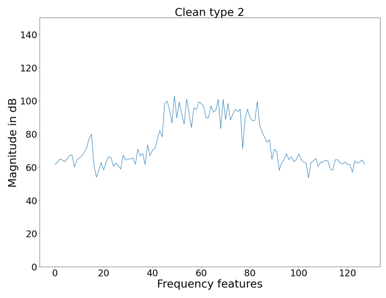
Figure A.6: Examples of continuous wave interference with three interfering tone frequencies (a, d) in I/Q-format, (b, e) after computing FFT, and (c, f) after transforming the data representation into magnitude in dB.



(a) I/Q data for the second ms.

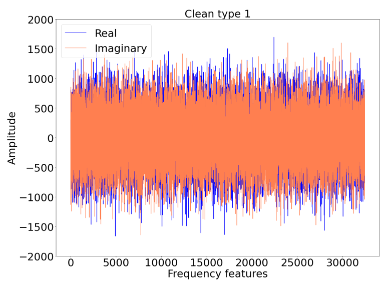


(b) Data in the complex domain.

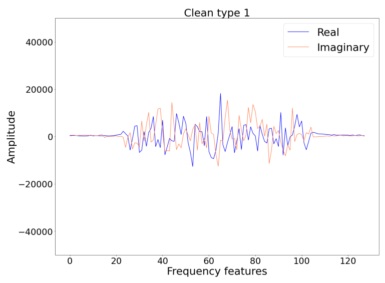


(c) Data in the real domain.

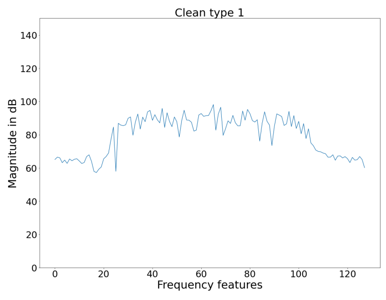
Figure A.7: Example of clean type 2 for the second millisecond of data in (a) I/Q-format, (b) after computing FFT, (c) and after transforming the data representation into magnitude in dB. This clean signal was used in training



(a) I/Q data for the second ms.

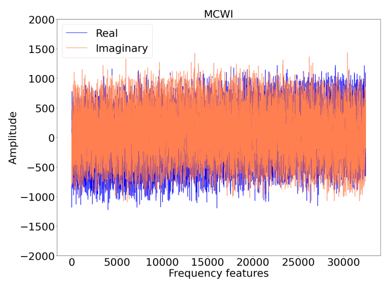


(b) Data in the complex domain.

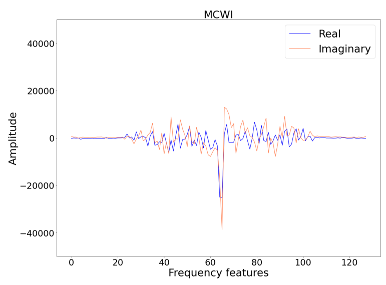


(c) Data in the real domain.

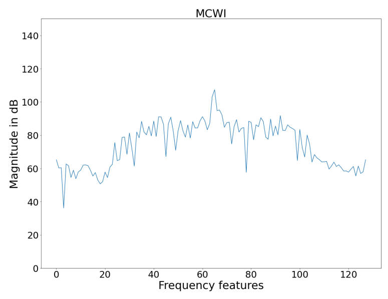
Figure A.8: Example of clean type 1 for the second millisecond of data (a) in I/Q-format, (b) after computing FFT, and (c) after transforming the data representation into magnitude in dB.



(a) I/Q data for the second ms.

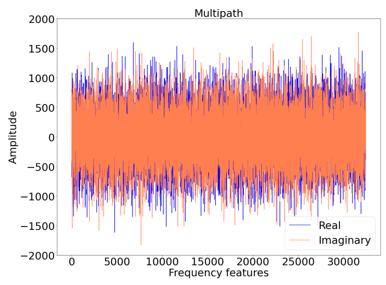


(b) Data in the complex domain.

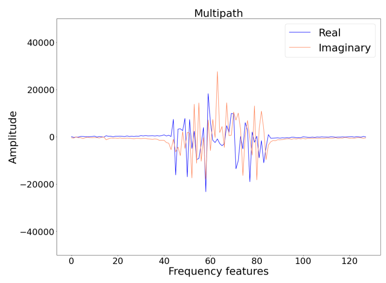


(c) Data in the real domain.

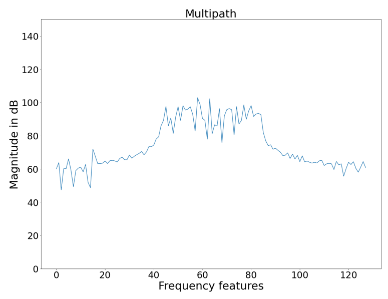
Figure A.9: Example of continuous wave interference with two interfering tone frequencies for the second millisecond of data (a) in I/Q-format, (b) after computing FFT, and (c) after transforming the data representation into magnitude in dB.



(a) I/Q data for the second ms.

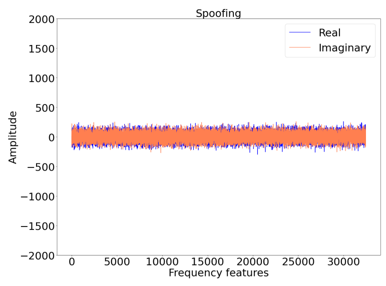


(b) Data in the complex domain.

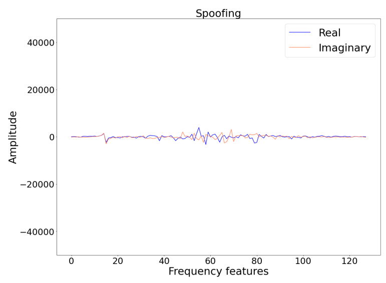


(c) Data in the real domain.

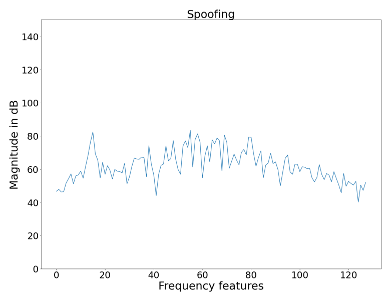
Figure A.10: Example of multipath interference for the second millisecond of data (a) in I/Q-format, (b) after computing FFT, and (c) after transforming the data representation into magnitude in dB.



(a) I/Q data for the second ms.



(b) Data in the complex domain.



(c) Data in the real domain.

Figure A.11: Example of spoofing interference for the second millisecond of data (a) in I/Q-format, (b) after computing FFT, and (c) after transforming the data representation into magnitude in dB.