UNIVERSIDADE D
COIMBRA

Diogo Gonçalo dos Santos Henriques

Magnitude Modulated Signals Detection with Neural Networks

Dissertation in the context of the Master in Electrical and Computer Engineering, Specialization in Computers, Subspecialization in Computational Learning, supervised by Prof. Dr. Marco Alexandre Cravo Gomes and Prof. Dr. Vítor Manuel Mendes da Silva and presented to the Faculty of Sciences and Technology, Department of Electrical and Computer Engineering.

September 2022

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Magnitude Modulated Signals Detection with Neural Networks

Diogo Gonçalo dos Santos Henriques

Coimbra, September 2022

Magnitude Modulated Signals Detection with Neural Networks

Supervisor:

Professor Doutor Marco Alexandre Cravo Gomes

Co-Supervisor:

Professor Doutor Vítor Manuel Mendes da Silva

Jury:

Professor Doutor Fernando Manuel dos Santos Perdigão

Professor Doutor Marco Alexandre Cravo Gomes

Professor Doutor Pedro Manuel Gens de Azevedo de Matos Faia

Dissertation submitted in partial fulfillment for the degree of Master in Electrical and Computer Engineering.

Coimbra, September 2022

This work was developed in collaboration with:

Acknowledgements

Primeiramente, quero deixar um grande agradecimento aos meus orientadores, Professor Doutor Marco Gomes e Professor Doutor Vítor Silva, por terem acreditado em mim para levar este desafio a bom porto, mostrando não só o profissionalismo que lhes é devido, mas também a liberdade, confiança e boa disposição durante todo o processo. Foi um privilégio poder integrar a vossa equipa e aprender convosco.

Em segundo lugar, agradeço ao DEEC e, particularmente, à sua comunidade, pela formação e valores conferidos nestes últimos anos. Um obrigado também ao Instituto de Telecomunicações que foi a minha segunda casa nos últimos tempos da minha passagem pela Universidade de Coimbra.

Deixo também um grande agradecimento aos companheiros do "Laboratório de *Photoshop*" por todas as histórias e vivências partilhadas e por todas as *inside jokes* que levaram a *memes* que ficarão para sempre eternizados! Obrigado Óscares, João, Filipe, Zé e Pedro por terem tornado esta passagem mais leviana e por toda a ajuda que me deram!

Aos amigos de longa data que entraram comigo neste curso e que partilharam comigo esta etapa, Jof, Vasco, Daniel e Maia, deixo-vos também um grande obrigado!

"O que Coimbra uniu, ninguém separa!" Pois bem, ao Edgar, companheiro de estudo e atividades para além do mesmo; ao André, pelo zelo demonstrado, aventuras vividas e demais peripécias; ao Óscar, pela partilha, paciência e companheirismo ao longo de toda esta jornada; à Marta, pela constante positividade mesmo nos momentos mais difíceis e por toda a ajuda; ao Marco, pelos desabafos, vivências e conhecimentos partilhados; ao Miguel, pela prontidão a ajudar e por todos os trabalhos em que cooperámos; ao Simão, pelo seu feitio particular; e ao Cavaleiro pela boa disposição e preocupação para com os outros: a todos um gigante obrigado!

Aos que sempre cá estiveram e sempre cá estarão, não há palavras para agradecer. Obrigado mãe, por sempre me fazeres ver que não é possível agarrar o mundo com as duas mãos, mas é possível construir um caminho à sua volta! Obrigado mano, por desde pequeno me ensinares a ver sempre para além do que é óbvio e por estares sempre presente para mim!

Obrigado João, por sempre me incentivares a ser melhor e por me dares os meios para o alcançar! Obrigado Telma, por seres a minha conselheira e partilhares comigo a tua experiência e conhecimento! Obrigado avó Lina, pelo orgulho que transparecia e reluzia nos seus olhos! Obrigado a todos pelo apoio, carinho, alento e força que me deram ao longo deste caminho.

Por último, deixar um agradecimento especial à Patrícia por todo o carinho, amor, força, paciência e compreensão ao longo destes últimos anos. Obrigado por caminhares lado a lado comigo, juntos, e por tornares esta etapa mais leve e bonita!

A todos os que deixaram a sua marca no meu percurso académico,

Muito Obrigado!

Este trabalho é financiado pela FCT/MEC através de fundos nacionais e quando aplicável cofinanciado pelo FEDER, no âmbito do Acordo de Parceria PT2020 no âmbito do projeto UIDB/50008/2020, UIDP/50008/2020.

"The only limit to our realization of tomorrow will be our doubts of today."

— Franklin D. Roosevelt

"I think the brain is essentially a computer and consciousness is like a computer program. It will cease to run when the computer is turned off. Theoretically, it could be re-created on a neural network, but that would be very difficult, as it would require all one's memories."

— Stephen Hawking

Abstract

The exponential growth of Deep Learning (DL) algorithms as drawed the attention of the most wide variety of areas as they prove to be applicable in almost every imaginable use case, giving relatively fast solutions, with a desirable abstraction in a world thirsty for reliable solutions and with only two requirements: huge quantities of data and computational power. The telecommunications area is no different, reaching for answers to new problems or optimizations for already existing ones. An example of this is the tradeoff made by the Magnitude Modulation (MM) technique. As current applications become more and more demanding, with increasingly higher data rates and consequently higher spectral efficiencies, Root-Raised Cosine (RRC) filters with a low roll-off and high order constellations have become preponderant. With this, an undesirable increase in the signal's Peak-to-Average Power Ratio (PAPR) occurred, which has harmed the Power Amplifier (PA) efficiency. The MM was then one of the techniques that solved this problem. However, it undermined the demodulator, establishing the already mentioned compromise.

Traditional demodulators perform worst when subjected to magnitude modulated and noisy symbols. Hence, in this thesis, different single DL models were developed and tested, as well as multi-model systems, aiming to attenuate the effect of MM at 4-QAM symbols when transmitted through an Additive White Gaussian Noise (AWGN) channel. The models were always thought in a way that allowed to keep the simplicity high and computational expensiveness low. Thus, three models were built based on the following Neural Networks (NN): Feed-Forward Neural Network (FFNN), Convolutional Neural Network (CNN) and Bidirectional Long Short-Term Memory (BLSTM). The most suited E_b/N_0 ratio for training was studied, as well as the symbol's window length to consider.

Hence, this thesis presents a new demodulator based on deep neural networks models, capable of mitigate the MM effect at 4-QAM symbols, transmitted through an AWGN channel. Furthermore, it leaves the possibility of taking the results even further via a multi-model system.

Keywords: Deep Learning, Neural Networks, Magnitude Modulation, Demodulator

Resumo

O crescimento exponencial dos algoritmos de Aprendizagem Profunda (*Deep Learning*) atrai a atenção das mais variadas áreas, reforçado pelo facto de serem aplicáveis em quase todos os casos imagináveis, facultando soluções de forma relativamente rápida e com uma abstração cada vez mais desejável num mundo que urge de soluções confiáveis e com dois requisitos principais: grandes quantidades de dados e poder computacional. A área das telecomunicações não é diferente, procurando soluções para novos problemas ou otimizações para problemas já existentes. Um exemplo disto é o compromisso estabelecido pela técnica Modulação de Magnitude (MM). À medida que as aplicações atuais se tornam cada vez mais exigentes, com taxas de débito cada vez mais altas e, consequentemente, crescentes eficiências espetrais, o uso de filtros Root-Raised Cosine (RRC) com baixo roll-off e constelações de ordens cada vez mais elevadas tornaram-se fulcrais. Contudo, isto leva ao aumento indesejado do rácio entre a potência de pico e a potência média dos sinais, o que diminui drasticamente a eficiência dos amplificadores de potência. A MM foi então uma das técnicas propostas para resolver este problema, permitindo um aumento da eficiência de transmissão. No entanto, prejudica o desempenho do desmodulador, estabelecendo o compromisso já mencionado.

Os desmoduladores tradicionais veem o seu trabalho dificultado quando na presença de símbolos modulados em magnitude e transmitidos por canais ruídosos. Desta forma, nesta tese, diferentes modelos de DL foram desenvolvidos e testados individualmente, assim como a sua combinação, com o objetivo de atenuar o efeito da MM na desmodulação de símbolos para a modulação 4-QAM, aquando da transmissão num canal AWGN. Os modelos foram projetados sempre com a premissa de os manter o mais simples possível e com baixo peso computacional. Os três modelos construídos acentam nas seguintes redes neuronais: Feed-Forward Neural Network (FFNN), Convolutional Neural Network (CNN) e Bidirectional Long Short-Term Memory (BLSTM). O rácio E_b/N_0 mais apropriado para o treino dos modelos foi estudado, assim como o tamanho da janela de símbolos a considerar.

Por conseguinte, esta tese apresenta um novo desmodulador baseado em modelos com redes neuronais profundas, capaz de mitigar o efeito da modulação em magnitude em símbolos

de constelações 4-QAM e transmitidos em canais AWGN. É ainda deixada a possibilidade de melhorar os resultados através da utilização de um sistema com modelos complementares.

Palavras-chave: Aprendizagem Profunda, Redes Neuronais, Modulação de Magnitude, Desmodulador

Contents

List of Acronyms

Adam	Adaptive Moment Estimation
AF	Activation Function
AI	Artificial Intelligence
ANN	Artificial Neural Network
AUC	Area Under the Curve
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BLSTM	Bidirectional Long Short-Term Memory
CNN	Convolutional Neural Network
CVNN	Complex-Valued Neural Network
DAC	Digital-to-Analog Converter
DL	Deep Learning
DNN	Deep Neural Networks
ELU	Exponential Linear Unit
FFNN	Feed-Forward Neural Network
FIR	Finite Impulse Response
HPA	High-Power Amplifier
KL	Kullback-Leibler
LDPC	Low-Density Parity-Check

LR	Learning Rate
LSTM	Long Short-Term Memory
LUT	Look-Up Table
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multilayer Perceptron
MM	Magnitude Modulation
MPMM	Multistage Polyphase Magnitude Modulation
MSE	Mean Square Error
NN	Neural Networks
OQPSK	Offset Quadrature Phase Shift Keying
PA	Power Amplifier
PAPR	Peak-to-Average Power Ratio
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
ReLU	Rectified Linear Unit
RRC	Root-Raised Cosine
RNN	Recurrent Neural Network
ROC	Receiving Operating Characteristics
SC	Single-Carrier
SNR	Signal to Noise Ratio
TC	Trellis Coding

List of Figures

List of Tables

1 Introduction

We live in an era where information, even if in a hidden way, is becoming our most valuable asset. Diving into subjects like Big Data, Blockchain, Data Mining, and many others, it is clear the increasing dimensions that they have been taking. What most people do not realize is that, in order for all this data to flow correctly, emergent, up-to-date, complex and reliable systems of transmission and reception are needed.

1.1 Motivation

There is no way to deny the emergent application of Machine Learning (ML) and Deep Learning (DL) techniques in everything that surrounds us, in a way that it is becoming difficult to think of something in our daily life that does not have some Artificial Intelligence (AI) behind it. For example, the simple process of writing an e-mail or text message, has Recurrent Neural Network (RNN) based models embedded to predict the next word to be written. It is rare to enter a website without the pleasant pop-up "accept cookies", which allows algorithms to cross-reference our searches with our tastes in order to present more specific advertisements. Taking a simple photo can become a DL task in order to recognize faces, for example, using Convolutional Neural Network-based models, and the process of editing can also become quite complex if unwanted elements are present. With AI, that job turned out to look simple, as smartphones, nowadays, come with neural engines to support tasks like the ones exposed. All this to remind that, even though AI or ML/DL might be seen as black boxes from a high level, their potential is immeasurable.

Successfully applied in fields like image and speech recognition, self-driving cars, online fraud detection, and many more, deep learning algorithms are showing as well great results in the signal processing systems area and have high hopes to take many techniques a step further.

The efficiency of a High-Power Amplifier (HPA) employed in a system's transmitter is highly dictated by the signal being transmitted. Magnitude modulated and bandwidth

limited information signals show high carrier fluctuations, which lead to the use of linear power amplifiers (PAs). Anyhow, as nowadays applications demand for increasingly higher data rates, which implies higher spectral efficiencies, very low roll-off Root-Raised Cosine (RRC) filters and constellations with higher orders have become a constant. In this way, undesirable consequences emerged, like an increase in the signal's Peak-to-Average Power Ratio (PAPR). With this raise, the PA efficiency is drastically reduced, typically to around 16% [1].

Data *Magnitude Modulation* was first proposed as a concept by Miller *et al.* [1, 2] and was used to construct an adaptive peak-suppression algorithm that worked well for Phase Shift Keying (PSK) and generalized PSK constellations. Later, Tomlinson *et al.* [3] improved the initial concept of MM with an Look-Up Table (LUT) implementation that was successfully applied to QPSK and Offset Quadrature Phase Shift Keying (OQPSK) constellations. Lastly, Gomes *et al.* [4, 5, 6, 7, 8] presented a Multistage Polyphase Magnitude Modulation (MPMM) scheme to control the envelope's peak power of Single-Carrier (SC) band limited signals in a more efficient way. This novel technique allowed to expand the MM concept to higher order constellations, even for non-constant amplitude ones.

However, MM, besides its great contribution to reduce the PAPR, induces higher difficulties in the reception side of the system, which is translated, for example, in a higher Bit Error Rate (BER). Even tough there were efficient solutions proposed to mitigate this drawback [8], they either weren't powerful enough or implied changes in the conventional receivers.

So, the motivation to this thesis is to take advantage of the potential of DL, in particular making use of Deep Neural Networks, to conceive a demodulator that allows to reduce the effect of MM in the received signal's envelope.

1.2 Objectives

The goal of this dissertation is to connect the DL capabilities to the MM technique, aiming to vanish the tradeoff established when the PAPR got successfully reduced, while the errors through a conventional demodulator increased. To do so, a DL-based demodulator for magnitude modulated signals upon transmission on an AWGN channel is studied. We consider multiple simulation scenarios, such as the variation of the noise power of the AWGN channel, and the MM memory to consider upon signal detection.

1.3 Dissertation Outline

This thesis is organized into the following chapters:

- **Chapter 1** presents the motivation and objectives for this study.
- **Chapter 2** gives an overview on the Magnitude Modulation technique, exploring its concept as well as its successive refinements overtime.
- **Chapter 3** introduces some core concepts about Artificial Neural Networks, useful for further understanding the models employed and their details.
- **Chapter 4** gives an explanation of the subjects used in the work, explains the models constructed in detail, and describes and discusses the results achieved with the direct application of the NN-based models.
- **Chapter 5** presents multi-model systems, combining the ones presented in the previous chapter, in an attempt for taking the results a step further.
- **Chapter 6** summarises the conclusions extracted from the work performed and enhances future work and possible refinements to be made.

2 Magnitude Modulation

In order to keep up with present high demands for higher data rates, which implies higher spectral efficiency, very low roll-off RRC pulse shaping filters and constellations with a high-order became much-requested resources. However, these led to undesirable consequences, like an increase on signal's PAPR. This is an issue, since it constrains either the transmitter's Digital-to-Analog Converter (DAC), as well as the HPA, while degrading the system's power efficiency. So, in order to decrease the PAPR, several techniques emerged, such like Trellis Coding (TC) [9, 10] and MM [2, 3, 4]. Although their undeniable efficiency, the trellis PAPR reduction techniques were very computationally expensive and hard to embed in actual operating radio transmission systems, making the MM more desirable since, performance-wise, it was similar and offered a considerable ease of integration with the current systems.

In this chapter, the MM concept will be explored, giving an historical context and followed by the current approaches and improvements made to date.

2.1 MM Concept

The principle of MM [6, 11, 12], illustrated in Figure 2.1, consists of controlling the envelope resulting from the pulse shaping filter by multiplying each complex modulated symbol, s_n, at the entry of the bandwidth limiting pulse shaping filter, by a time-varying factor m_n, with $m_n \geq 0$ and $m_n \in \Re$. The transmitted magnitude modulated signal, x_n, is given by

$$x_n = h_n * \sum_k m_k s_k \delta_{n-kL}, \tag{2.1}$$

in which L represents the oversampling rate of operation of the pulse shaping filter, δ_n being the discrete Dirac delta signal, h_n being the impulse response, and '*' the discrete convolution.

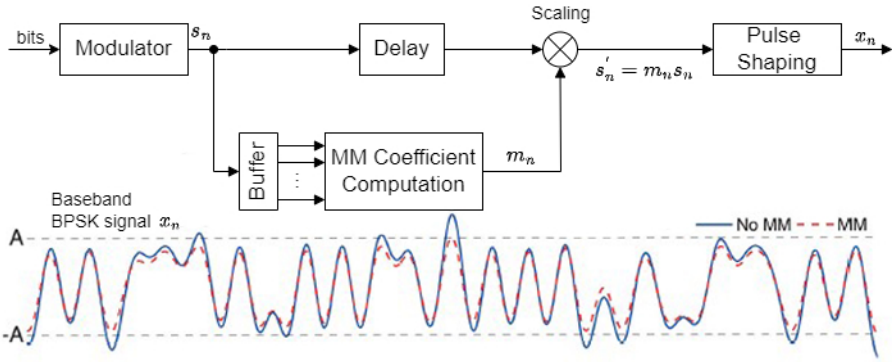


Figure 2.1: Block diagram of a transmitter employing MM and example of a magnitude modulated BPSK signal (adapted from [11]).

Taking into consideration the maximum allowed excursion that does not lead to the PA saturation, a maximum amplitude, A, on the transmitted signal, is set up through the continuous adjustment of the m_n factor applied to each symbol s_n by the MM technique, ensuring that $|x_n| \leq A$, while seeking to maintain the average power of the signal unchanged. This highly reduces the PAPR of the transmitted signal, thus improving the transmitter's power efficiency. Each MM coefficient, m_n, depends on the imposed limit A, on the impulse response, h_n, of the filter, and, ultimately, on the s_n neighbors (that depend on the filter length) that contribute to the transmitted signal x_n. These coefficients can be determined in advance for low order constellations being stored in a LUT (as exposed in the following Section 2.2.2) or computed in real time, using the MPMM technique (Section 2.2.3).

As depicted in the example of Figure 2.1, MM smooths all the peaks that surpass the desired amplitude A, with the advantage of not affecting the spectral bandwidth once it is applied previously to pulse shaping.

2.2 Context

MM was primarily proposed in 1998 by Miller *et al.* [1, 2] in order to develop an adaptive peak-suppression algorithm that performs well for PSK and generalized PSK constellations. About five years later, Tomlinson *et al.* [3] proposed an alternative LUT-based implementation which was shown to be very efficient when applied to QPSK and OQPSK constellations. Later, Gomes *et al.* presented an improved evolution to the LUT-based MM technique [13, 14], as well as a filter polyphase decomposition-based MM technique [4, 5, 6, 8], designated Multistage Polyphase Magnitude Modulation, both enabling applying the MM principle to high-order constellations.

2.2.1 Adaptive Peak Supression Algorithm

So, as the first introduction to the concept of MM, Miller *et al.* [1, 2] came up with this algorithm that adjusts the amplitude of every single data pulse, trying to attenuate or even suppress peaks in the transmitted signal.

Working majorly well for PSK and concentric PSK formats, this algorithm was based in a few considerations. Firstly, the transmitted information in PSK formats is held in the phase, making this type of modulation very robust to tenuous changes in the amplitude of the signal. On the other hand, at concentric PSK formats, there is some data information being sent in the amplitude, obligating to a wise choice of the N-PSK format in order to guarantee that the distance between points on the same circle, i.e., corresponding to the same original constellation symbol, is smaller than the distance between points on different circles, making this format also impartial to small amplitude variations. Secondly, using a pulse shape such as the RRC to constraint the bandwidth, when a peak occurs in the transmitted waveform at some instant t_0, in a way that $(k-1)\,T_S < t_0 < k\,T_S$, the main contribution to that peak is due to the pulses corresponding to the $(k-1)$th and kth data symbol, being T_S the symbol's duration. In this way, reducing the amplitude of these, will substantially reduce the resultant peak. Lastly, considering a long stream of random data symbols, it will exist some symbol intervals where the peak value of the transmitted waveform is somewhat small, inducing that the amplitudes of the data symbols on each side can be raised without raising the overall peak power of the transmitted waveform. This allow to maintain the average power constant, as the amplitudes that have been decreased are balanced.

After simulation, it was concluded that this algorithm had a lot of potential, even though it left some drawbacks and loose ends. Firstly, due to the small adjustments made in the amplitudes of the data symbols, as a way to get exempt of peaks in the transmitted symbols, the sensitivity to noise was increased, inducing a tradeoff between PAPR and average power efficiency. Then, this algorithm brings a somewhat computational complexity attached, which will be addressed later. Finally, the fact that the information is processed in blocks, introduces a delay. So, as exposed, there are some topics to be further improved.

2.2.2 LUT-based MM approach

Seeking to apply MM to other constellations like QPSK and OQPSK and addressing some of the problems identified by Miller *et al.*, refering to the computational complexity of the algorithm, Tomlinson *et al.* [3] came up with a LUT-based implementation.

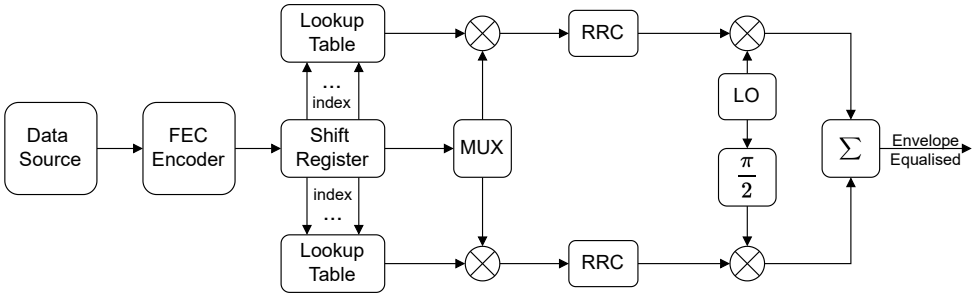


Figure 2.2: Magnitude modulation system proposed by Tomlinson *et al.* (adapted from [3]).

As can be seen in Figure 2.2, the idea behind this approach was to magnitude modulate each data pulse before the RRC filtering. The magnitude modulation coefficients are defined based on the symbol which they apply to and its closest $(m-1)$ left and right neighbors, thus having the MM system memory. These coefficients are computed *a-priori* using an iterative loop (described by Algorithm 1) and stored in two 2^m-size LUTs (for their in-phase and quadrature signal components, respectively), which enables this data magnitude modulation implementation to be applied in real time.

The iterative loop was composed by the following steps, described in Algorithm 1.

Algorithm 1 LUT-based MM iterative loop to determine the MM coefficients.

i. Filter data stream using RRC filtering.

ii. Limit the magnitude of the resulting signal to 1.

iii. RRC filter the signal again. (When no more signal limitation occurs at step ii, this filter will act as true matched filter.)

iv. Sample the signal to extract the data values, now magnitude modulated.

v. Return to step i with the magnitude modulated data while signal limitation occurs in step ii.

vi. Output the absolute values of the magnitude modulated data.

These factors are then used to modulate the central bit of the data sequence. Apart from other problems, MM leads to variations in the bit level, inducing, as mentioned, a higher sensitivity to noise for some data bits, which ones that tend to dominate the BER of the system. However, the authors have shown that error correction coding is quite effective in compensating for data magnitude modulation, as it improves the signal's resistance to noise.

2.2.3 Multistage Polyphase Magnitude Modulation

The MPMM scheme, proposed by Gomes *et al.* [4, 5, 6, 7, 8], introduced the capability of computing the MM coefficients in real-time via a polyphase filter structure that is solely dependent on the RRC impulse response. Because it was independent of the modulation used on the data symbols, this scheme allowed the MM concept to be generalized to higher order constellations, even if they were not of constant amplitude.

This technique foundation was the *polyphase decomposition* of the pulse shaping filter, $H(z)$, FIR. So, using the concept of *multirate systems* [15] applied to FIR filters, the structure shown in Figure 2.3 established the kernel to this alternative MM approach.

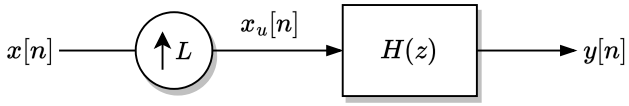


Figure 2.3: Interpolator: 1-to-L up-sampler preceding a FIR filter $H(z)$ (adapted from [8]).

Considering a causal FIR transfer function $H(z)$ of order N:

$$H(z) = h[0] + h[1]z^{-1} + ... + h[N]z^{-N}. \tag{2.2}$$

For any $L \leq N + 1$, $L \in Z$, the previous equality can be reformulated as

$$H(z) = \sum_{i=0}^{L-1} z^{-i} E_i(z^L), \tag{2.3}$$

with

$$E_i(z) = \sum_{n=0}^{\lfloor (N+1)/L \rfloor} h[nL + i + \lambda]z^{-n}, 0 \leq i \leq L - 1. \tag{2.4}$$

The added $\lambda \in Z$ represents a time-delay or advance ($\lambda < 0$ or $\lambda > 0$, respectively) of the filter impulse response. Thus, from (2.3) and (2.4) it is possible to transform the structure shown in Figure 2.3, first by replacing $H(z)$ with a polyphase bank, as shown in Figure 2.5a, and then, using the *noble identity*[1] presented in Figure 2.4, bringing the upsampler through the polyphase filters, Figure 2.5b.

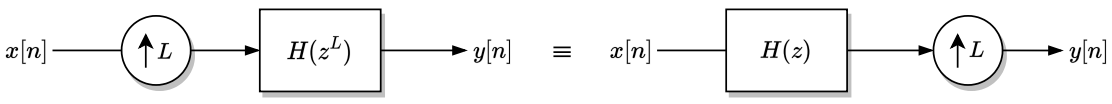


Figure 2.4: Noble identity (adapted from [8]).

[1] An equality that allows the upsampler operator to be commutated, simplifying the multirate structure.

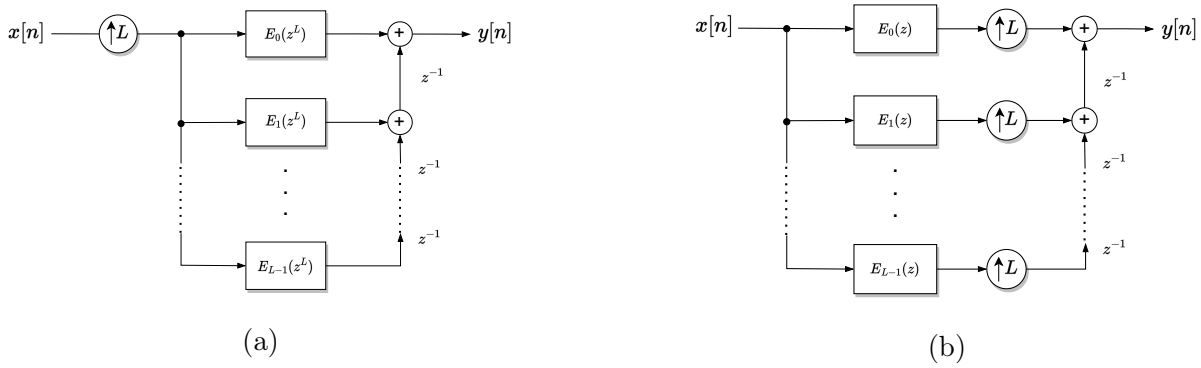


Figure 2.5: $H(z)$ decomposition: (a) replacing it by the polyphase bank; (b) applying the noble identity (adapted from [8]).

So, the pulse shaping filter shown in Figure 2.1 was replaced by its efficient multirate polyphase derivation.

Once the RRC pulse shaping filter is derived, the block depicted in Figure 2.6 was developed to adjust the symbol's amplitude in order to control the output excursion of the polyphase filters $E_i(z)$. These were divided into filters $G_{0i}(z)$ and $G_{1i}(z)$, whose impulse responses are

$$g_{0i}[n] = \begin{cases} e_i[n] & , \ 0 \leq n \leq N \\ 0 & , \ otherwise \end{cases}, \tag{2.5}$$

$$g_{1i}[n] = \begin{cases} e_i[n+N+1] & , \ 0 \leq n \leq N-1 \\ 0 & , \ otherwise \end{cases}. \tag{2.6}$$

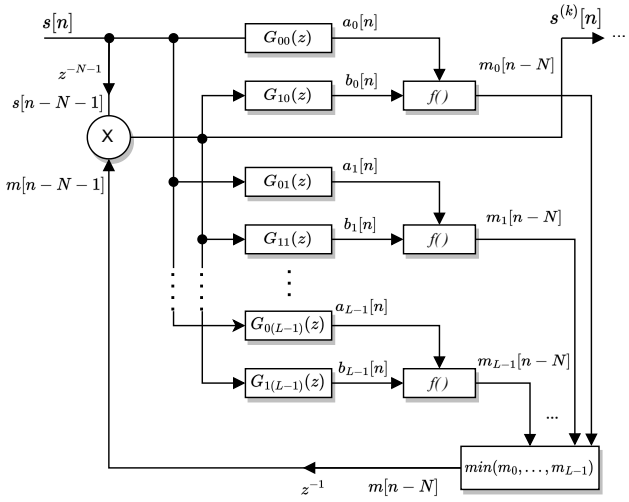


Figure 2.6: MPMM block for an arbitrary stage k (adapted from [8]).

Without getting into too much detail, the purpose of this block is to predict the output of each $E_i(z)$ in order to determine the MM coefficient that, when applied to the symbol prior to the pulse shaping filter, allows to limit its excursion.

2.3 Detection

As pointed out earlier in this document, the MM technique was introduced as a successful way to decrease the PAPR of a signal to be transmitted, which is a value that dictates the efficiency of the HPA, i.e., a high PAPR places high linearity requirements on the HPA, resulting in high power consumption and, as a result, low power efficiency.

Nonetheless, as in most practical cases, in order to improve something, commitments have to be made along the way. In this case, conventional demodulators, completely unaware of the MM applied at the transmitter's side, saw their performance hugely degraded due to the distortion provoked by the MM, which increased the symbols' sensitivity to the noise introduced by the transmission channel. This had a significant impact on the BER during demodulation.

This problem was then addressed with the introduction of Low-Density Parity-Check (LDPC) codes and by reversing the MPMM on the receptor side [8], but left a margin for improvement. In this way, seeking for new strategies to bring the MM awareness to the demodulation side of the system, DL, in particular, Artificial Neural Networks (ANNs), seemed capable of replacing the conventional demodulators, which led to this study.

3 Artificial Neural Networks

The birth of the so-called ANN remarks dates back to 1943, but it was only in 1975 that their potential was unlocked with the appearance of the backpropagation algorithm that allowed the training of multi-layer networks. These structures were based on the biological composition of the animals' nervous systems, whose basic unit is the neuron. A real neuron is essentially composed of: a branching dendritic tree that establishes the connection with other neurons and collects their signals; a cell body that processes the signals and produces a response; and a branching axon responsible for transmitting that response to the following neurons. In a similar way, ANNs are composed of a variable number of processing units (neurons) that are joined through weighted connections (dendrites/axons) whose outputs depend on the incoming value and on the weights attached to each link.

In this chapter, the basics of neural networks will be addressed, particularly those composing the models used in this study.

3.1 Basic unit

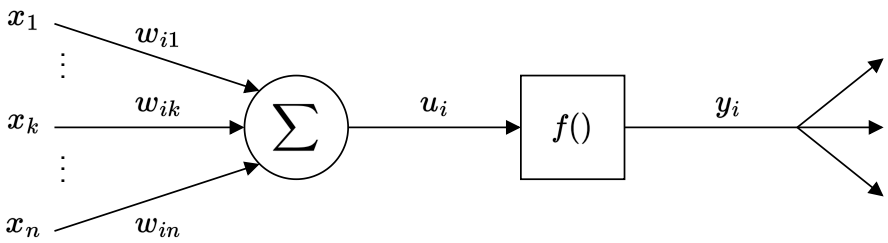


Figure 3.1: Artificial neuron model (adapted from [16]).

As already introduced, the model presented in Figure 3.1 constitutes the basis of a neural network. A neuron has a response according to

$$y_i = f\left(\sum_j^n w_{ij} x_j\right) \tag{3.1}$$

where x_j are the inputs to the network or the outputs from previous nodes, w_{ij} are the weights associated to each link (to a particular neuron i), and $f()$ is a nonlinear function, called Activation Function (AF). This unit computes a weighted linear combination of its inputs that pass through the AF, yielding a scalar output. To u_i (Figure 3.1) can be added an extra parameter, so-called bias, not represented in the figure.

By combining multiple units in parallel, it is possible to form a layer, and by sequentially linking layers, a feedforward network structure arises, which is the basis for almost every network nowadays.

3.2 Supervised Learning

In machine learning, there are essentially three types of learning: supervised, unsupervised, and reinforcement learning. The last two are irrelevant to this study, since only supervised learning was considered.

Supervised learning can be seen as a function approximation problem, in the sense that the model is trained with a pair of inputs, x, and expected targets, t, i.e., we pretend to find a function $y()$ that, given x, outputs t. It is unnecessary to say that in most cases, there is not an obvious relationship between the values, also called features, and their correspondent targets. Otherwise, much more comprehensible methods could be used [17].

A neural network "learns" by iterative refinement of its connections weights, which is obtained by the back-propagation algorithm (explained in Section 3.4). Anyhow, within the training phase, the inputs are propagated through the network, which produces an output. Even when fully trained, a model is not, and should not be, perfect (generalization capacity). So, an error function is used in order to evaluate the deviation between the target and the output of the model, and the goal is to minimize the error for the training data without incurring overfitting, in which the model "memorizes" the data presented to it, which is translated in poor performance when different data is shown.

Training can then be seen as an optimization problem with a considerable number of factors prone to being adjusted, such as: features to consider, network structure, error function, optimization method, among others.

3.3 Activation Functions

An AF, also known as a transfer function, is used to introduce non-linearity into the neural network. Without an AF as an intermediate step, each neuron would only be performing a linear combination of its inputs, gathering its weights and biases. In this way, it would not matter if we added layers to the network since combining two linear functions results in a linear function as well; that is, all layers would behave in the same way.

So, there exist essentially three types of activation functions: binary step function, linear and non-linear. The binary step function is expressed by

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

and is useful for deciding if a neuron should be activated or not based on a threshold. It presents some constraints as it cannot be employed in multi-class classification problems and, since its gradient is zero, it jeopardises the back-propagation process.

The linear activation function, or identity function, is straight forward, as it does nothing to the input, which is why it is barely used. Additionally, given its derivative is a constant, the back-propagation algorithm is inapplicable, and, as all consequent layers are linear functions of the first one, the network can be resumed to one layer. Finally, the most useful AFs are the non-linear ones, as they permit the use of back-propagation (their derivative is input related). Some of the most used AFs are presented in Table 3.1.

Table 3.1: State-of-the-art activation functions.

AF	**Equation**	**Pros**	**Cons**
Sigmoid/ Logistic	$\frac{1}{1+e^{-x}}$	Range between 0 and 1, which makes it optimal for probabilistic outputs. Differentiable and gives a smooth gradient.	Makes the training difficult and unstable. Vanishing gradient problem.
Tanh	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	Output zero centered, which allows to differ the outputs as strongly negative, neutral or strongly positive. Makes the learning for consequent layers easier.	Vanishing gradient problem.
ReLU	$max(0, x)$	Computationally efficient (only certain neurons are activated). Accelerate the convergence of gradient descent towards the global minimum.	Dying ReLU problem.
Leaky ReLU	$max(0.1x, x)$	Same as the ReLU, and enables back-propagation even for negative values.	For negative input values, the predictions may be inconsistent. The gradient is small for negative values, which delays the training process.
ELU	$\begin{cases} x, & x \geq 0 \\ \alpha(e^{x}-1), & x < 0 \end{cases}$	Smooth transition around the origin. Avoids the Dying ReLU problem.	Increases the computational time due to the exponential operation. Exploding gradient problem.

These are just some of the most commonly used activation functions. There are many others, especially derivations of the ReLU.

3.4 Back-Propagation

Back-propagation was developed by Rumelhart, Hinton, and Williams [18] and is currently the most widely used training method for FFNNs.

As said earlier, the learning capability of a neural network comes from the fine adjusting of its weights to the data presented to it. The role of the back-propagation algorithm is precisely to, making use of the chain rule, modify the weights with a basis on the error between the output and the desired target.

So, in order to train a network, there are the basic steps enumerated in Algorithm 2.

Algorithm 2 Basic steps of the training stage of a NN.

i. Feed the data and propagate it through the network to get the outputs.

ii. Match the outputs with the targets to measure the error.

iii. Calculate the derivative of the error relating to the weights, i.e., the gradient $\partial E/\partial w_{ij}$.

iv. Amend the weights to minimize the error, by moving in the opposite direction of the gradient.

v. Repeat until the error/performance is acceptable.

Trying to explain the back-propagation algorithm without getting into too much detail, let's consider a simple example with a network with two layers (Figure 3.2), sigmoid activation function, squared error cost function (3.3) and gradient descent [19] as the optimisation algorithm.

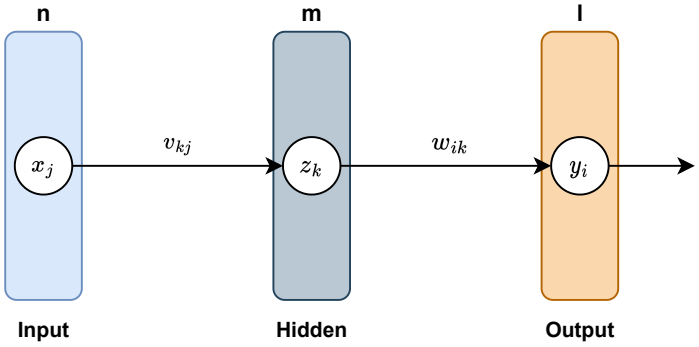


Figure 3.2: Two-layer FFNN example. x, z and y represent the neurons of the input, hidden and output layers, respectively. v and w are the weights attached to each connection between layers. n, m and l are the total number of neurons in the correspondent layers.

$$E_i = \frac{1}{2}\sum_{i=0}^{l}(t_i - y_i)^2 \tag{3.3}$$

Let's agree then, to simplify the notation, that

$$I_i = \sum_{k=0}^{m} w_{ik} z_k, \tag{3.4}$$

so

$$y_i = \sigma(I_i) = \frac{1}{1 + e^{-I_i}}. \tag{3.5}$$

Introducing η as the learning rate hyper-parameter (this will be explained in Section 3.5), the updating factor of the weights between the hidden and output layer can be defined as

$$\Delta w_{ik} = -\eta \frac{\partial E_i}{\partial w_{ik}} \tag{3.6}$$

which, through the chain rule, can be decomposed into

$$\Delta w_{ik} = -\eta \frac{\partial E_i}{\partial w_{ik}} = -\eta \frac{\partial E_i}{\partial y_i}\frac{\partial y_i}{\partial I_i}\frac{\partial I_i}{\partial w_{ik}} = \eta(t_i - y_i)y_i(1 - y_i)z_k. \tag{3.7}$$

Now considering E as the total error and

$$I_k = \sum_{j=0}^{n} v_{kj} x_j, \tag{3.8}$$

from the hidden layer to the input layer, the updating factor is represented as

$$\Delta v_{kj} = -\eta \frac{\partial E}{\partial v_{kj}} \tag{3.9}$$

which can be extended in

$$\Delta v_{kj} = -\eta \frac{\partial E}{\partial v_{kj}} = -\eta \frac{\partial E}{\partial I_k} \frac{\partial I_k}{\partial v_{kj}} = -\eta \frac{\partial E}{\partial I_k} x_j. \tag{3.10}$$

So, the last thing to do is to establish a relation between the total error E and the input signal I_k. The key idea here is that the total error E is equal to the sum of the partial errors E_i.

$$\frac{\partial E}{\partial I_k} = \sum_{i=0}^{l} \frac{\partial E_i}{\partial I_k} \tag{3.11}$$

Decomposing the previous equality like

$$\frac{\partial E}{\partial I_k} = \sum_{i=0}^{l} \frac{\partial E_i}{\partial I_k} = \sum_{i=0}^{l} \left(\frac{\partial E_i}{\partial I_i} \frac{\partial I_i}{\partial I_k} \right) \tag{3.12}$$

and establishing that

$$\frac{\partial E_i}{\partial I_i} = \frac{\partial E_i}{\partial y_i} \frac{\partial y_i}{\partial I_i} = -(r_i - y_i) y_i (1 - y_i) = -\delta_i \tag{3.13}$$

and

$$\frac{\partial I_i}{\partial I_k} = \frac{\partial I_i}{\partial z_k} \frac{\partial z_k}{\partial I_k} = w_{ik} z_k (1 - z_k) \tag{3.14}$$

we can put it all together and define

$$\Delta v_{kj} = \eta z_k (1 - z_k) \left(\sum_{i=0}^{l} \delta_i w_{ik} \right) x_j. \tag{3.15}$$

With both the updating factors determined, all that remains is to add them to the previous weights values

$$w_{ik}(t+1) = w_{ik}(t) + \Delta w_{ik}$$
$$v_{kj}(t+1) = v_{kj}(t) + \Delta v_{kj}$$

with t and $t+1$ denoting the currently and following iterations, respectively.

Thus, all the weights get refreshed to a better suited value for the purpose with a backward pass.

3.5 Hyperparameters

3.5.1 Learning Rate

The learning rate, usually identified by η, is one of the most important hyperparameters. This factor is responsible for controlling the speed at which the model learns by influencing the weight updates during the optimisation stage. A learning rate too high leads to faster training but could translate into a miss of the minimum of the lost function. On the other hand, a learning rate too low can induce very slow training and make the model never reach the optimal solution or stay stuck at a local minimum.

3.5.2 Batch Size

The batch size defines how many samples of the training dataset are fed to the model between each weight update. This hyperparameter is highly limited by the amount of memory in the system, since the samples have to be loaded into it.

3.5.3 Number of Epochs

In ML/DL, an epoch is the process that takes all the dataset, divided into batches, to pass through the network. Usually, we define a number of epochs for the training to happen, which means that the entire dataset will be fed to the model that number of times. Complex models can take several thousand epochs to train, but usually techniques like early stopping are used to prevent unnecessary training time (and, consequently, reduce allocated resources), as well as overfitting scenarios. These techniques evaluate the model's performance and if minor or no progress is being made, they cease the training.

3.6 Convolution Layers

The CNN is a special type of network designed to work with images, i.e., two-dimensional data, although it works with one-dimensional and three-dimensional data as well.

The main type of layer in these networks is the convolution layer, which, as the name indicates, performs an element-wise multiplication (dot product) of a set of weights, called the filter/kernel, with the input, similar to a conventional neural network. This kernel is smaller than the input array, allowing it to be applied several times at different points.

Figure 3.3 gives an example of what happens at the convolution layers. The kernel is applied from left to right, top to bottom of the 2D array, generating a scalar value from each convolution that consolidates a feature map. In the example, a 3x3 kernel was considered, but other sizes can be taken into account. Another option made was stride 1, which means the kernel moves only one value each time. Other values can also be chosen, having in mind that, since no padding was applied, the feature map will always be smaller than the input. Padding consists of adding one or more values around the original array to extinguish loss of information.

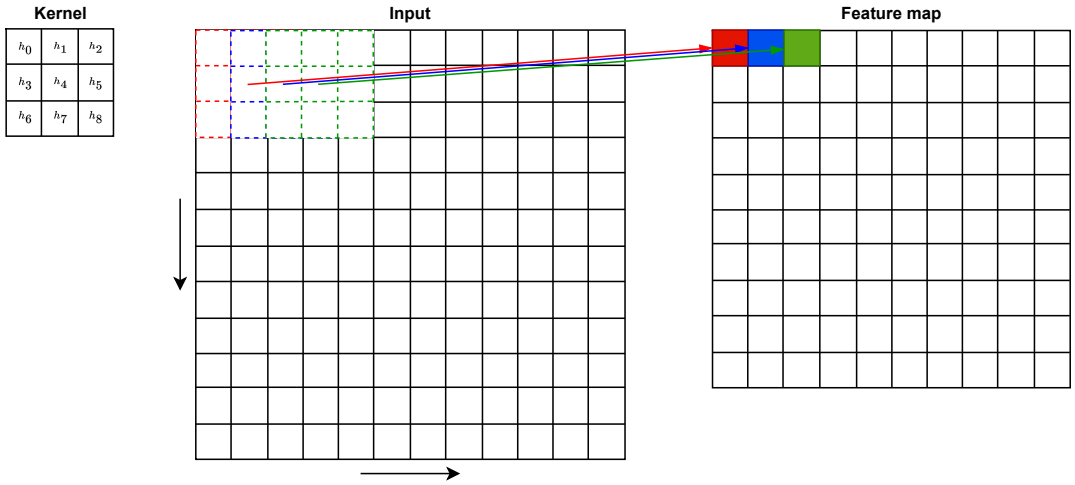


Figure 3.3: Example of convolution on a 2D level, considering a 3x3 kernel, stride of 1 and no padding.

Enhance that many other layers might compose a CNN, like pooling [20] and normalization layers [21, 22, 23].

3.7 LSTM Networks

The Long Short-Term Memory (LSTM) network was introduced by Hochreiter and Schmidhuber in 1997 [24] as a successful way to mitigate the long training times observed in traditional RNNs, mainly due to inadequate error backflow. They are optimal to avoid the long-term dependency problem as they are designed to remember information for long periods of time.

This type of network, similarly to what happens in all the recurrent neural networks, follows a chain structure where the modules are as presented in Figure 3.4.

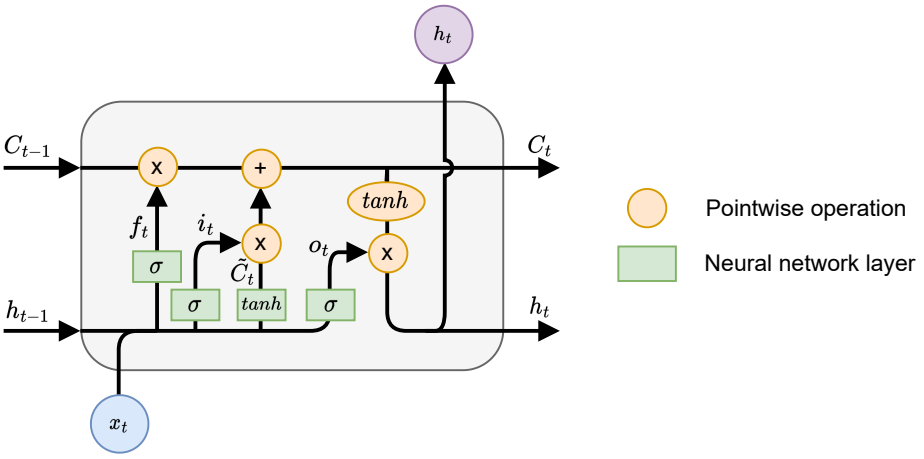


Figure 3.4: LSTM module.

The top horizontal line that flows across the module is the cell state. As it can be seen, there are some minor interactions with this line, i.e., only small alterations are made to the information running through the cell state, carefully controlled by the gates, composed by the sigmoid neural network layer and by the pointwise multiplication. The module has three of these gates, whose function is to safeguard and manage the cell state. As known, the sigmoid function outputs values in the range [0, 1], 0 meaning that nothing passes through and 1 meaning that everything passes through.

As a first step in the module, is the forget gate layer, f_t (3.16). Taking the new input, x_t, and the previous output, h_{t-1}, this gate determines the information of the cell state that is kept.

$$f_t = \sigma\big(W_f.[h_{t-1}, x_t] + b_f\big) \tag{3.16}$$

The W parameters in the previous equation and the ones that follow represent the weights of the neural network layer within each gate.

The next stage, composed of two parts, consists of determining what new data is stored in the cell state. The first part, called the input gate layer, i_t, is given by

$$i_t = \sigma\big(W_i.[h_{t-1}, x_t] + b_i\big), \tag{3.17}$$

and consists of a sigmoid layer that decides which values will be updated. The second part originates, through a tanh layer, a vector of potential new values to add to the cell state, $\tilde{C}_t$, computed as

$$\tilde{C}_t = \tanh\big(W_C.[h_{t-1}, x_t] + b_C\big). \tag{3.18}$$

These two parts are then combined to originate the update to the cell state, according to

$$C_t = f_t.C_{t-1} + i_t.\tilde{C}_t. \tag{3.19}$$

The last step is to determine the output of the module, which is based on the concatenation of the last output and the new input passing through a gate, o_t, and on the tanh of the cell state, as per

$$o_t = \sigma\big(W_o.[h_{t-1}, x_t] + b_o\big) \tag{3.20}$$

and

$$h_t = o_t.\tanh(C_t). \tag{3.21}$$

As a final note about this topic, remark that LSTMs have evolved throughout the years, existing many versions of them [25, 26, 27].

With the fundamental concepts of the MM technique and DL established, it is time to move on to the work done and the corresponding results.

3.8 Performance metrics

Performance metrics are required to assess a model's behavior. Because there are several, the one(s) best suited to the task at hand must be chosen in order to have an accurate perception of the model's capabilities.

Following, are presented the most common metrics used in the state of the art.

Confusion matrix

The confusion matrix is a performance metric that allows to immediately compare the output of the model (predicted labels) with the ground-truth/labels, in terms of the number of samples correctly and incorrectly classified, exposing that distribution by class.

Taking as example a binary classification task, the confusion matrix would be like the one at Figure 3.5. This tool allows to extract several metrics, particularly those exposed below and used in this work.

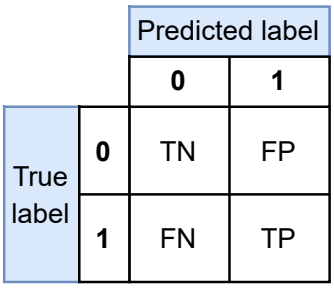


Figure 3.5: Confusion matrix of a binary classification task.

- True Positive (TP): the model predict the positive class (1) and the actual output is the positive class;
- True Negative (TN): the model predict the negative class (0) and the actual output is the negative class;
- False Positive (FP): the model predict the positive class and the actual output is the negative class;
- False Negative (FN): the model predict the negative class and the output is the positive class.

Accuracy

The accuracy metric is probably the most well-known and widely used, as it indicates the percentage of correctly classified samples. It is applicable when dealing with balanced data and is calculated as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{3.22}$$

Precision

Another important metric is the Precision, as it evaluates the proportion of the predicted positive class actually being positive, and is obtained through

$$Precision = \frac{TP}{TP + FP}. \tag{3.23}$$

Recall

This metric gives information about how many of the actual samples of the positive class the model has captured by labeling them as positive.

$$Recall = \frac{TP}{TP + FN} \tag{3.24}$$

F1 Score

The F1 Score metric can be seen as the Harmonic mean of the Precision and Recall metrics. If one of these values tends to be very low, the F1 Score will be closer to the smaller one, giving the model an adequate score.

$$F1Score = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{3.25}$$

Specificity

Also known as True Negative Rate, this metric evaluates the proportion of the negative class that are rightfully predicted as negative, with respect to all the negative class samples.

$$Specificity = \frac{TN}{TN + FP} \tag{3.26}$$

4 MM signals detection with single model NN-based systems

In this chapter, we explain the approaches taken to the development of a NN-based detector for QAM magnitude modulated signals. It also addresses the necessary concepts to further understand the experiences performed in the investigation, giving a detailed description and explaining the decisions that were made.

4.1 4-QAM Constellation

In an M-ary PSK system, the components of the modulated signal are typically constrained in order that the envelope remains constant, resulting in circular constellations. Nevertheless, if that constraint is removed, allowing in-phase and quadrature to be independent of one another, a new type of modulation, known as M-ary QAM, emerges. Once the carrier is modulated in amplitude and phase, QAM is an hybrid type of modulation.

Hence, this constellation is described by two orthogonal passband basis functions:

$$\begin{aligned} \phi_1(t) &= \sqrt{\frac{2}{T_s}}\cos(2\pi f_c t), \quad 0 \leq t \leq T_s \\ \phi_2(t) &= \sqrt{\frac{2}{T_s}}\sin(2\pi f_c t), \quad 0 \leq t \leq T_s \end{aligned} \tag{4.1}$$

in which T_s corresponds to symbol duration and f_c is the carrier frequency. In this way, a transmitted M-ary QAM signal for any symbol k is defined as the engaging of two phase-quadrature carriers:

$$s_k(t) = \sqrt{\frac{2E_0}{T_s}}a_k\cos(2\pi f_c t) - \sqrt{\frac{2E_0}{T_s}}b_k\sin(2\pi f_c t), \quad 0 \leq t \leq T_s \tag{4.2}$$

with E_0 being the energy of the lowest amplitude symbol, and a_k and b_k corresponding to the symbols to be transmitted in phase and quadrature, respectively.

So, referring to the constellation under study, the 4-QAM, represented in Figure 4.1, it is a square constellation since the number of bits per symbol, L, is even:

$$L = \sqrt{M} = 2 \tag{4.3}$$

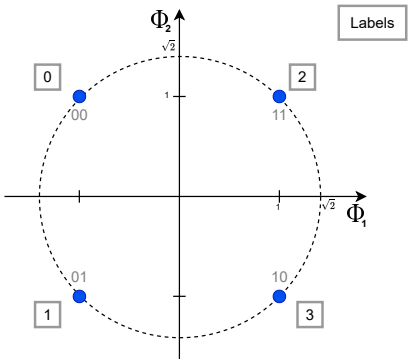


Figure 4.1: Signal-space diagram of a 4-QAM constellation with gray-encoded dibits.

Even though this signaling scheme is bandwidth efficient, it needs to operate in the linear region of the PA to dismiss any signal deterioration [28].

The Magnitude Modulation technique applies polar scaling to the symbols by multiplying both the in-phase (I) and quadrature (Q) components by the same MM coefficient in order to avoid phase modulation, which spreads the constellation symbols throughout the linear functions $y = x$ and $y = -x$, as demonstrated in Figure 4.2. As it can be seen, this brings the symbols closer to the origin, which is why MM increases constellation symbol's sensitivity to noise and can even cause some symbol's errors. Figure 4.3 illustrates a 10000 4-QAM symbol' sequence, magnitude modulated, and scattered with different noise levels. In green are the symbols that kept their quadrant. In red are the ones that changed quadrant due to MM and noise.

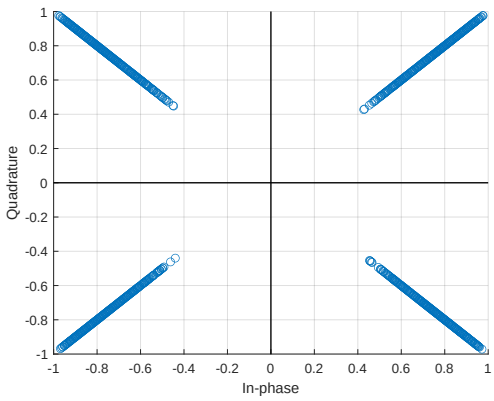
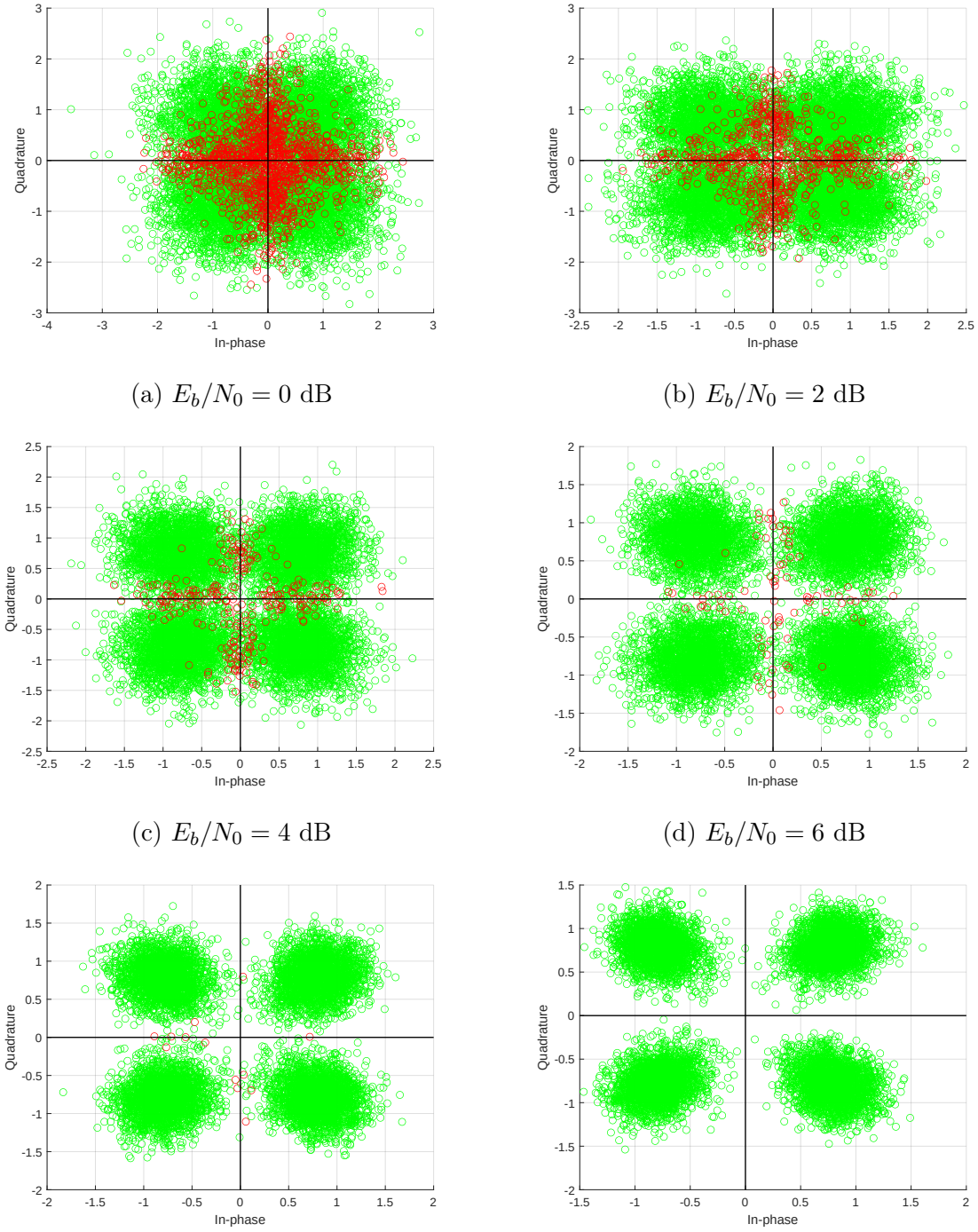


Figure 4.2: Effect of MM applied to 4-QAM symbols.



(a) $E_b/N_0 = 0$ dB

(b) $E_b/N_0 = 2$ dB

(c) $E_b/N_0 = 4$ dB

(d) $E_b/N_0 = 6$ dB

(e) $E_b/N_0 = 8$ dB

(f) $E_b/N_0 = 10$ dB

Figure 4.3: Sequence of 10000 QPSK symbols with MM scattered with different E_b/N_0 ratios. Presented in green are the symbols that kept their original quadrant. The red ones switched quadrants due to magnitude modulation plus noise and are prone to being misclassified by conventional demodulators.

Following in this document, it will be presented BER curves for this type of constellation. As a benchmark or method of comparison with the results achieved, two curves are always shown: one that represents the theoretical demodulation BER of a conventional demodulator,

and another that displays the theoretical demodulation BER of a conventional demodulator when in the presence of MM symbols. Since in this study only the AWGN channel was considered, the first curve mentioned was obtained through (4.6) [29], which was simplified from (4.5), with $M = 4$. Remark that $Q(.)$ is the Gaussian Q-function defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt \ . \tag{4.4}$$

$$BER_{M-PSK} = \frac{2}{\log_2(M)} Q\left(\sqrt{\frac{2E_b \log_2(M)}{N_0}} \sin\left(\frac{\pi}{M}\right)\right) \tag{4.5}$$

$$BER_{4-QAM} = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{4.6}$$

The second curve was obtained by taking a magnitude modulated sequence and considering its transmission through an AWGN channel with different noise powers, and by demodulating it with a conventional and unaware detector. Nonetheless, analytical formulas were determined for the BER performance of MM M-PSK signals transmission over AWGN channels. The first one was presented by Gomes *et al.* [30] and relied on the grasp of the MM factor's distribution statistics. Unfortunately, it was an onerous integral difficult to evaluate and somewhat exclusive to the AWGN channel, i.e., it was harsh to broad to other channel types, like time-dispersive ones. Thus, Equation (4.7) [11] was developed, much simpler than the previous one. Without getting into unnecessary detail, this expression took a Gaussian approximation to model the statistical characterisation of the distortion term introduced by the MM technique, α, which is used as an weighting factor on MM distortion's power: α is a function of the Kullback-Leibler (KL) divergence [31] of the MM coefficients' probability function from the Gaussian approximation.

$$P_e^{MM}(\gamma) \simeq \frac{2}{\log_2(M)} Q\left(\sqrt{\frac{2\bar{m}^2 E_s}{\sigma_{n_z}^2 + \alpha\sigma_d^2 \sin^2(\pi/M)}} \sin\left(\frac{\pi}{M}\right)\right) \tag{4.7}$$

Here, the BER performance is in function with the Signal to Noise Ratio (SNR), γ, which is given by $\gamma = \sigma_{s_n}^2/\sigma_{n_z}^2$, with $\sigma_{s_n}^2 = E\left[|s_n|^2\right] = E_s = E_b \log_2(M)$, where E_s and E_b symbolize the energy per information symbol and per information bit, respectively, and $\sigma_{n_z}^2 = N_0$, this being the power spectral density. Finally, $\bar{m} = E[m]$ is the average constellation symbol position after MM and $\sigma_d^2 = E\left[|s_n' - \bar{m}s_n|^2\right]$ is the power of $d_n = s_n' - \bar{m}s_n$ which is the MM distortion. This nomenclature is in accordance with the block diagram of Figure 2.1. The relation between α and D_{KL} was then found to be accurately expressed by the logarithmic

expression $\alpha = a \log_{10}(D_{KL}) + b$, where a and b depend on the constellation being studied [11].

4.2 Datasets

Datasets had to be generated in order to train and test the DL models. Because the entire project is simulation-based, the data was gathered using a *Monte Carlo Simulation* of digital modulation techniques over a noisy channel with and without magnitude modulation, employing the MPMM technique, as described in Section 2.2.3, using the MATLAB framework.

The base system model of a transceiver employing MM is here remembered and depicted in Figure 4.4. An uncoded stream of bits is modulated, originating the symbol's sequence s. This sequence is magnitude modulated, s_{MM}, passing then through an RRC filter in order to shape the pulse, giving rise to x. This is then transmitted over an AWGN channel. The receptor, here our NN-based system, takes y (x with the noise added by the channel), and outputs $\widehat{s}$, a prediction of the original modulated symbols.

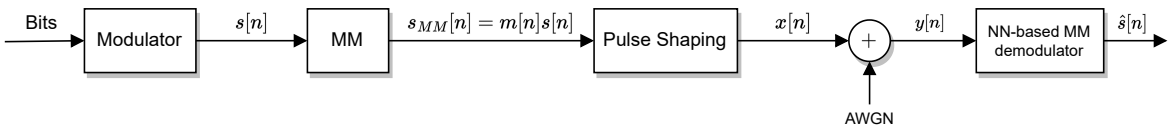


Figure 4.4: Block diagram of a transceiver system comprising MM.

Thus, three different MM symbol sequences were generated: one for training, other for validation and another for testing. Tipically, a portion of the training dataset is used for validation, especially when the data available is limited. However, because that was not the case, a different sequence was chosen. This justify as well why techniques like *cross-validation* were not used. The initial sequences, i.e., those without any modulation, were also saved, since they represent our labels/ground-truth. Enhance that the original sequences had nearly 26.5 million symbols, but due to the number of different datasets required, each one ended to comprise only 15 million samples, enabling to balance the representativeness and richness of the dataset with its memory occupancy.

So, the datasets were constructed according to the following sequence:

i. Stochastic generation of a stream of numbers in the interval $[0, M-1]$, M being the constellation order;

ii. Modulation of the previous stream to QAM symbols, using gray encoding as shown in Figure 4.1;

iii. MM of the sequence, using the parameters of Table 4.1;

iv. Addition of White Gaussian Noise according to the specified SNR given by

$$SNR_{(dB)} = E_b/N_{0_{(dB)}} + 10\ log_{10}(k * coderate) \tag{4.8}$$

with $k = log_2(M)$ being the number of bits per symbol;

v. Decomposition of the symbols in real and imaginary parts[1].

Table 4.1: RRC filter definition parameters for the MPMM technique usage and constants.

Parameters	**Values**
Filter oversampling	4
Roll-off	0.2
Delay	18
Span	36
Coderate	1
k	2
M	4

Once the symbol sequences were prepared, i.e., magnitude modulated and with addition of noise due the transmission channel, the samples were truncated using a sliding window method, as shown in Figure 4.5, with the size of the window depending on the memory being considered by the receiver upon detection.

[1]There has already been a substantial amount of research into Complex-Valued Neural Networks (CVNNs) [32, 33], and the results have been promising. However, their application still is difficult and beyond the scope of this study.

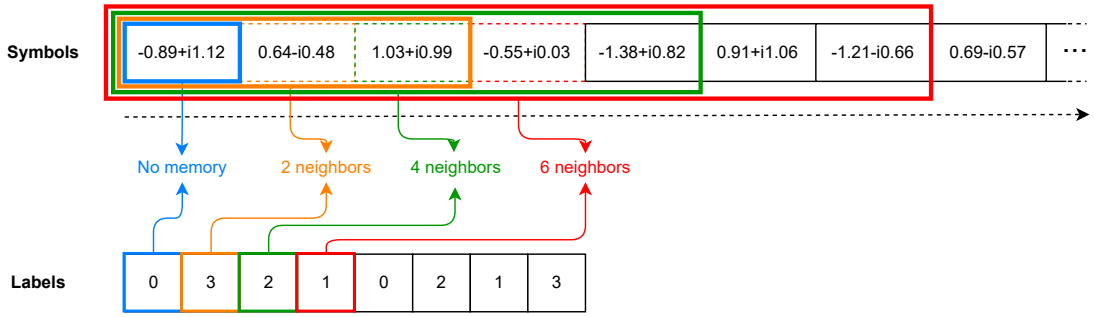


Figure 4.5: Schematic representation of the sliding window method used to build the datasets, showing a sequence of MM symbols with noise and the original sequence, used as ground-truth/labels (see Figure 4.1), for the various window sizes employed.

Usually, when using public datasets, there is a need for preparing the data, which is a critical process nowadays in the data science field, as the Law of Reality (*"Garbage in, garbage out."*) plays a huge role in the ML algorithms' performance. Around 60% of data scientists' time is spent cleaning and organizing the data, so-called feature engineering. Nevertheless, since the data was generated throughout a controlled simulation, the only concerns taken into account were possible, but very unlikely, redundant samples (repetitions) and equally distributed classes within the datasets [34].

4.3 Deep Fully Connected model

As an initial note, this and the following models were fully developed using the PyTorch framework [35].

The main goal of this study was to develop a receiver based on a NN to demodulate QPSK/OQPSK signals with MM. So, as this is an untouched topic yet, we started by testing the performance of one of the less complex neural networks used in supervised learning tasks, the FFNN/Multilayer Perceptron (MLP) [36], using it to demodulate MM signals of one of the simplest constellations, the 4-QAM. A relatively similar approach was made by Liu and Romero [37], where a DNN was used as a demodulator to decode symbols envolved in radar signals. Their model showed up to be very robust, outperforming other current state-of-the-art techniques such match filtering demodulation [38].

Because there is some disagreement about how to count layers in a network, the convention adopted in [16] will be followed for the purposes of this work: an L-layer network has $L-1$ hidden layers and an output layer. After many iterative refinements to the network structure, the final architecture that yielded the best results is depicted in Figure 4.6, making

up five layers, which turns this into a DL model [39]. This architecture counts with a total of 7.549 parameters for the maximum input size, N, tested, and is fully exposed in Appendix A.1. As shown in Figure 4.5, the biggest window considered comprised 7 symbols, which led to $N = 14$ due to the decomposition into real and imaginary parts.

The other decisions made to compose the model, like the activation function, weight initialization, optimization function, and others, are justified hereafter and summarized in Table 4.2.

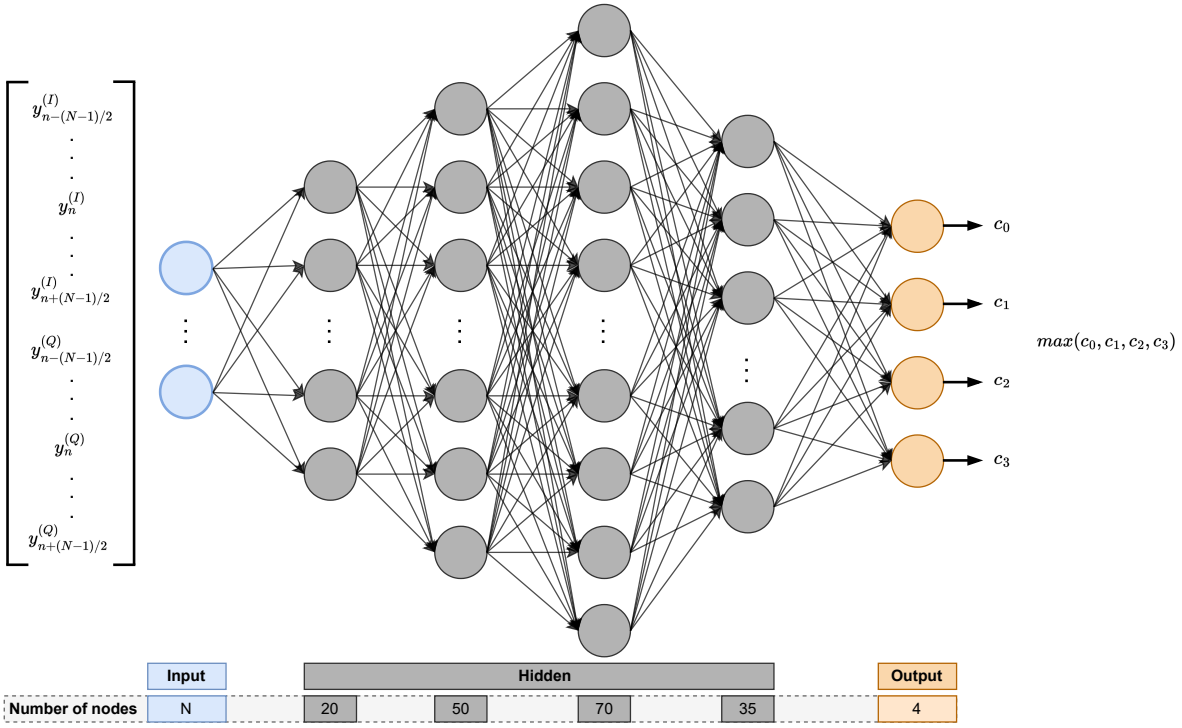


Figure 4.6: Feedforward Fully Connected Neural Network architecture developed. The number of input nodes depends on the memory being considered, N.

Table 4.2: Choices made for the FFNN-based model constitution and hyperparameter values.

(a)	**Activation Function**	ReLU
(b)	**Loss Function**	Cross Entropy
(c)	**Optimization Function**	AMSGrad
(d)	**Learning Rate**	$10^{-2}/10^{-3}$
(e)	**Batch Size**	500000
(f)	**Weight & Bias initialization**	Default

Following, there is a list justifying the options made to the model and presented in Table 4.2.

(**a**) In the process of designing a neural network, a crucial decision passes through the AF to be employed. Without it, the operations within the network would just be pure linear regressions, as mentioned in Section 3.3. Thus, the AFs allow the model to learn the feature representations more efficiently by introducing non-linearity. The Rectified Linear Unit (ReLU) and its derivations [40] have been successfully integrated into the majority of cutting-edge DL applications nowadays, and once they keep showing good results, they are the functions to go for. Another reason why the ReLU was the AF chosen, stands within the exposed at (**f**), since it is less sensitive to random weight initialization. Other AFs like the *Tanh* were tested during the development of the model, but none of them led to better results than the ReLU.

(**b**) As relevant as the previous point, the loss function is another important element of the model since it is the means by which the NN learns. Comparing the prediction made by the model with the true label of the data, the goal within training is to minimize the validation loss without incurring the overfitting problem. Thus, the loss function employed was the Cross Entropy. Known from Information Theory [41], it has been widely employed in the DL field and, as it shows a faster convergence rate than other loss functions like MSE [42, 43], further developments have been proposed to enhance it [44].

(**c**) In order for the parameters of the network, i.e., weights and biases, to be adjusted at each epoch, so that the data can be learned, there is a need for a function that updates them. That is exactly what an optimization function does. Hence, being an improvement from Adaptive Moment Estimation (Adam) [45], one of the currently best and largely used optimizers, the AMSGrad [46] was the optimization function employed in this model. Lets recall that L2 regularization is inherently associated in PyTorch, where the weight decay value used was 10^{-4}.

(**d**) From the several hyperparameters that comprise a DL model, the Learning Rate (LR) might be the one with greater immediate influence (Section 3.5). So, even though AMSGrad adjusts it, an adaptive LR technique was employed, the MultiStepLR, with one milestone in which the LR was multiplied by a 0.1 valued decay factor.

(**e**) Between many other aspects, the size of the batch utilized heavily dictates how fast the training stage is performed. Considering the datasets size, batches of 500 thousand samples was the size that established the better tradeoff between model performance

and training duration.

(**f**) Another major boost in the model performance could be induced by the way the NN parameters are initialized. Hence, this option was left as the PyTorch's default, i.e., the weights and biases values were instantiated according to the formulation proposed by Kaiming He [47], the Gaussian distribution $\omega_l = G\left(0.0, \sqrt{\frac{2}{n_l}}\right)$, where n is the number of inputs to the particular node l.

In addition to what has already been stated, it was used early stopping in the training, preventing overfitting, unnecessary time and computational resources. It must also be pointed that, in the data preprocessing stage, the features were standardized.

4.3.1 FFNN results

4.3.1.1 System without memory

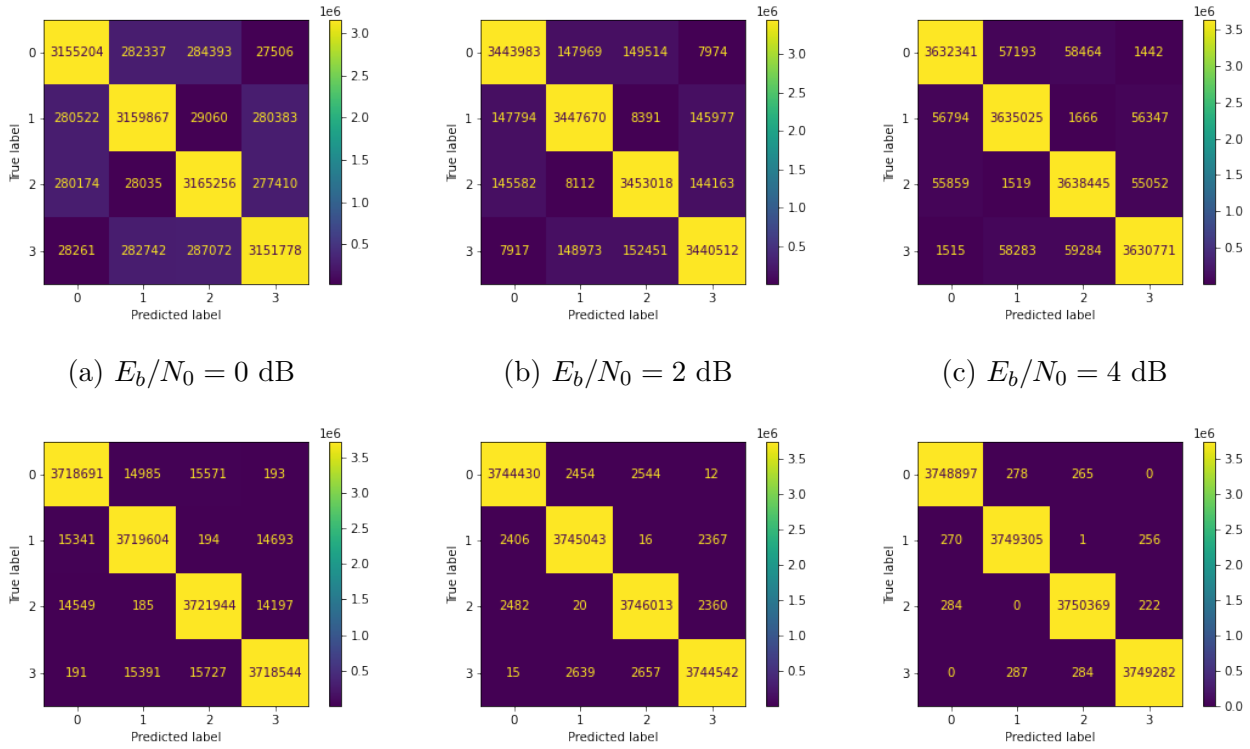
Similarly to conventional demodulators, where the memory characteristics of the Magnitude Modulation algorithm are not taken into account, the performance of a FFNN was tested first with the symbol itself, i.e., only the in-phase and quadrature components were considered as features ($N = 2$).

So, in order to test the influence of the E_b/N_0 (i.e. the energy per information bit to noise power spectral density ratio), characterizing the AWGN channel on the performance of the network, two trainings were performed: one with $E_b/N_0 = 6$ dB and another with $E_b/N_0 = 10$ dB. These choices were based on preliminary tests that had revealed minor differences when the training was performed with values like 0, 2, 4 and 8 dB, leading to the choice of those two E_b/N_0 values. The value $E_b/N_0 = 6$ dB corresponds to an average SNR situation in which noise plus MM causes some errors. On the other hand, $E_b/N_0 = 10$ dB is a scenario where noise is residual, leaving the magnitude modulation as main causer of distortion. In this case, only sporadic errors occur. Both of the mentioned circumstances are depicted in Figure 4.3. Datasets made up of symbols with different E_b/N_0 ratios were also tested, but they provided no benefit.

Relating to the training stage of this model, for an $E_b/N_0 = 6$ dB, it took 63 epochs to train until it was stopped by the early stopping, which was set with a patience of 10 epochs.

In this way, the confusion matrices for each test are presented in Figure 4.7, Table 4.3 displays the performance metrics derived from them, and the BER curves obtained are those depicted in Figure 4.8. The confusion matrix is a helpful performance measurement tool for

classification tasks in ML. It allows to have a quick perception of the model behavior, by establishing a direct comparison between the predictions of the model (horizontal axis) and the desired outputs, i.e. labels/ground-truth (vertical axis). From this matrices, it is possible to extract the metrics presented in Table 4.3, which allows to evaluate the model performance. Analysing the results, it is clear that, even though the FFNN was trained with MM symbols with an E_b/N_0 of 6 dB, the model can generalize well for all the spectrum considered, performing better as the noise intensity gets lower.



(a) $E_b/N_0 = 0$ dB (b) $E_b/N_0 = 2$ dB (c) $E_b/N_0 = 4$ dB

(d) $E_b/N_0 = 6$ dB (e) $E_b/N_0 = 8$ dB (f) $E_b/N_0 = 10$ dB

Figure 4.7: Confusion matrices for each E_b/N_0 value applied to the test dataset, for the model trained with a sequence with $E_b/N_0 = 6$ dB and without considering any memory.

Table 4.3: Metrics extracted from the previous confusion matrices.

$\mathbf{E_b/N_0}$ (dB)	F1 Score (%)	Specificity (%)	Recall (%)	Precision (%)	Accuracy (%)
0	84.214	94.738	84.214	84.214	84.214
2	91.901	97.3	91.901	91.901	91.901
4	96.911	98.97	96.911	96.911	96.911
6	99.192	99.731	99.192	99.192	99.192
8	99.866	99.955	99.866	99.866	99.866
10	99.986	99.995	99.986	99.986	99.986

As it can be seen from Figure 4.8, there is not much of a difference between the curves for different trainings. Our goal is to get as close as possible to the red curve, which represents the theoretical BER for a hard decoder for signals without MM, i.e., only affected by noise. For an E_b/N_0 ratio in the interval $[0, 4]$, the neural network performs quite well. However, as we continue to reduce the noise, the effect of the MM gets increasingly higher, and the curve starts to follow the theoretical BER for a hard decoder when in presence of MM signals. Even though the result obtained is better than the one from the method currently applied, there is still margin for improvements.

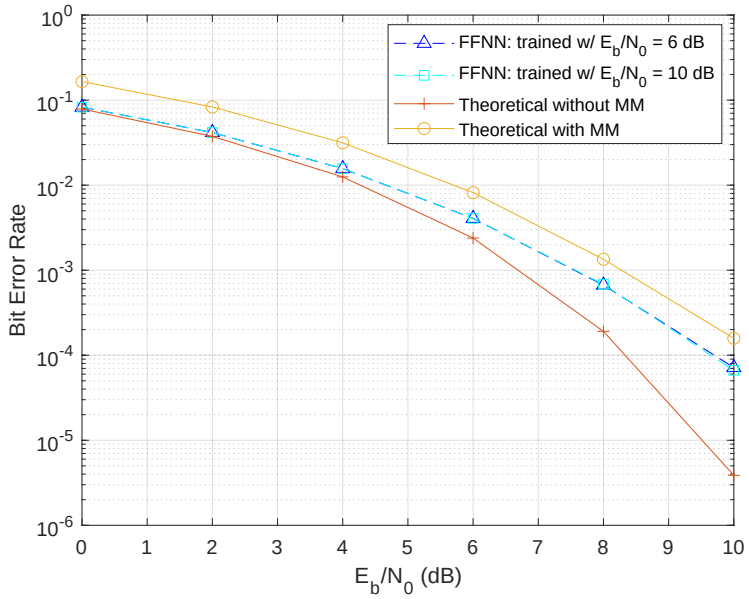


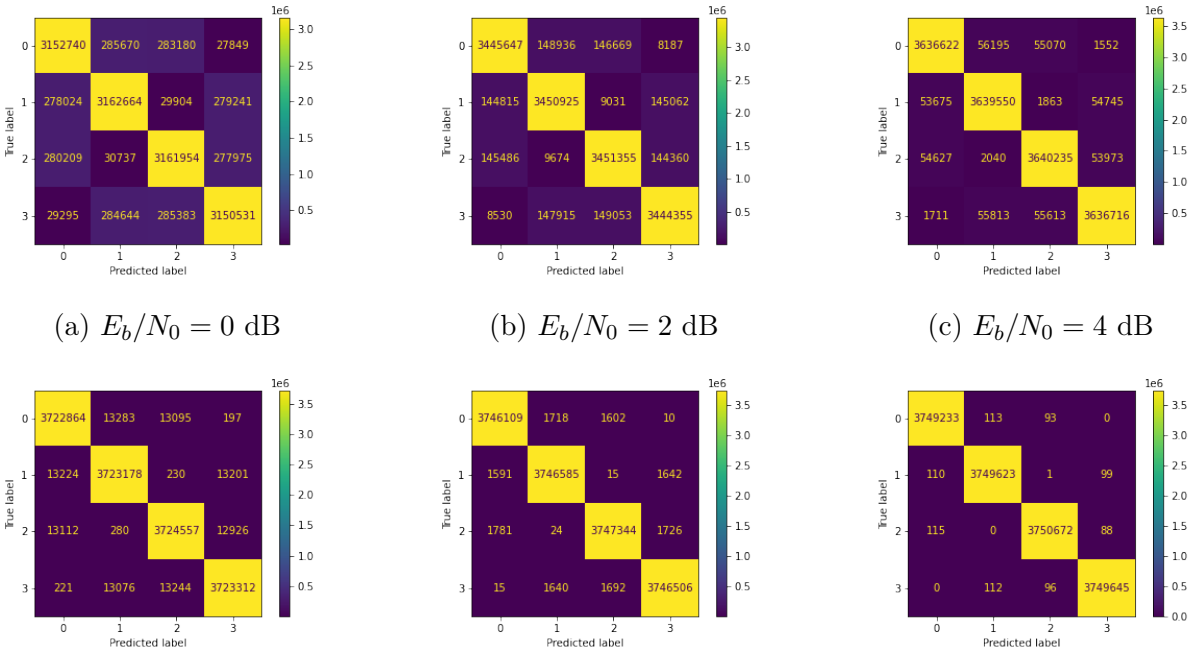
Figure 4.8: BER for a FFNN trained with MM symbols and $E_b/N_0 = 6$ and 10 dB for comparison, without consideration of memory in the system.

4.3.1.2 System with memory

Given that, in the previous section, we were only feeding the FFNN with two features (i.e. the real and imaginary part of the received symbol to be demodulated), it can be considered that it was not enough. So, as the Magnitude Modulation algorithm embeds memory within the transmitted signal, since it considers the neighbors of each symbol to magnitude modulate in order to determine the MM coefficient to apply to it, we trained the FFNN-based model considering this memory of the system, starting first with two neighbors.

As in 4.3.1.1, the model was once trained with symbol sequences with an $E_b/N_0 = 6$ dB, during 35 epochs, but this time with a dataset comprising six features, in-phase and quadrature components of each MM symbol, y_i, to be demodulated, and its right and left neighbors, y_{i+1} and y_{i-1}, and the target being the original symbol (without MM and noise), i.e. s_i. From 4.3.1.1, it was possible to conclude that training the neural network with $E_b/N_0 = 10$ dB did not bring any advantage, as reducing the noise gives less generality to the dataset.

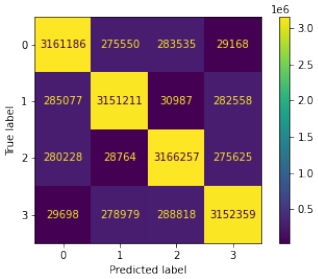
Observing the following Figure 4.9, as well as the Table 4.4, it is noticeable that, in fact, by inducing memory, consequently giving more features to the DNN to learn, the performance can be enhanced and the BER can be reduced even more.



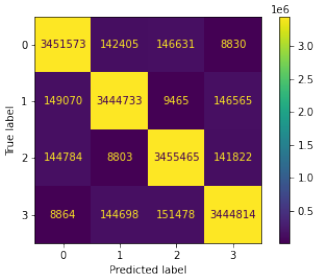
(a) $E_b/N_0 = 0$ dB (b) $E_b/N_0 = 2$ dB (c) $E_b/N_0 = 4$ dB

(d) $E_b/N_0 = 6$ dB (e) $E_b/N_0 = 8$ dB (f) $E_b/N_0 = 10$ dB

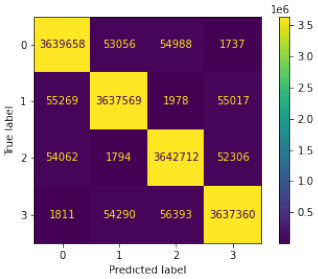
Figure 4.9: Confusion matrices for each E_b/N_0 value applied to the test dataset, for the model trained with a sequence with $E_b/N_0 = 6$ dB and considering a 3-symbol window.



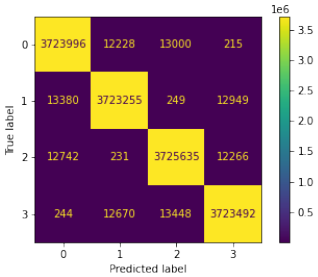
(a) $E_b/N_0 = 0$ dB



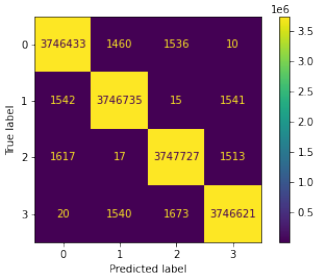
(b) $E_b/N_0 = 2$ dB



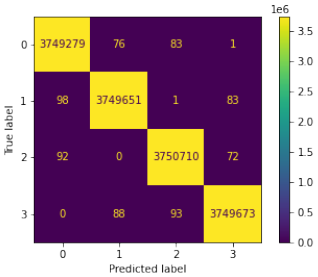
(c) $E_b/N_0 = 4$ dB



(d) $E_b/N_0 = 6$ dB

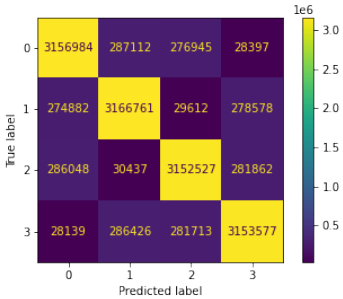


(e) $E_b/N_0 = 8$ dB

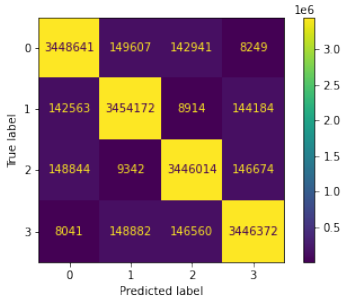


(f) $E_b/N_0 = 10$ dB

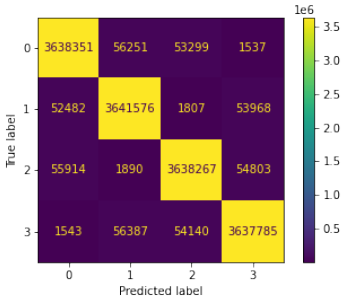
Figure 4.10: Confusion matrices for each E_b/N_0 value applied to the test dataset, for the model trained with a sequence with $E_b/N_0 = 6$ dB and considering a 5-symbol window.



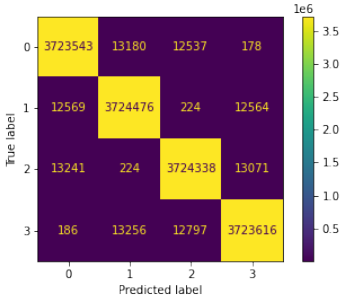
(a) $E_b/N_0 = 0$ dB



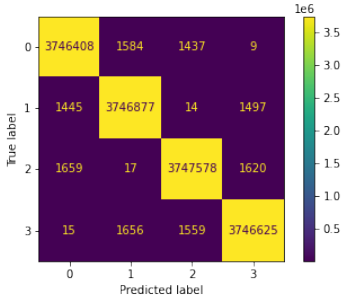
(b) $E_b/N_0 = 2$ dB



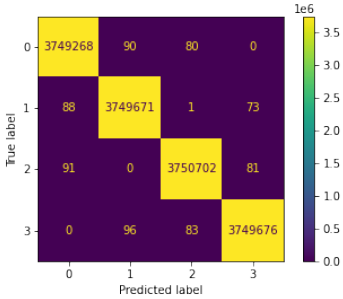
(c) $E_b/N_0 = 4$ dB



(d) $E_b/N_0 = 6$ dB



(e) $E_b/N_0 = 8$ dB



(f) $E_b/N_0 = 10$ dB

Figure 4.11: Confusion matrices for each E_b/N_0 value applied to the test dataset, for the model trained with a sequence with $E_b/N_0 = 6$ dB and considering a 7-symbol window.

Table 4.4: Metrics extracted from the confusion matrices of Figures 4.9, 4.10, and 4.11.

Memory Window Size	**E_b/N_0 (dB)**	**F1 Score (%)**	**Specificity (%)**	**Recall (%)**	**Precision (%)**	**Accuracy (%)**
3	**0**	84.186	94.729	84.186	84.186	84.186
	2	91.949	97.316	91.949	91.949	91.949
	4	97.021	99.001	97.021	97.021	97.021
	6	99.293	99.764	99.293	99.293	99.293
	8	99.293	99.764	99.293	99.293	99.293
	10	99.994	99.998	99.994	99.994	99.994
5	**0**	84.2	94.736	84.2	84.2	84.2
	2	91.977	97.326	91.977	91.977	91.977
	4	97.049	99.016	97.049	97.049	97.049
	6	99.309	99.77	99.309	99.309	99.309
	8	99.917	99.972	99.917	99.917	99.917
	10	99.995	99.998	99.995	99.995	99.995
7	**0**	84.199	94.733	84.199	84.199	84.199
	2	91.977	97.326	91.977	91.977	91.977
	4	97.04	99.013	97.04	97.04	97.04
	6	99.306	99.769	99.306	99.306	99.306
	8	99.917	99.972	99.917	99.917	99.917
	10	99.995	99.998	99.995	99.995	99.995

Looking to find if augmenting the window size, i.e., the number of neighbors, would translate into a more accurate decoding, we considered a 5-symbol window, inducing a ten-feature dataset with the same ground-truth as in the previous case. The training lasted for 30 epochs and, analysing Figure 4.10 and Table 4.4, it is clear that there is an improvement relative to the previous cases. This was the best BER curve achieved for this DL model and, probably, by spending more time fine tuning the model hyperparameters, these results could be slightly improved.

As a last attempt to study the effect of memory on the performance of the model, a 7-symbol window was used for training. This time, the model took 35 epochs to train. It is perceivable from the confusion matrices of Figure 4.11 and their correspondent metrics in Table 4.4 that there were no major advantages in the use of a larger memory window. The BER of the tested scenarios so far are combined in the following section for comparison.

4.3.1.3 Comparison

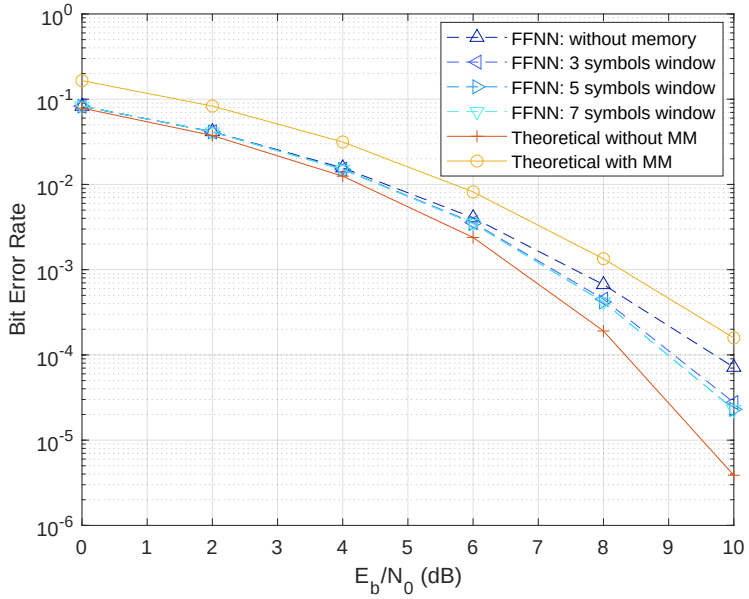


Figure 4.12: Combination of all the BER curves achieved with a FFNN-based model, trained with MM symbols with $E_b/N_0 = 6$ dB and considering different symbol' window sizes.

Gathering all the results exposed at the previous sections, it is noticeable that taking into consideration the memory of the system translates into a gain. However, increasing the symbol window size barely makes a difference: as it can be seen in Figure 4.12, there is a small improvement from a 3 to 5 window size, but from 5 to 7 it is unnoticeable.

4.4 Convolutional Neural Network model

The results obtained with the FFNN-based model fell a little short of expectations. This could be due to the model's simplicity, as a FFNN is the entry base model in the ANN subject.

Seeking to improve them, another well-known NN in classification tasks was employed, the Convolutional Neural Network (CNN) [48]. This network architecture is extensively used in image classification. Nonetheless, it has demonstrated to be effective in other tasks such as time series classification [49], which is why it was worth a try in this study.

So, following a network structure somewhat similar to the one presented in [49], the architecture presented in Figure 4.13 was the one that led to the finest results. This model is fully detailed in Appendix A.2.

Even though CNNs can perform 1D convolutions, as pointed out previously in Section 3.6, which would be the obvious choice for this application, we opted to transform the symbol window into a 2D layout, as illustrated in Figure 4.14. This opens up a wider range of options for model tuning. The symbol window was then reshaped from $1 \times 2N$ to $2 \times N$, N being the size of the window considered.

Analysing the network structure, following the input layer is the feature extraction part, which counts with two *conv2D* layers (Section 3.6), each of them followed by an Exponential Linear Unit (ELU) activation [50] and a *max pooling* layer. Adding dropout [51] showed up to be effective. The details on the mentioned layers and other options made to the model are presented in Table 4.5 and justified afterwards. Regarding the classification part of the network, it counts with three hidden layers.

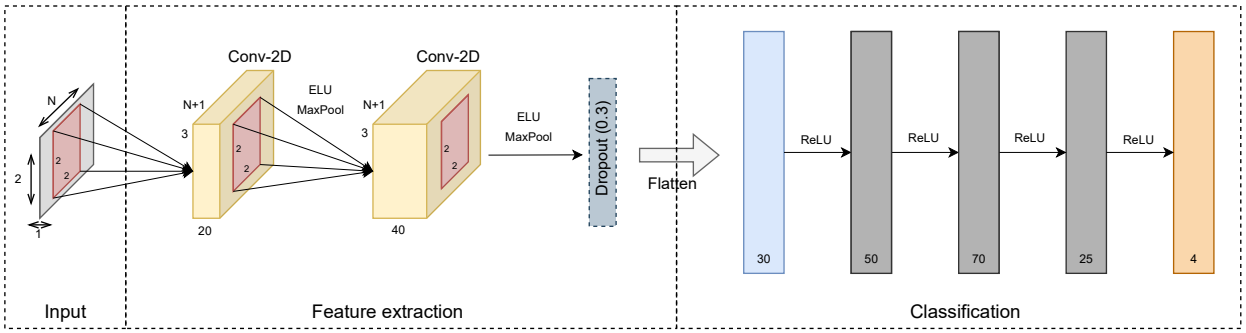


Figure 4.13: Convolutional Neural Network architecture developed. N represents the window of MM and noisy symbols being considered.

$Re(y_{n-(N-1)/2})$	...	$Re(y_n)$	...	$Re(y_{n+(N-1)/2})$
$Im(y_{n-(N-1)/2})$	...	$Im(y_n)$	...	$Im(y_{n+(N-1)/2})$

Figure 4.14: Data format at the entry of the network, considering a window size of N symbols.

Table 4.5: Layers' details, options made to the CNN-based model constitution and hyper-parameter values.

(a)	**Conv2D**	Kernel size = (2,2)
		Stride = 1
		Padding = zeros (1,1)
(b)	**ELU**	Alpha = 1.0
(c)	**MaxPool**	Kernel size = (2,2)
		Stride = 1
		No padding
(d)	**Dropout**	Probability = 0.3
(e)	**Optimization Function**	Adam
(f)	**Batch size**	100000

Next, is presented a list giving further insight and justifications for the options made for the model and presented in Table 4.5.

(**a**) Due to the small size of the input given to the network, there was not much of an option in terms of the convolution kernel size to be applied and to employ padding, in this case with zeros.

(**b**) At the first iterations of this model, ReLU was the AF utilized at the feature extraction part of the network. However, the results achieved were on pair with the ones obtained with the FFNN. Changing to the ELU activation function (Section 3.3) contributed to improve the accuracy. This choice was due as well to the stated in [50], where the advantages of this AF were shown, not only in the generalization capability of the model, but also in speeding up the training stage.

(**c**) The options to the *max pooling* layer follow the considerations already made at point (**a**).

(**d**) The introduction of dropout is usually made in cases were overfitting problems occur. Even though that was not the case, zeroing some elements before the flatten with a probability of 0.3 proved to be helpful.

(**e**) In this model, there was no visible advantage in using AMSGrad, contrarily to the FFNN-based one, reason why Adam was kept.

(**f**) Being computationally heavier than the FFNN-based model, a smaller batch size was needed in order to keep the training duration acceptable.

The remaining options and hyperparameter values were kept as indicated in Table 4.2.

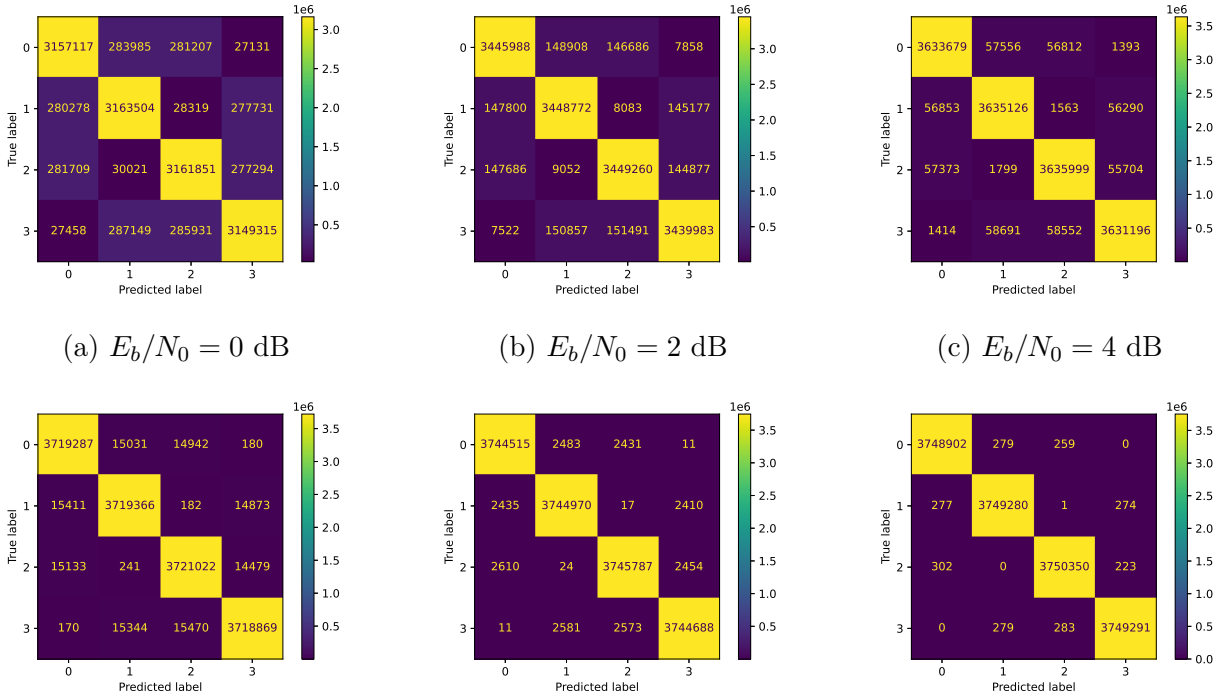
4.4.1 CNN Results

4.4.1.1 System without memory

From Section 4.3.1, it was noticeable that training the network without considering the neighbors (further and previous) of the symbols to demodulate, led to poor classification results, which translates into a BER curve far from what was desired. Anyhow, to establish a comparison between the CNN and FFNN-based models, similar training conditions were tested.

So, giving as input just the symbol itself, which results are shown in Figure 4.15 and Table 4.6, allows to see that no improvements were obtained in comparison with the ones for the FFNN-based model (Figure 4.7 and Table 4.3): this conclusion is translated into the curve shown in Figure 4.16. It should be noted that, since training the previous model with symbols having $E_b/N_0 = 10$ dB was worthless, only trainings with 6 dB were performed for this model. It should also be mentioned that the training went for 50 epochs.

Once more, this proves that giving as input to a model just two learning features to learn, in this specific study case, is insufficient.



(a) $E_b/N_0 = 0$ dB (b) $E_b/N_0 = 2$ dB (c) $E_b/N_0 = 4$ dB

(d) $E_b/N_0 = 6$ dB (e) $E_b/N_0 = 8$ dB (f) $E_b/N_0 = 10$ dB

Figure 4.15: Confusion matrices for each E_b/N_0 value applied to the test dataset, for the CNN-based model trained not considering memory.

Table 4.6: Metrics extracted from the confusion matrices of Figure 4.15.

$\mathbf{E_b/N_0}$ **(dB)**	**F1 Score** (%)	**Specificity** (%)	**Recall** (%)	**Precision** (%)	**Accuracy** (%)
0	84.212	94.737	84.212	84.212	84.212
2	91.893	97.298	91.893	91.893	91.893
4	96.907	98.969	96.907	96.907	96.907
6	99.19	99.73	99.19	99.19	99.19
8	99.866	99.955	99.866	99.866	99.866
10	99.985	99.995	99.985	99.985	99.985

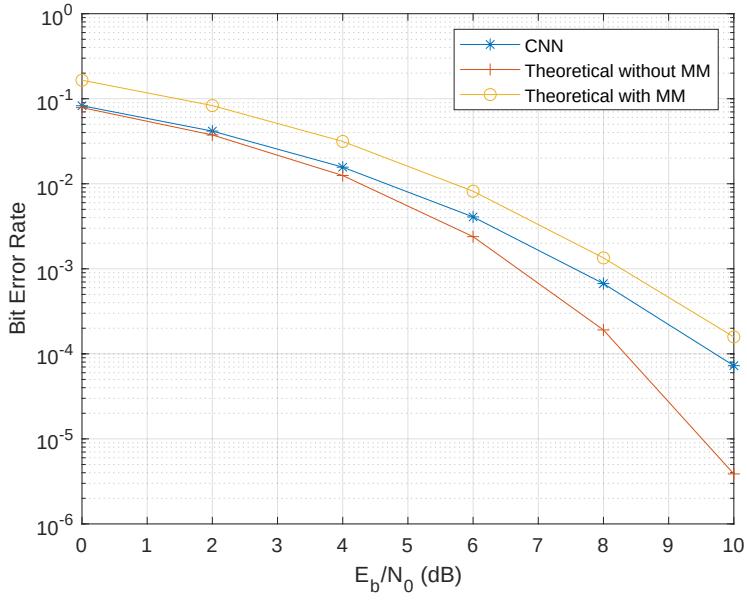
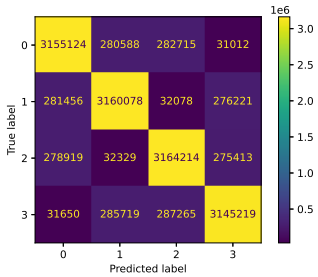


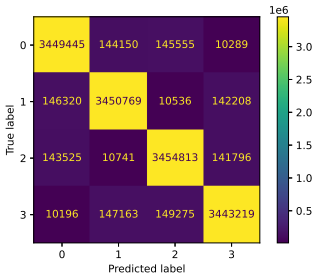
Figure 4.16: BER for a CNN trained with MM symbols and $E_b/N_0 = 6$ dB, without consideration of memory in the system.

4.4.1.2 System with memory

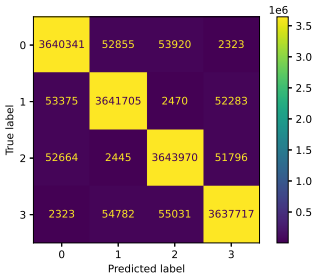
The CNN-based model, similarly to what have been made in Section 4.3.1.2, was tested with different symbol input sizes, in order to determine the influence of the memory in the system. As intended, being a more complex model than the FFNN-based one, the following Figures 4.17, 4.18, and 4.19 and Table 4.7 prove that there is a considerable improvement in the performance, which results in the BERs of Figure 4.20.



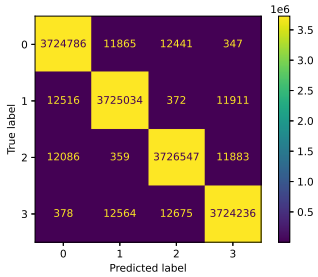
(a) $E_b/N_0 = 0$ dB



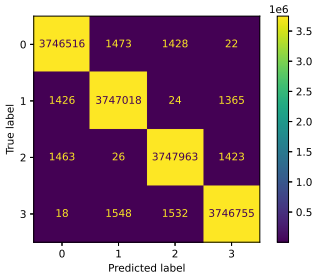
(b) $E_b/N_0 = 2$ dB



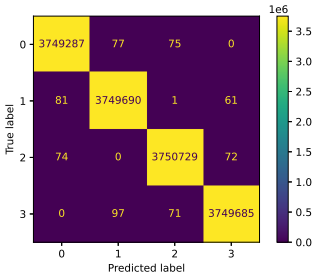
(c) $E_b/N_0 = 4$ dB



(d) $E_b/N_0 = 6$ dB

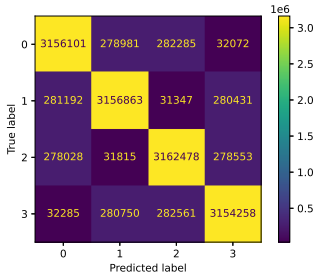


(e) $E_b/N_0 = 8$ dB

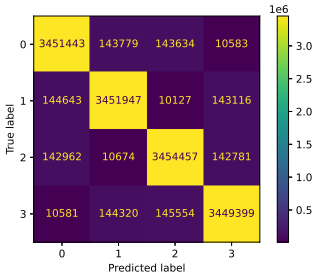


(f) $E_b/N_0 = 10$ dB

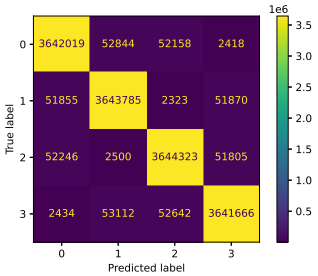
Figure 4.17: Confusion matrices for each E_b/N_0 value applied to the test dataset, for the model trained considering a memory window of 3 symbols.



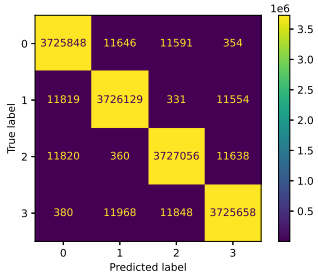
(a) $E_b/N_0 = 0$ dB



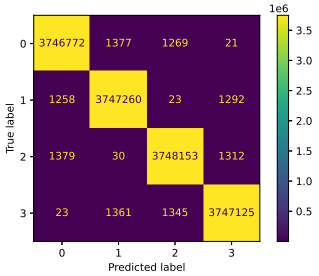
(b) $E_b/N_0 = 2$ dB



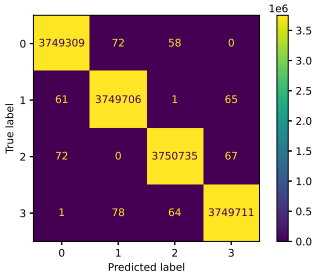
(c) $E_b/N_0 = 4$ dB



(d) $E_b/N_0 = 6$ dB

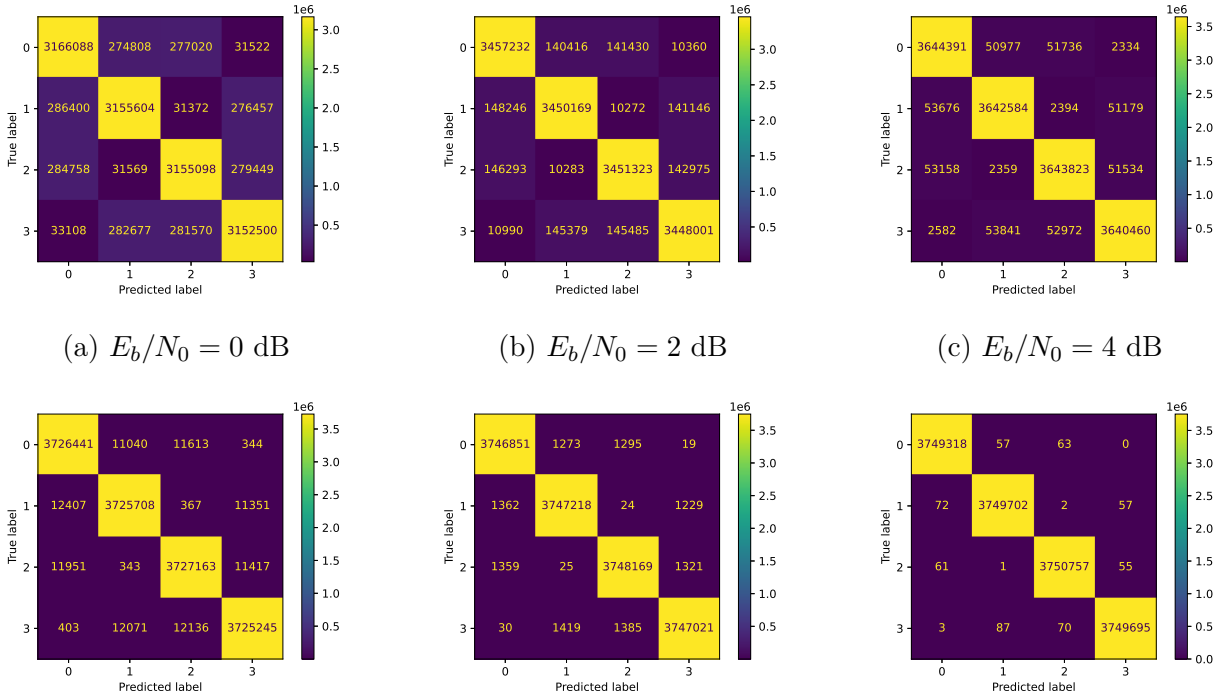


(e) $E_b/N_0 = 8$ dB



(f) $E_b/N_0 = 10$ dB

Figure 4.18: Confusion matrices for each E_b/N_0 value applied to the test dataset, for the model trained considering a memory window of 5 symbols.



(a) $E_b/N_0 = 0$ dB (b) $E_b/N_0 = 2$ dB (c) $E_b/N_0 = 4$ dB

(d) $E_b/N_0 = 6$ dB (e) $E_b/N_0 = 8$ dB (f) $E_b/N_0 = 10$ dB

Figure 4.19: Confusion matrices for each E_b/N_0 value applied to the test dataset, for the model trained considering a memory window of 7 symbols.

Table 4.7: Metrics extracted from the confusion matrices of Figures 4.17, 4.18, and 4.19.

Memory Window Size	**E_b/N_0 (dB)**	**F1 Score (%)**	**Specificity (%)**	**Recall (%)**	**Precision (%)**	**Accuracy (%)**
3	**0**	84.165	94.721	84.165	84.165	84.165
	2	91.988	97.329	91.988	91.988	91.988
	4	97.092	99.03	97.092	97.092	97.092
	6	99.337	99.779	99.337	99.337	99.337
	8	99.922	99.974	99.922	99.922	99.922
	10	99.996	99.999	99.996	99.996	99.996
5	**0**	84.198	94.733	84.198	84.198	84.198
	2	92.048	97.349	92.048	92.048	92.048
	4	97.145	99.048	97.145	97.145	97.145
	6	99.365	99.788	99.365	99.365	99.365
	8	99.929	99.976	99.929	99.929	99.929
	10	99.996	99.999	99.996	99.996	99.996
7	**0**	84.196	94.731	84.196	84.196	84.196
	2	92.045	97.348	92.045	92.045	92.045
	4	97.142	99.047	97.142	97.142	97.142
	6	99.364	99.788	99.364	99.364	99.364
	8	99.928	99.976	99.928	99.928	99.928
	10	99.996	99.999	99.996	99.996	99.996

4.4.1.3 Comparison

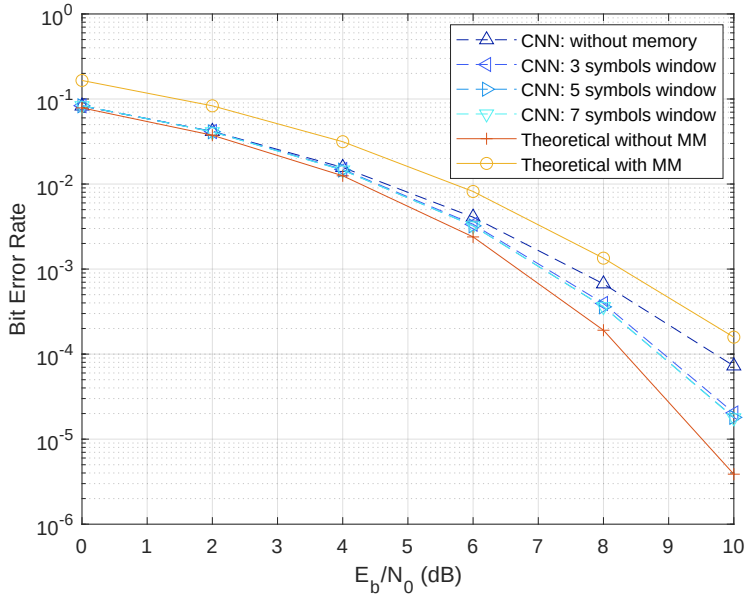


Figure 4.20: Combination of all the previous results shown for a CNN trained with MM symbols with $E_b/N_0 = 6$ dB.

As it can be seen from the previous figure, the BER curves arising from this model are very similar to the ones presented in Figure 4.12. However, in the next section, the results of both models are analysed in greater detail.

4.5 Model performance comparison

Although the BER curves seem equal to the ones provided by the FFNN-based model, when compared side by side in detail in Figure 4.21, it is noticeable that the CNN performed better. As a quantitative measure, for an $E_b/N_0 = 10$ dB, the BER suffered an improvement of 5×10^{-6} (AU) from the FFNN to the CNN model. Considering a high-speed data transfer like, for example, 1 Mb/s, it would translate in 5 error bits per second.

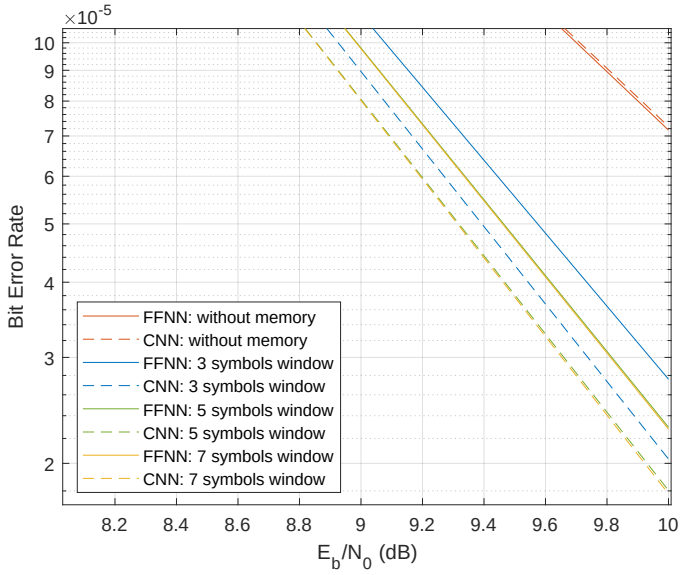


Figure 4.21: Excerpt of the BER curves of both models for comparison, when the networks were trained with MM symbols with $E_b/N_0 = 6$ dB.

Having explored the capabilities of two common state-of-the-art neural networks, the FFNN allowing to enquire if a simpler model would be enough for the task intended and the CNN inducing a higher complexity in the system, and knowing the limitations in terms of richness of the data, it was clear that, without bringing something new to the study, it would be difficult to achieve an even better demodulation BER. This prompt us to the next chapter.

5 MM signals detection with multi-model NN-based systems

The models presented previously proved that it is possible to mitigate the MM effect in the demodulation stage. However, since there is still margin for improvement, a new approach was taken, whose premise was based on reverting the MM system.

At an early stage of the work, unwittingly, the MM coefficients of the symbols were being fed to the model, together with both the components of the symbols, accordingly with the window being employed. The BER obtained was staggering, since it matched almost perfectly the theoretical BER for the 4-QAM constellation's symbols, as presented in Figure 5.1. The problem with this was: the receptor only gets the MM symbols with AWGN, y, so, unless the MM coefficients were estimated on the basis of y, there was no way to achieve such results. So, the next step was precisely to come up with a system composed of complementary models in order to estimate the MM coefficients prior to the symbol classification.

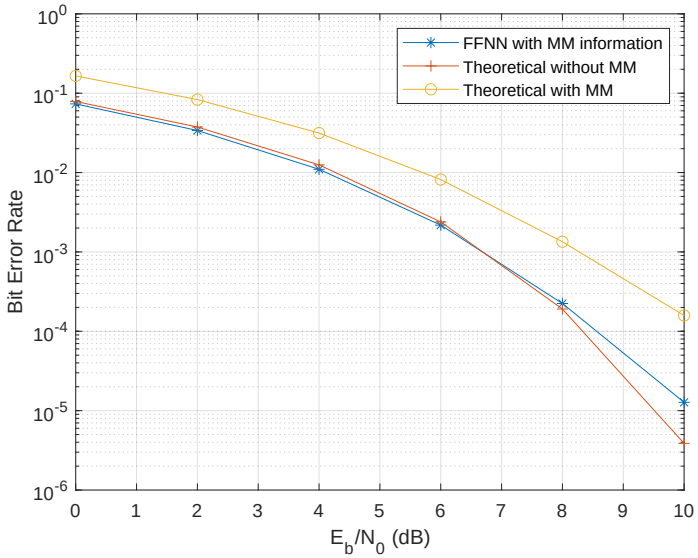


Figure 5.1: Demodulation BER of a FFNN-based model trained with MM symbols with AWGN of $E_b/N_0 = 6$ dB and with the addition of the MM coefficients in the dataset, considering a 3-symbol window.

5.1 Bidirectional LSTM model

In addition to the models presented in the last chapter, a BLSTM-based one was implemented for the classification task intended.

Widely used at Natural Language Processing (NLP) related fields like hand-writing and speech recognition [52, 53], the RNNs have proven to be effective due to their architecture, specially the LSTM (Section 3.7). Attending to the current state of the art, complexity and computational cost were not taken into consideration here, since Transformers [54] have proven to lead on those topics. However, the BLSTM allows to consider not just the past symbols, but, by reversing the sequence, take into account the future ones, which is somewhat similar to what happens at the MPMM technique.

Firstly introduced for framewise phoneme classification [55], the bidirectional version of an LSTM outperformed both the RNN and its bidirectional version, as well as the "vanilla" LSTM, showing improvements in performance and on the number of training epochs. So, it was a typology worth trying.

Regarding the model employed, its architecture is as the one presented in Figure 5.2, and the setup options made are presented at Table 5.1.

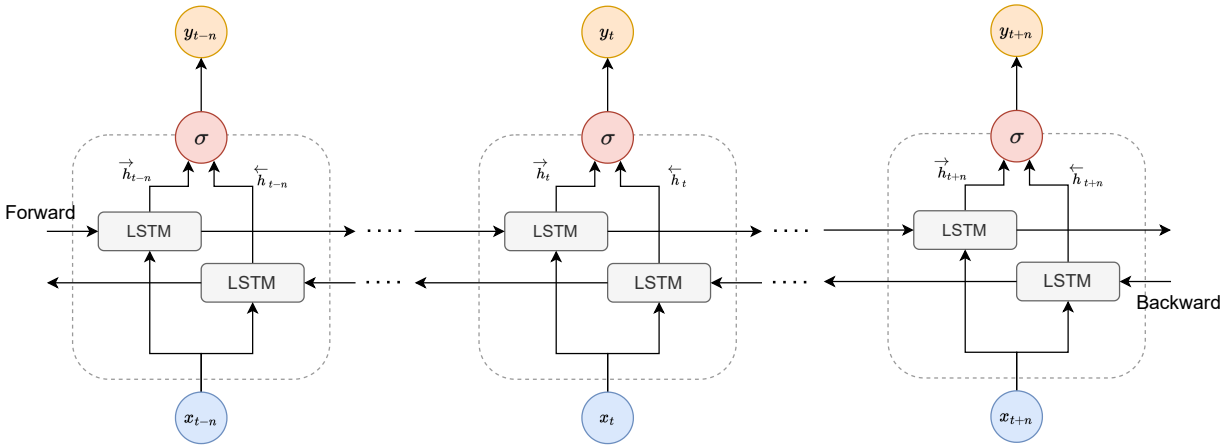


Figure 5.2: Bidirectional Long Short-Term Memory Recurrent Neural Network architecture developed. x represent the input sequence at a specific instant t, h is the LSTM module output, σ represent the activation function and y is the layer output.

Table 5.1: Choices made for the BLSTM-based model constitution and hyperparameter values.

(a)	**Hidden size**	128
(b)	**Activation Function**	ELU
(c)	**Loss Function**	Cross Entropy
(d)	**Optimization Function**	Adam
(e)	**Learning Rate**	$10^{-2}/10^{-3}$
(f)	**Batch Size**	64
(g)	**Weight & Bias initialization**	Default

The model options were similar to the other models in that they followed current state-of-the-art functions and chose specific values such as (a) and (f) in Table 5.1 that resulted in better results. The difference between the previous models is visible in relation to the parameter (f).

Contrarily to the FFNN and CNN-based models, this model was applied with a new paradigm, which was to predict the MM coefficient used for a symbol. Thus, beyond considering the symbols, the MM coefficients were also added to the datasets. Due to the symbol sequences nature, these coefficients follow a far from uniform distribution, leading to unbalanced datasets. To avoid this, sample selection had to be done, drastically reducing the dataset size, which is why the batch size hyperparameter is significantly lower than in the other models.

5.2 First approach: Cascaded Feed-Forward

The first approach taken was to consider the feed-forward cascaded system represented in Figure 5.3, composed by two DL estimators. The first estimator, A, receives the MM symbols with added white noise, y, and outputs an estimation of the corresponding MM factors, $\widehat{m}$. Then, the second model, B, takes those estimations and the received symbols y to classify them into the original ones, $\widehat{s}$.

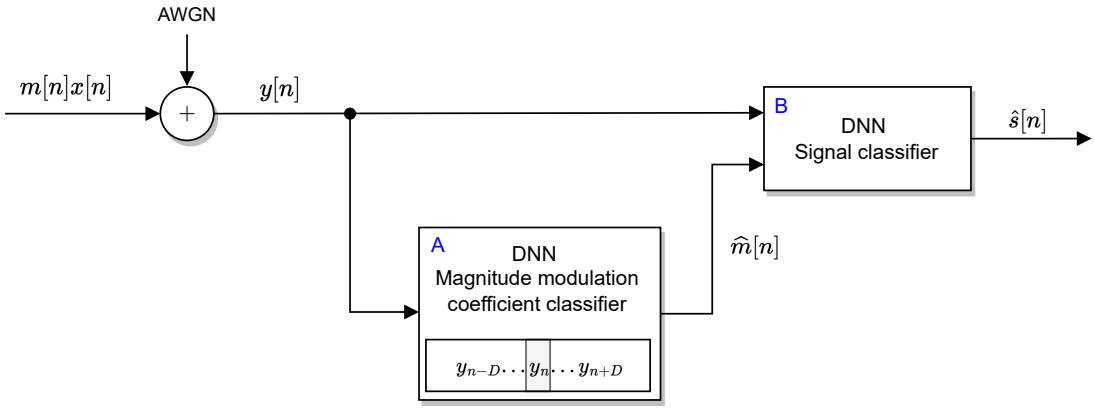


Figure 5.3: System composed by two DNNs: A - takes as input the symbols received and outputs an estimation of the MM coefficients class; B - accepts both the symbols and respective estimated MM coefficients and predicts the original symbols.

Getting into more detail, the model A consisted of the BLSTM RNN described in Section 5.1. In order to keep it fairly simple and the most computationally inexpensive possible, this was the one that led to the best classification among the evaluated ones. An important decision made was one regarding the matter of the quantization applied to the MM coefficients. Pointing out that, regarding the magnitude modulated symbol's sequence in study, the coefficients were comprised in the interval $[0.39, 1.00]$, with a distribution as presented in Figure 5.4. However, in order to enable a DL model to perform the classification of the MM coefficients, it was necessary to discretize this interval.

Two concerns stand out from the following figure:

1. In order to maintain a class balanced dataset, no matter what kind of quantization is applied, there will be a loss of samples for the training phase because most of the MM coefficients stand between 0.8 and 0.87, approximately. With such an uneven distribution, the most populated classes have to be trimmed, leading to sample loss.

2. Having established that the MM coefficients can take an almost continuous range of values, how much can they be discretized without arming the system?

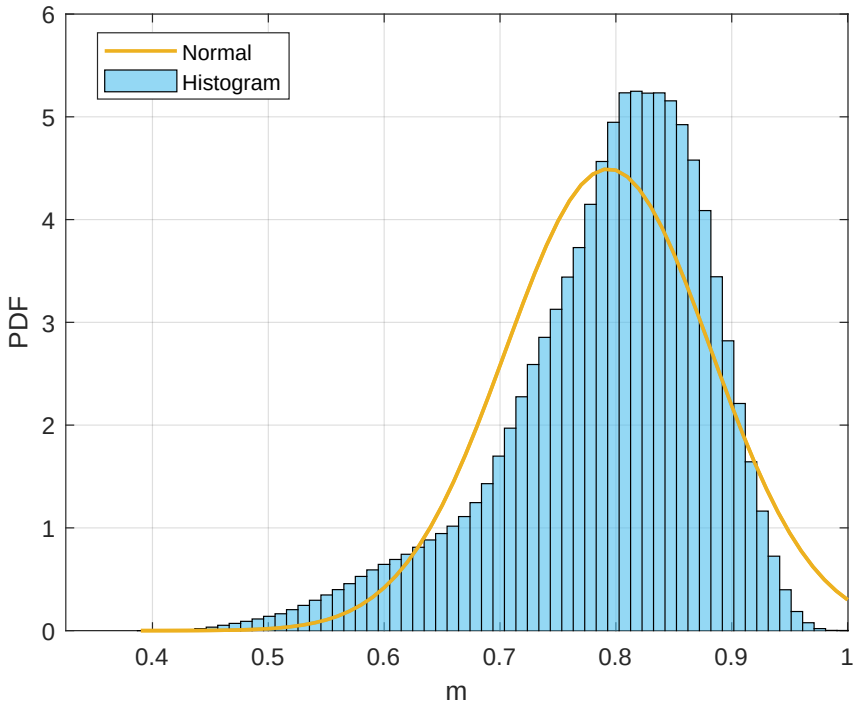


Figure 5.4: Distribution of the 15 million MM factors, m, rounded to two decimals, corresponding to the training sequence symbols.

Regarding model B, it was based on the FFNN described in Section 4.3, with some modifications in the hyperparameters. As demonstrated, although simple, this model led to the BER curve at Figure 5.1, which is an optimal result.

So, seeking to understand how much the accuracy of the MM factors matters to the model while comprehending their importance as features, B was tested considering different levels of quantization (from 3 to 8 bits) within the provided MM factors, using Lloyd's algorithm [56, 57]. Surprisingly, model B proved to be very robust. Figure 5.5 shows that there is only a minor performance loss when using a 3-bit quantization, which gives an answer to the second concern. On the other hand, a 3-bit quantization is equivalent to have 8 classes, allowing to simplify model A, since, generally, a higher class classification problem requires a more complex model.

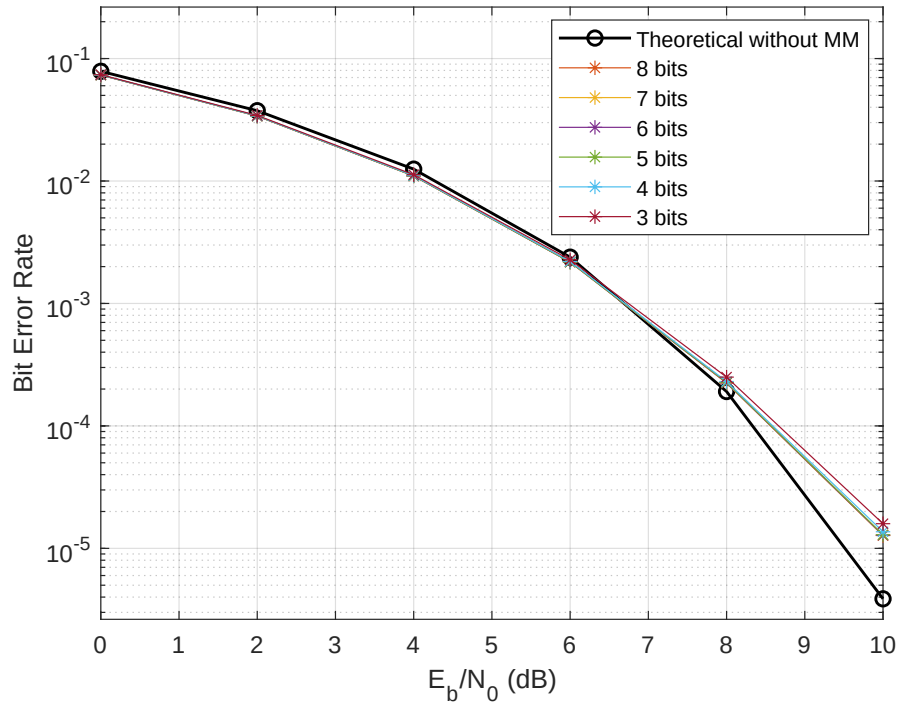


Figure 5.5: BER achieved from the system when the model B was provided with MM factors quantized with different levels, for comparison. Training was performed considering a window of 3 symbols with $E_b/N_0 = 6$ dB.

Thus, the system was tested making use of a window of 7 magnitude modulated and noisy symbols.

5.2.1 First approach results

Knowing the indifference of model B to the quantization applied to the MM coefficients, we opted for the simplest path, training model A as an 8-class classifier. Despite the efforts to gather a model with a promising performance, the one that led to the best results was based on the BLSTM RNN described in Section 5.1, achieving an overall accuracy of 60.2%.

Nonetheless, despite the training dataset having been balanced, the symbols' sequence of the testing dataset gathers a highly unevenly distributed MM factors sequence, as exemplified before, reason why using the accuracy as an evaluation metric is not ideal. So, accordingly with Branco *et al.* [58], the Area Under the Curve (AUC)-Receiving Operating Characteristics (ROC) curve is the best suited metric to deal with imbalanced data, ROC being a probability curve, and AUC the degree of separability (i.e., representing the model capability of distinguish classes). Having an AUC close to 1, indicates that the model can differentiate almost perfectly between classes. Since this is an 8-class model, the one *vs*

all methodology is employed, which means that each ROC curve refers to one class being compared against all the other seven. For example, having an AUC equal to 0.8 for class 0, means that there is a 80% chance that the model differs class 0 from classes 1 to 7.

So, the system was tested and its performance evaluated, leading to the results of Figures 5.6 and 5.7.

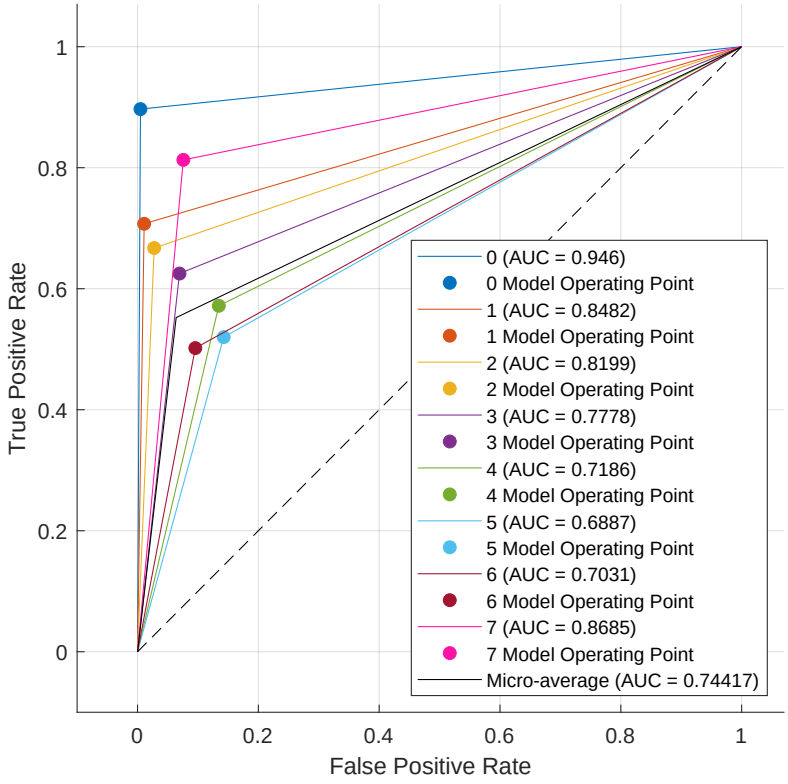


Figure 5.6: AUC-ROC for each class of the results from the BLSTM when tested for an $E_b/N_0 = 6$ dB.

As the previous graph shows, the performance of the model is quite promising, with an average AUC rounding 0.74, meaning that, in average, the model can distinguish one class from the remaining ones with 74% certainty. However, it performs worse at the predominant classes, which proved to compromise model B, since the BER curve achieved (Figure 5.7) was far from what is desired (Figure 5.1).

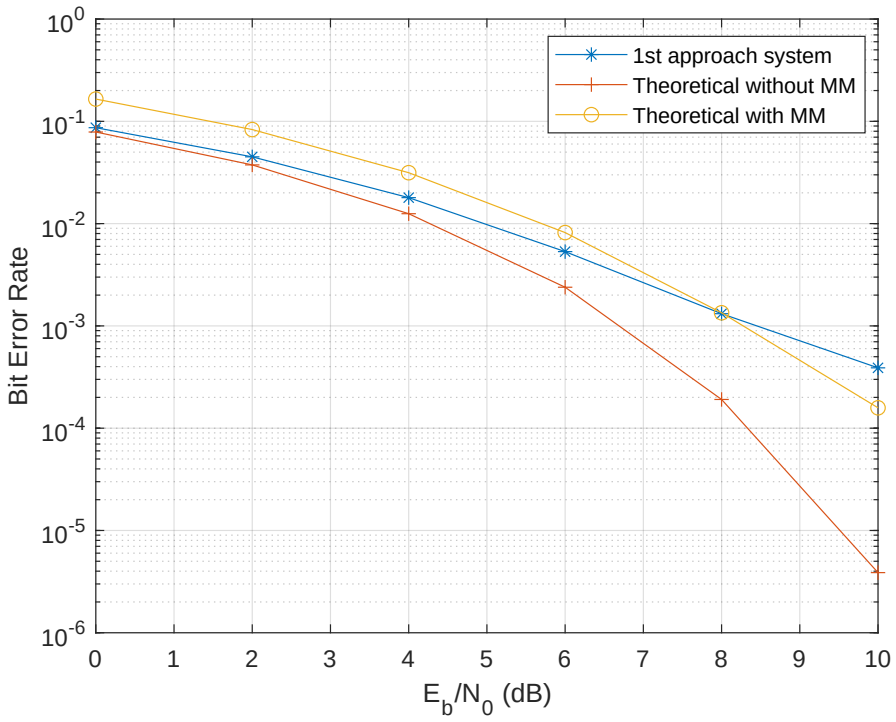


Figure 5.7: BER of the system when feeding the predictions of the MM coefficients from model A (BLSTM RNN as classifier) to the symbol classifier, B (FFNN as classifier), considering a 7-symbol window.

Knowing that the symbol classifier model was resistant to the quantization applied to the MM factors, led to conclude that, while the MM factors classifier (model A) performance was reasonable, it was insufficient to meet the expectations. As it can be seen, for lower levels of noise, the system loses completely its advantage relatively to traditional demodulators.

5.3 Second approach: Cascaded Feed-Forward with a Twist

The poor performance of the first multi-model system prompted us to consider the options available to achieve the desired BER while mitigating the MM effect.

As previously stated, classification and regression are two of the most common applications of machine learning algorithms. Classification is the process of associating a set of characteristics with a specific type or class. Regression is about approximating a given sample to a value, like a function that, given a certain input, produces a correspondent output. Until now, only classification was taken into account. Nonetheless, reminding that, for the configuration exposed, the magnitude modulation factors of the sequences being used stand

at the interval $[0.39, 1.00]$, there is a possibility to make use of a DNN for the regression task of estimating the MM factor inherent to each symbol.

So, a twist was given to the system developed for the first approach, resulting in the one depicted at Figure 5.8, in which model A is now a regressor instead of a classifier.

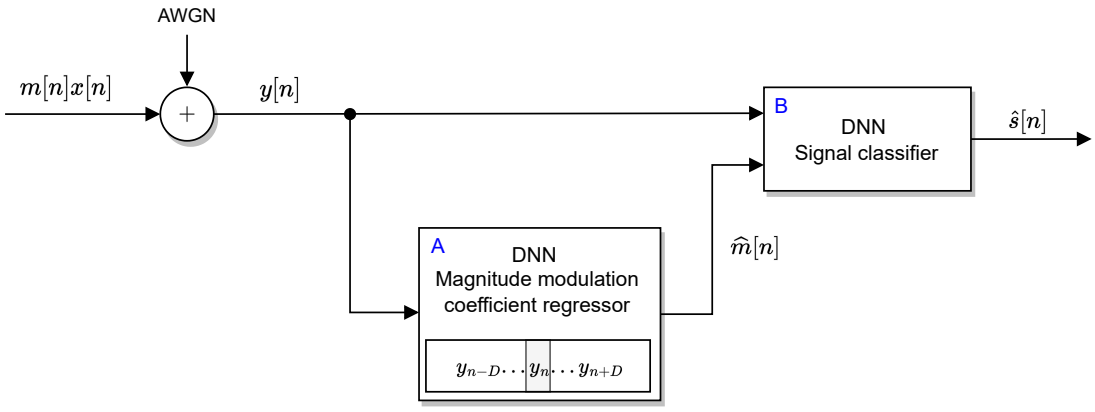


Figure 5.8: System composed by two DNNs: A - takes as input the symbols received and outputs an estimation of the MM coefficient values; B - accepts both the symbols and respective estimated MM coefficients and predicts the original symbols.

Thus, the FFNN-based model introduced earlier in Section 4.3 was used to this task, with the appropriate amendments, replacing the model A in Figure 5.3. Particularly, the output layer now has just one neuron, and the loss function employed was the L1/ Mean Absolute Error (MAE).

5.3.1 Second approach results

In order to compare the performance of both models A, this is, the classification one employed in the first approach and the regression model employed in the second approach, the MSE metric was used. Figure 5.10 shows the discrepancy between the model performing classification with the BLSTM RNN (full blue line), and the one performing regression with the FFNN (full red line). In relation to the real coefficients, the regression model can get much closer to them. However, this is still not enough to achieve the BER presented at Figure 5.1, as shown in Figure 5.9.

Figure 5.10 explains why the system using the regressor, although performing better than the one of the first approach with the classifier predicting the MM coefficients, do not reach the optimal result. Having in consideration the value kept by the horizontal pink line that gives the MSE between the real coefficients used and their correspondent quantized

(with 3 bits) values, we can observe that the red line never reaches it. So, the robustness guaranteed by the quantization algorithm and comproved in Figure 5.5 is not achievable with the regression model.

Nonetheless, the red line shows low MSE values for symbols with little noise. To emphasize this point, the model's behavior with no-noise symbols was tested, resulting in the value represented by the horizontal green line. As it can be seen, this value gets pretty close to the one achieved with the quantized true coefficients, implying that this system could perform well for channels with little to no noise interference.

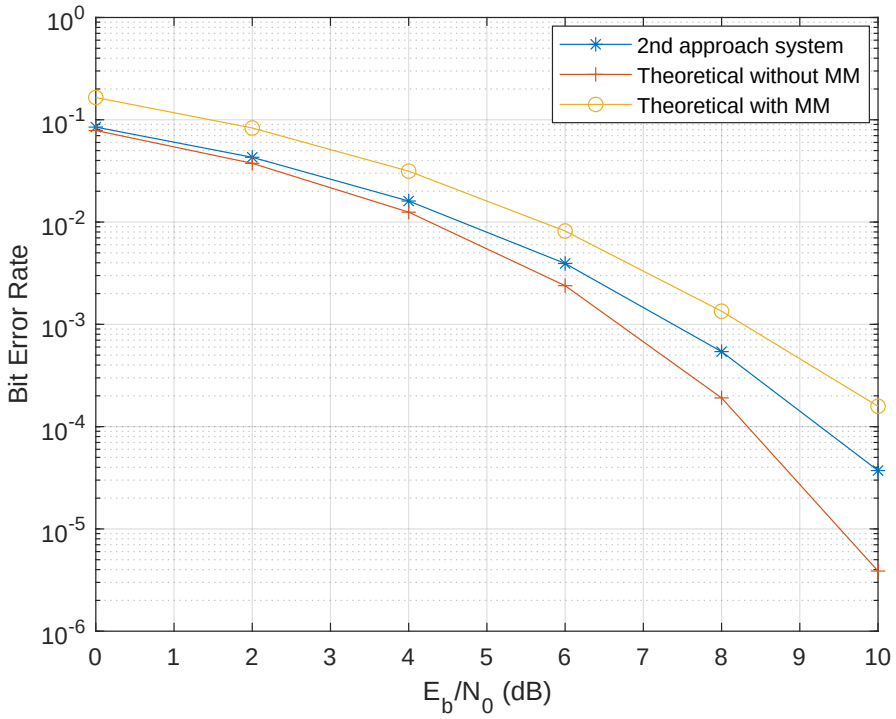


Figure 5.9: BER of the system when feeding the predictions of the MM coefficients from model A (FFNN as regressor) to the symbol classifier, B (FFNN as classifier), considering a 7-symbol window.

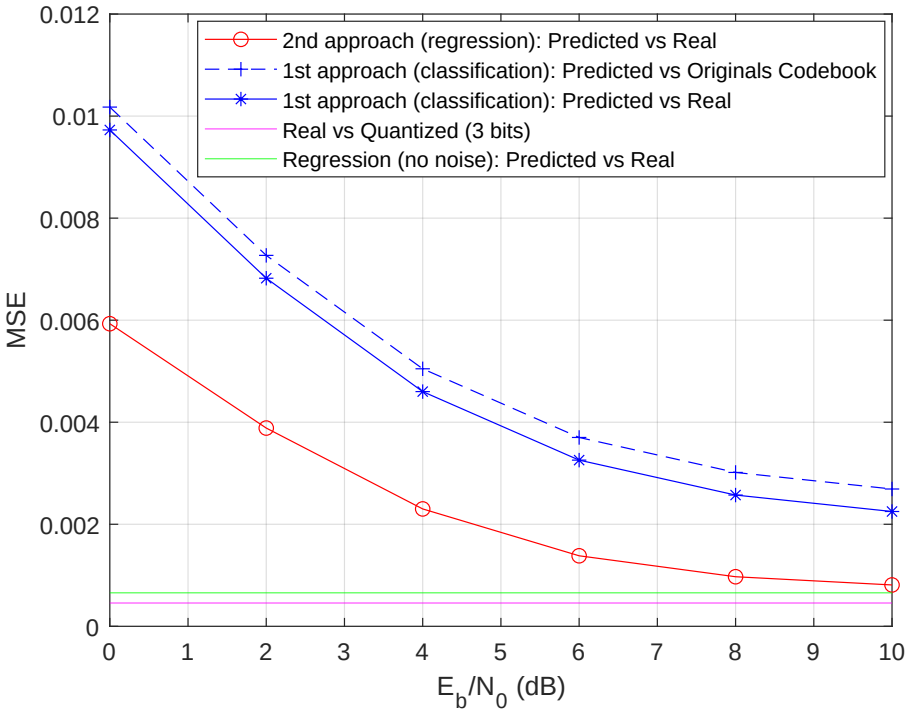


Figure 5.10: MSE plots for comparison of the regression and classification-based models.

Although the system had been trained with symbols with an $E_b/N_0 = 6$ dB, as a test, we presented to it symbols just magnitude modulated, i.e., without noise, and it performed flawlessly.

So, this brought us to the final approach, as the problem seems to reside on the MM factors.

5.4 Third approach: Classification with Iterative Refinement

Since neither of the two previous multi-model systems led to the expected results, outperforming the CNN-based model performance presented in Section 4.4, another one was thought of.

The study started by tuning a model capable of demodulating MM symbols with the addition of AWGN. Furthermore, it was noticed that introducing the MM factors as features could result in a significant gain. This took the study through a new path in which models were combined in order to, previously to the symbol demodulation, predict the MM factor that was used. The thought approach was: we have a model that works fairly well by itself, giving it just the received symbols, and another that, in theory, behaves greatly

with the addition of the magnitude modulation factors, showing no BER performance loss when compared to a non-MM transmission; so, why not use the MPMM directly to get the coefficients from the predicted symbol sequence and give them as input to the second model?

Thus, the third system employed was the one displayed in Figure 5.11. This system relies on iterations, with the goal of iteratively refining the results. First, the received corrupted symbols, y, pass through model A, which outputs a prediction of the original symbols, $\widehat{s}_1$. This sequence is then given to the MPMM, from which the MM factors $\widehat{a}_1$ are extracted and followed to a second iteration. Here, model B takes y and $\widehat{a}_1$ as input and outputs a new prediction for the original symbols $\widehat{s}_2$ and the cycle is repeated until an optimal demodulation BER is achieved, in theory.

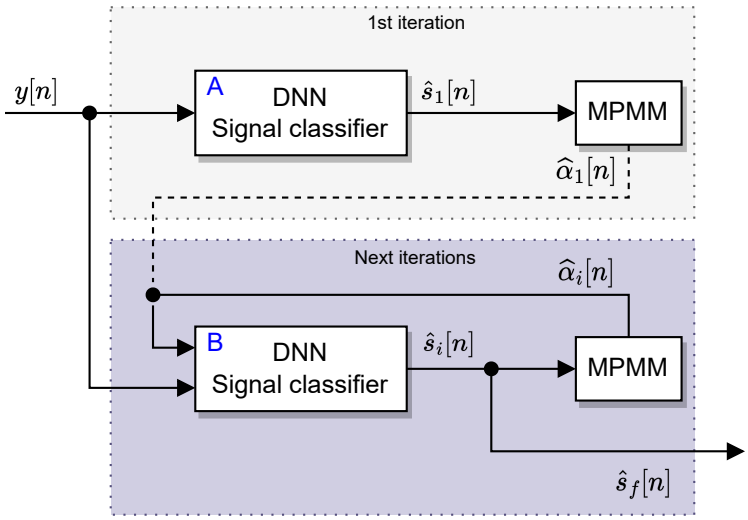


Figure 5.11: Iterative system employing two models, A and B, and the MPMM technique.

As pointed previously, among the models tested, the one containing the CNN (Section 4.4) led to the best BER, which is why it was chosen for model A. On the other hand, the FFNN-based model (Section 4.3) behaved greatly when the magnitude modulation factors were introduced as features, making it model B.

5.4.1 Third approach results

With this newly introduced system, the results obtained were more promising, as it can be seen from Figure 5.12: indeed there is a significant gain with respect to the prior one (Figure 5.9). Nonetheless, it remains a little bit short to the ideal BER and the difference between iterations is unnoticeable.

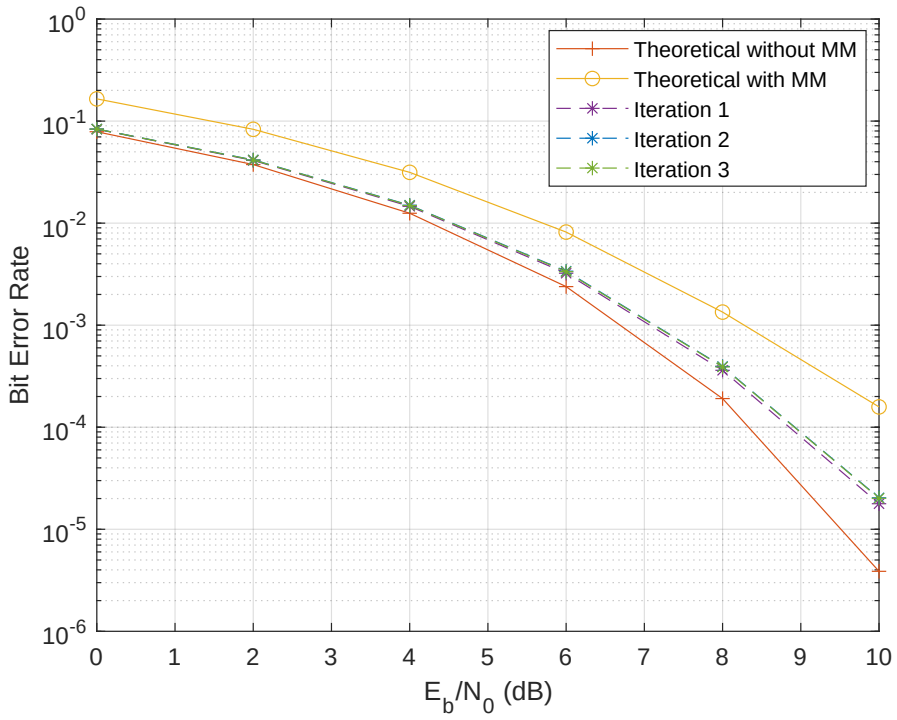


Figure 5.12: BER of the third system, considering a 7-symbol window and comprising three iterations.

The previous figure leads us to the conclusion that there was no significant benefit to iteratively refining the models' predictions, as no significant BER performance was achieved in relation to the previous approaches. The similarities with the results of the CNN-based model (Section 4.4.1) are due to the fact that the first iteration of this approach was based on the outputs of such a model.

Thus, despite the efforts made to take the BER of a DNN-based demodulator a step further, improving even more the BER in comparison with the one provided by the conventional demodulators, the best attained result still was the one achieved with the CNN-based model. Nonetheless, the main idea to take the results a step further was found, in theory, and the BER already achieved is very promising.

6 Conclusions

The aim of this study was to find a neural network-based DL model able to extinguish the increase in the demodulation error introduced by the Magnitude Modulation technique.

To create a fully capable model and system, it was necessary to first understand the data, specifically the symbols' sequences, how the AWGN channel affected them, and how the extent of the features affected the model's ability to learn. This study began by determining the best SNR to be used (and therefore the variance of the AWGN applied to the sequences) in order for a model to learn them and generalize the demodulation for sequences affected by other noise intensities other than the one for which the model was trained. So, after experimenting with a wide range of E_b/N_0 ratios and even composing datasets with mixtures of symbol sequences having different SNRs, we concluded that $E_b/N_0 = 6$ dB was the ratio most representative of the entire spectrum considered. It provides a good margin for errors and is a good compromise between high and low level noise.

The next step was to comprehend the role of memory in the system. It is well known that the more the characteristics of the data, provided their representativeness, the better the model will learn. We were able to define a window of symbols to input to the neural network-based detector by introducing a buffer, intending to discern the effect of the MM symbols window size in the performance. Here, we came to the conclusion that there is a noticeable improvement on the detection performance (i.e. lower BER) when increasing the memory length up to 5 symbols, while additional gains for higher memory lengths were almost imperceptible. These findings come aligned with the starting assumption of the work that MM introduces memory on the transmitted signal, and that we can leverage this to improve detection. Nonetheless, following this evaluation, all the models were trained and tested over a 7-symbol interval, bringing us to the main task: finding the best suited model.

Since this is meant to be applied in real-time operating systems, speed is a non-functional requirement to have in mind, which means we want models as simple and efficient as possible. So, among the models built using current state-of-the-art neural networks, the one that performed best was a CNN-based one. We also attempted to extend the results by

composing systems that combine multiple models, such as predicting the magnitude modulation coefficients prior to the symbol classification stage. However, and despite the fact that for a genie receiver (that is, having knowledge of exact MM factors) it works in theory, the three different approaches tested to predict the MM factors failed to estimate these with the required precision. So, in practice, we were never able to achieve the desired BER coincident to the one of the demodulation when in the presence of non-magnitude modulated symbols.

In conclusion, this study demonstrated that it is possible to use DL algorithms to significantly improve the demodulation BER compared to state-of-the-art techniques and mitigate the tradeoff established by the MM technique initially, while also leaving open the possibility of taking the results a step ahead.

6.1 Future Work

This thesis's overall goal was accomplished. However, there are always topics that can be explored in greater depth.

Some future considerations include studying the performance of DL algorithms in other types of transmission channels, such as Rayleigh. As a proof-of-concept, only a 4-QAM constellation was considered in this study, so bigger constellations should be taken into account. Other topic subject to further analysis is the amplification scenario. Here, was considered an operation in the linear region of the amplifier. Would be interesting to bring non-linear amplification scenarios to the equation. One last relevant approach might be consider not just neural network-based models but probabilistic-based ones, for example.

7 Bibliography

[1] S.L. Miller and R.J. O'Dea. Peak power and bandwidth efficient linear modulation. *IEEE Transactions on Communications*, 46(12):1639–1648, 1998.

[2] Scott L. Miller. Radio with peak power and bandwidth efficient modulation, Apr 1997.

[3] M. Ambroze, M. Tomlinson, and G. Wade. Magnitude modulation for small satellite earth terminals using qpsk and oqpsk. *IEEE International Conference on Communications, 2003. ICC '03.*, 3:2099–2103 vol.3, 2003.

[4] Marco Gomes, Vitor Silva, Francisco Cercas, and Martin Tomlinson. Real-time lut-less magnitude modulation for peak power control of single carrier rrc filtered signals. In *2009 IEEE 10th Workshop on Signal Processing Advances in Wireless Communications*, pages 424–428, 2009.

[5] Marco Gomes, Francisco Cercas, Vitor Silva, and M. Tomlinson. Polyphase magnitude modulation for peak power control. 01 2009.

[6] Marco Gomes, Vitor Silva, Francisco Cercas, and Martin Tomlinson. Power efficient back-off reduction through polyphase filtering magnitude modulation. *IEEE Communications Letters*, 13(8):606–608, 2009.

[7] Marco Gomes, Francisco Cercas, Vitor Silva, and M. Tomlinson. Magnitude modulation for vsat's low back-off transmission. *Communications and Networks, Journal of*, 12:544–557, 12 2010.

[8] M. Gomes. Magnitude modulation for peak power control in single carrier communication systems. 2010.

[9] Mei Chen and Oliver M. Collins. Trellis pruning for peak-to-average power ratio reduction. *CoRR*, abs/cs/0511037, 2005.

[10] Makoto Tanahashi and Hideki Ochiai. Near constant envelope trellis shaping for psk signaling. *IEEE Transactions on Communications*, 57(2):450–458, 2009.

[11] P. Bento, A. P. S. Silva, M. Gomes, R. Dinis, and V. Silva. A simplified and accurate ber analysis of magnitude modulated m-psk signals. *IET Communications*, 13(10):1443–1448, June 2019.

[12] A. Tomlinson, A. Ambroze, and G. Wade. Power and bandwidth efficient modulation and coding for small satellite communication terminals. In *2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333)*, volume 5, pages 2943–2946 vol.5, 2002.

[13] Marco Gomes, Francisco Cercas, Vitor Silva, and Martin Tomlinson. Efficient m-qam transmission using compacted magnitude modulation tables. In *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*, pages 1–5, 2008.

[14] Marco Gomes, Vitor Silva, Francisco Cercas, and M. Tomlinson. Low back-off 16-apsk transmission using magnitude modulation and symbol quantization. pages 229 – 233, 11 2008.

[15] Fredric J. Harris. *Multirate Signal Processing for Communication Systems*, pages i–lvii. 2021.

[16] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[17] Lukas-Valentin Herm, Jonas Wanner, Franz Seubert, and Christian Janiesch. I don't get it, but it seems valid! the connection between explainability and comprehensibility in (x) ai research. In *ECIS*, 2021.

[18] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[19] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

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

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[22] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016.

[23] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[25] Felix Gers and Jurgen Schmidhuber. Recurrent nets that time and count. volume 3, pages 189 – 194 vol.3, 02 2000.

[26] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[27] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated recurrent neural networks. 08 2015.

[28] Tony J. Rouphael. Chapter 3 - common digital modulation methods. In Tony J. Rouphael, editor, *RF and Digital Signal Processing for Software-Defined Radio*, pages 25–85. Newnes, Burlington, 2009.

[29] K Deergha Rao. *Channel coding techniques for wireless communications*. Springer, 2015.

[30] Marco Gomes, Rui Dinis, Vitor Silva, Francisco Cercas, and M. Tomlinson. Error rate analysis of m-psk with magnitude modulation envelope control. *Electronics Letters*, 49:1184–1186, 08 2013.

[31] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[32] H. Leung and S. Haykin. The complex backpropagation algorithm. *IEEE Transactions on Signal Processing*, 39(9):2101–2104, 1991.

[33] Simone Scardapane, Steven Van Vaerenbergh, Amir Hussain, and Aurelio Uncini. Complex-valued neural networks with non-parametric activation functions, 2018.

[34] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[36] Věra Kůrková and Marcello Sanguineti. Classification by sparse neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2746–2754, 2019.

[37] Christopher Y. Liu and Ric A. Romero. Deep neural network detection for pulsed radar-embedded m-psk communications. In *2020 17th European Radar Conference (EuRAD)*, pages 238–241, 2021.

[38] Jorge Torres, Fidel Ernesto Montero, and Joachim Habermann. Demodulators for bfsk signals based on matched filters: A survey. *Revista Telemática*, 15:62–72, 08 2016.

[39] A. Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

[40] Chaity Banerjee, Tathagata Mukherjee, and Eduardo Pasiliao. An empirical study on generalizations of the relu activation function. In *Proceedings of the 2019 ACM Southeast Conference*, ACM SE '19, page 164–167, New York, NY, USA, 2019. Association for Computing Machinery.

[41] MTCAJ Thomas and A Thomas Joy. *Elements of information theory*. Wiley-Interscience, 2006.

[42] Arash Sangari and William Sethares. Convergence analysis of two loss functions in soft-max regression. *IEEE Transactions on Signal Processing*, 64(5):1280–1288, 2016.

[43] Anna Sergeevna Bosman, Andries P. Engelbrecht, and Mardé Helbig. Visualising basins of attraction for the cross-entropy and the squared error neural network loss functions. *CoRR*, abs/1901.02302, 2019.

[44] Yangfan Zhou, Xin Wang, Mingchuan Zhang, Junlong Zhu, Ruijuan Zheng, and Qingtao Wu. Mpce: A maximum probability based cross entropy loss function for neural network classification. *IEEE Access*, 7:146331–146341, 2019.

[45] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[46] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[48] Sakshi Indolia, Anil Kumar Goswami, S.P. Mishra, and Pooja Asopa. Conceptual understanding of convolutional neural network- a deep learning approach. *Procedia Computer Science*, 132:679–688, 2018. International Conference on Computational Intelligence and Data Science.

[49] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, 2017.

[50] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[51] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[52] Alex Graves, Marcus Liwicki, Horst Bunke, Jürgen Schmidhuber, and Santiago Fernández. Unconstrained on-line handwriting recognition with recurrent neural networks. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[53] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 273–278, 2013.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[55] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4, 2005.

[56] Joel Max. Quantizing for minimum distortion. *IRE Trans. Inf. Theory*, 6:7–12, 1960.

[57] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[58] Paula Branco, Luis Torgo, and Rita Ribeiro. A survey of predictive modelling under imbalanced distributions, 2015.

A Appendix

A.1 Deep Neural Networks Architectures

```
FFNN Architecture: NeuralNet(
  (fc1): Linear(in_features=14, out_features=20, bias=True)
  (fc2): Linear(in_features=20, out_features=50, bias=True)
  (fc3): Linear(in_features=50, out_features=70, bias=True)
  (fc4): Linear(in_features=70, out_features=35, bias=True)
  (fc5): Linear(in_features=35, out_features=4, bias=True)
  (relu): ReLU()
)
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
Linear: 1-1                              300
Linear: 1-2                              1,050
Linear: 1-3                              3,570
Linear: 1-4                              2,485
Linear: 1-5                              144
ReLU: 1-6                                —
==========================================================================
Total params: 7,549
Trainable params: 7,549
Non-trainable params: 0
==========================================================================
```

Listing A.1: FFNN architecture in detail for the maximum window size employed, $N = 14$.

```
CNN Architecture: nnModel(
  (cnn_layers): Sequential(
    (0): Conv2d(1, 10, kernel_size=(2, 2), stride=(1, 1),
    padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=1, padding=0,
    dilation=1, ceil_mode=False)
    (3): Conv2d(10, 20, kernel_size=(2, 2), stride=(1, 1),
    padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=(2, 2), stride=1, padding=0,
    dilation=1, ceil_mode=False)
    (6): Dropout(p=0.5, inplace=False)
  )
  (linear_layers): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=280, out_features=30, bias=True)
    (2): ReLU()
    (3): Linear(in_features=30, out_features=50, bias=True)
    (4): ReLU()
    (5): Linear(in_features=50, out_features=70, bias=True)
    (6): ReLU()
    (7): Linear(in_features=70, out_features=25, bias=True)
    (8): ReLU()
    (9): Linear(in_features=25, out_features=4, bias=True)
  )
)

==========================================================================

Layer (type:depth-idx)                        Param #

==========================================================================

Sequential: 1-1                                 --
    Conv2d: 2-1                                 50
```

```
    ReLU: 2-2                                --
    MaxPool2d: 2-3                           --
    Conv2d: 2-4                              820
    ReLU: 2-5                                --
    MaxPool2d: 2-6                           --
    Dropout: 2-7                             --
Sequential: 1-2                              --
    Flatten: 2-8                             --
    Linear: 2-9                              8,430
    ReLU: 2-10                               --
    Linear: 2-11                             1,550
    ReLU: 2-12                               --
    Linear: 2-13                             3,570
    ReLU: 2-14                               --
    Linear: 2-15                             1,775
    ReLU: 2-16                               --
    Linear: 2-17                             104
=================================================================
Total params: 16,299
Trainable params: 16,299
Non-trainable params: 0
=================================================================
```

Listing A.2: CNN architecture in detail for the maximum window size employed, $N = 14$.