Inatel

Instituto Nacional de Telecomunicações

Dissertação de Mestrado

Link Adaptation Techniques Based on Inner Receiver Statistics and Machine Learning

ROBERTO MICHIO MARQUES KAGAMI

OCTOBER / 2023

LINK ADAPTATION TECHNIQUES BASED ON INNER RECEIVER STATISTICS AND MACHINE LEARNING

ROBERTO MICHIO MARQUES KAGAMI

This thesis was presented at Inatel, as part of the requirements for obtaining a Master's Degree in Telecommunications.

ADVISOR: Prof. Dr. Felipe Augusto Pereira de Figueiredo.

Santa Rita do Sapucaí
2023

Kagami, Roberto Michio Marques.
K118l
Link Adaptation Techniques Based on Inner Receiver Statistics and Machine Learning / Roberto Michio Marques Kagami – Santa Rita do Sapucaí, 2023.
114 p.

Orientador: Prof. Dr. Felipe Augusto Pereira de Figueiredo
Dissertação de Mestrado em Telecomunicações – Instituto Nacional de Telecomunicações – INATEL.
Inclui bibliografia e anexo.

1. Link Adaptation 2. Adaptive Coding and Modulation 3. 6G 4. Machine Learning 5. Reliability 6. Mestrado em Telecomunicações. I. Figueiredo, Felipe Augusto Pereira de II. Instituto Nacional de Telecomunicações – INATEL. III. Título.

CDU 621.39

Ficha Catalográfica elaborada pelo Bibliotecário da Instituição
CRB6-2718 – Paulo Eduardo de Almeida

APPROVAL FORM

Dissertação defendida e aprovada em _____/_____/_____, pela comissão julgadora:

Prof. Dr. Felipe Augusto Pereira de Figueiredo
INATEL

Dr. Fabbryccio Akkazzha Chaves Machado Cardoso
CPQD

Prof. Dr. Samuel Baraldi Mafra
INATEL

Coordenador do Curso de Mestrado
Prof. Dr. José Marcos Câmara Brito

"Live as if you were to die tomorrow. Learn as if you were to live forever"

Mahatma Gandhi

Acknowledgments

After graduating, I took a long break from the academic scope. But I often felt the urge to endure deeper into my studies and research, which, although present in my professional life, aroused a desire for improvement and immersion to follow the fast-paced world of telecommunications and its complex technologies. However, I learned that it's never too late to pursue new accomplishments. Luckily, destiny provided me with the conditions to fulfil this intention in harmony with work and the development of new ideas and projects.

So, I want to express my gratitude to the National Institute of Telecommunications for making this aspiration come true. At the same time, I congratulate this institution formed by extremely competent, dedicated, and enthusiastic professionals who have generated, throughout time, so much content, talent, and expertise for the country and abroad. I am thankful to all the Inatel staff who were there every day, supporting me in my pursuit of new knowledge, professional growth, and personal development.

In particular, thanks to my advisor Professor Felipe Augusto Pereira de Figueiredo, who, without any exception, has always been attentive, helpful, and interested in developing, with the best possible quality, a work that requires a lot of effort, knowledge, vocation, patience, and dedication. I also want to express my appreciation to my colleagues at the Telecommunications Reference Center and Professor Luciano Leonel Mendes for their guidance, teachings, collaborations, support, and interest.

To all the friends and family who supported me in the most challenging moments when it was necessary to find extra energy to stay strong towards my goals, I am truly grateful. Posthumously, but always in time, I also dedicate this work to my parents Michio Kagami and Lires Marques, who passed away shortly before I started this journey but were fundamental to my education, construction and evolution as a person.

Roberto Michio Marques Kagami

Table of Contents

Table of Contents **v**

List of Figures **vii**

List of Tables **viii**

List of Abbreviation and Acronyms **ix**

Resumo **xii**

Abstract **xiii**

1 Introduction **1**
- 1.1 Motivation 5
- 1.2 Publications 6
- 1.3 Document Structure 7

2 System Design and Architecture **8**
- 2.1 Top-level Architecture 9
  - 2.1.1 MAC Layer 9
  - 2.1.2 PHY Layer 10
  - 2.1.3 RF Front-end 10
- 2.2 Physical Layer Architecture 10
  - 2.2.1 Hardware Description 12
  - 2.2.2 Software Description 13

3 Elementary Link Status Indicators **18**
- 3.1 Related works 18
- 3.2 Mean Square Error for SINR estimation 20
- 3.3 Modified Mean Square Error Function 20
- 3.4 System Model for Data Analysis 23
- 3.5 Nonlinear regression function 26
- 3.6 Deep learning network description 27
- 3.7 Results 29
- 3.8 Conclusions 35

4 Complementary Link Status Indicators **36**

4.1 Related works 37
4.2 Error Correction Amplitude Indicator 38
4.3 System Model for Data Analysis 39
4.4 Results 44
4.5 Conclusion 46

5 Link Adaptation Control Using Machine Learning 47
5.1 Related works 47
5.2 Link Adaptation Control Topologies 49
5.3 Direct Decision with Standalone Error Correction Amplitude (ECA) . 50
5.4 Direct Decision with Q-Learning 53

6 Conclusions and Future Work 57
6.1 Conclusions 57
6.2 Future Work 58

References 61

I Codes 1
I.1 Deep Neural Network for SINR Estimation 1
I.1.1 Learning Code: QPSK 1
I.1.2 Learning Code: 16-QAM 2
I.1.3 Learning Code: 64-QAM 4
I.1.4 Learning Code: 256-QAM 5
I.2 Deep Neural Network for SINR Estimation -
Inference Block. 7
I.3 Deep Neural Network description 12
I.3.1 DNN for SINR estimation - QPSK modulation. 12
I.3.2 DNN for SINR estimation - 16-QAM modulation. 14
I.3.3 DNN for SINR estimation - 64-QAM modulation. 15
I.3.4 DNN for SINR estimation - 256-QAM modulation. 17
I.4 Machine Learning for Link Adaptation. 19
I.4.1 Direct Decision with Q-Learning 19
I.4.2 Indirect Decision with achievement index output 22
I.4.3 Low complexity DNN description 23
I.4.4 Indirect Decision with Achievement Probability 25

List of Figures

1.1 Mobile Network System Overview. 2

2.1 Top-level system diagram. 9
2.2 Physical Layer transmission diagram. 11
2.3 Physical Layer reception diagram. 11
2.4 GNU Radio system diagram of transmission blocks. 15

3.1 Instant error vector. 21
3.2 Data set generation diagram. 23
3.3 Modified MSE vs. SNR with AWGN channel - QPSK. 24
3.4 Modified MSE vs. SNR with AWGN channel - 16-QAM. 24
3.5 Modified MSE vs. SNR with AWGN channel - 64-QAM. 25
3.6 Modified MSE vs. SNR with AWGN channel - 256-QAM. 25
3.7 Polynomial nonlinear regression. 26
3.8 Modified MSE plus DNN scheme. 26
3.9 Perceptron architecture. 27
3.10 Implemented Deep Learning Neural Network topology. 28
3.11 Biases and weights for for the 256-QAM model. 29
3.12 DNN training (TensorBoard). 29
3.13 QPSK - DNN output. 30
3.14 16QAM - DNN output. 30
3.15 64QAM - DNN output. 31
3.16 256QAM - DNN output. 31
3.17 Proposed function response. 32
3.18 GNU Radio system diagram of reception including DNN block. . . . 33
3.19 Proposed model response for 256-QAM with real-time implementation and DNN retraining. 33
3.20 Real-time Signal-to-Interference-plus-Noise Ratio (SINR) estimation of a 256-QAM constellation. 34

4.1 Breaking point estimation based only in the SINR. 38
4.2 Breaking point estimation based on the inner receiver statistics. 39
4.3 Block diagram for generating the ECA index. 39
4.4 Data collection diagram. 41
4.5 ECA vs. SINR. 41
4.6 Measured SINR with imperfections. 43
4.7 System diagram. 44

4.8 BER using the overestimated SINR (0.5 dB). 45
4.9 BER using SINR with a nonlinear response. 46

5.1 Link adaptation control overall perspective. 49
5.2 Link Adaptation Control scheme based on standalone ECA. 50
5.3 ECA vs. hard BER. 51
5.4 Measured ECA vs. soft BER. 52
5.5 Neural Network topology for ECA normalization. 52
5.6 Normalized Error Correction Amplitude output accordingly to Modulation and Coding Scheme (MCS) index. 53
5.7 Direct decision with Q-learning diagram. 53
5.8 Reward vector: MCS 21 from 20dB to 28 dB. 55
5.9 Q-Learning algorithm response & ECA vs. SINR 56

6.1 Indirect decision: MCS table with achievement probabilities. 59
6.2 Proposed scheme using Deep Q-Learning and Softmax function. . . . 59

List of Tables

2.1 Transceiver’s hardware characteristics. 12
2.2 SDR USRP 2954 - Transmitter specifications. 14
2.3 SDR USRP 2954 - Receiver specifications. 14
2.4 Open-source Libraries. 16
2.5 Software installations for GPU usage. 16

4.1 MCS definition and breaking point SINR, assuming a codeword length of $N = 2048$ bits. 40
4.2 Transmitted bits with overestimated SINR. 45
4.3 Transmitted bits with nonlinear measured SINR. 45

List of Abbreviation and Acronyms

2G	Second Generation of Mobile Networks
3G	Third Generation of Mobile Networks
4G	Fourth Generation of Mobile Networks
5G	Fifth Generation of Mobile Networks
6G	Sixth generation of mobile networks
ACK	Acknowledge
ACM	Adaptive Coding and Modulation
ADC	Analog-to-Digital Converter
AI	Artificial Inteligence
ANN	Artificial Neural Network
API	Application Programming Interface
AVX	Advanced Vector Extensions
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BLER	Block Error Rate
BS	Base Station
CDF	Cumulative Distribution Function
CNN	Convolutional Neural Network
CP	Cyclic Prefix
CPU	Central Processing Unit
CQI	Channel Quality Indicator
CSI	Channel State Information
DAC	Digital-to-Analog Converter
DFT	Discrete Fourier transform
DNN	Deep Neural Network
DPD	Digital Pre-Distortion
DQL	Deep Q-Learning
DRLLA	Deep Reinforcement Learning for Link Adaptation
DSP	Digital Signal Processing
ECA	Error Correction Amplitude
eMBB	Enhanced Mobile Broadband
ERS	Engine Room Status
FCNN	Fully-Connected Neural Network
FEC	Forward error correction
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array

GCC	Gnu Compiler Collection
GFDM	Generalized Frequency Division Multiplexing
GPGPU	General Purpose Graphics Processing Unit
GPP	General Purpose Processors
GPU	Graphics Processing Unit
iFFT	Inverse Fast Fourier Transform
ILLA	Inner Loop Link Adaptation
IoT	Internet of Things
IP	Intellectual Property
IP	Internet Protocol
IPC	Interprocess Communication
KPI	Key Performance Indicators
LA	Link Adaptation
LDPC	Low density parity check
LLR	Log-Likelihood Ratio
LMS	Least Mean Squares
LSTM	Long Short Term Memory
LTE	Long Term Evolution
MAB	Multi-Armed Bandit
MAC	Media Access Control
MCS	Modulation and Coding Scheme
MIMO	Multiple Input Multiple Output
MIT	Massachusetts Institute of Technology
ML	Machine Learning
MMSE	Minimum Mean Square Error
mMTC	Massive Machine Type Communications
MSE	Mean Square Error
NACK	Non-Acknowledge
NN	Neural Network
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
OLLA	Outer Loop Link Adaptation
OoB	Out-of-Band
OOT	Out-Of-Tree module
OSI	Open Systems Interconnection
PCIe	Peripheral Component Interconnect Express
PDF	Probability Density Function
PHY	Physical Layer
PoC	Proof of Concept
POSIX	Portable Operating System Interface
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QPSK	Quadrature Phase-Shift Keying
RB	Resource Block
RF	Radio Frequency
RL	Reinforcement learning

RLLA	Reinforcement Learning Link Adaptation
RRC	Radiocommunications Reference Center
SC	Single Carrier
SCD	Successive Cancellation Decoding
SDR	Software-Defined Radio
SIMD	Single-Instruction Multiple-Data
SINR	Signal-to-Interference-plus-Noise Ratio
SNR	Signal-to-Noise Ratio
TVWS	TV White Space
USRP	Universal Software Radio Peripheral
XLA	Accelerated Linear Algebra

Resumo

As futuras redes móveis fornecerão uma ampla variedade de novas aplicações e casos de uso. Neste contexto, será impulsionada uma demanda significativa de taxa de dados e confiabilidade para que sejam suportados todos os novos serviços emergentes. Muitos destes serviços apresentam requisitos distintos, o que pode levar a decisões divergentes a serem tomadas de acordo com um mesmo indicador. Por exemplo, definindo os parâmetros da camada física de modulação e codificação através do índice Modulation and Coding Scheme, esta opção do sistema deve resultar em uma quantidade de erros inferior a uma condição limite quando o requisito é confiabilidade.

Por outro lado, se o serviço demandar uma alta taxa de dados, o índice MCS deve estar o mais próximo possível do ponto de maior eficiência espectral que ainda atinge uma meta máxima de taxa de erros menos severa. Mas em quaisquer dos casos, o indicador precisa de uma estimativa muito precisa do estado real do enlace. O indicador elementar é, então, a medida que representa o grau de interferência e de ruído em relação ao sinal enviado.

Dado que o cálculo deste indicador elementar ainda necessita de melhorias nos atuais métodos que têm sido utilizados, uma estratégia que se apresenta é o da implementação de um indicador complementar que gere uma informação, não somente mais precisa, mas também que considere todos os elementos presentes nos processos de recepção do sinal. Para uma aproximação maior do que se pode chamar de estado da arte, torna-se claro que devem também ser levadas em conta as particularidades de implementação em termos de algoritmos de processamento de sinais, características de hardware, quantização e tantos outros fatores presentes no processo de recepção.

Ainda assim, o controle na tomada de decisões é complexo, pois envolve o gerenciamento de vários usuários sendo adicionados e deixando o sistema a todo momento. Além disso, há cada vez mais uma heterogeneidade maior de serviços e diferentes demandas a serem acomodadas. De posse do maior número possível de informações, é possível melhor implementar o que é chamado de adaptação de enlace.

Para a melhor eficiência da adaptação do enlace, este trabalho tem como pilares estes três tópicos, propondo estratégias para cada um deles: uma proposta para a melhoria da estimação do indicador elementar (relação sinal-ruído), a criação de um indicador complementar e propostas mais sistêmicas para o controle propriamente dito. Além disso, a pesquisa por métodos visando o aumento de precisão e de eficiência apontou como principal estratégia a utilização de técnicas de aprendizado de máquina, que têm demonstrado notáveis resultados nas mais diversas áreas.

***Palavras-Chave*:** Adaptação de Enlace, Codificação e Modulação adaptativa, 6G, Aprendizado de Máquina, Confiabilidade.

Abstract

Future mobile networks will provide a wide variety of new applications and use cases. In this context, a significant demand for high data rate and reliability will be driven to support all new emerging services. Many of these services have different requirements, which can lead to divergent decisions being taken according to the same indicator. Using the modulation and coding parameters defined by the Modulation and Coding Scheme (MCS) index as an example, the system option must result in a number of errors lower than a limit condition when the requirement is reliability.

On the other hand, if the service demands a high data rate, the MCS index must be as close as possible to the point of highest spectral efficiency that still reaches a less severe maximum target of error rate. But in either case, the link status indicator needs a very accurate estimation. The elementary indicator is the measure that represents the interference level and noise compared to the sent signal.

Since the calculation of this elementary indicator still requires improvements in the current methods, a strategy presented in this work implements a complementary indicator that generates not only more precise information but also considers the imperfections of all processes involved in the signal reception. In the pursuit of state-of-the-art, it is clear that the implementation particularities in signal processing algorithms, hardware characteristics, quantization, and many other factors present in the reception process must be included.

Moreover, the decision-making process is also a complex task as it involves managing multiple users being added and leaving the system at any given time. In this scenario, there is an increasing heterogeneity of services and different demands to be accommodated. With as much information as possible, the process called Link Adaptation (LA) can be appropriately implemented.

For the best efficiency of link adaptation, this work is based on these three topics, presenting strategies for each one of them: a proposal for improving the estimation of the elementary indicator (signal-to-noise ratio), the creation of a complementary indicator and more systemic considerations for the control itself. In addition, the research looking for methods to increase precision and efficiency brought the employment of machine learning techniques as a primary strategy, which has shown remarkable results in many diverse areas.

***Keyords*:** Link Adaptation, Adaptive Coding and Modulation, 6G, Machine Learning, Reliability.

Chapter 1

Introduction

THE primary use cases of digital mobile networks were introduced by the Second Generation of Mobile Networks (2G), where services were mainly based on voice, short messages, and some very latent internet services, like e-mail and rudimentary browsers. From the Third Generation of Mobile Networks (3G) on, the data rate increased significantly, providing access to high-speed mobile internet. The use cases advanced to services like video calling, streaming, and social media access. The Fourth Generation of Mobile Networks (4G) boosted the throughput to rates that enabled high-speed download and upload of data, creating conditions for high-quality video for streaming and online games, high-quality voice calls, cloud-based applications, video conferencing, and others.

The Fifth Generation of Mobile Networks (5G) is providing technologies to increase reliability, lower latency, and higher bandwidth. Thus, many more assorted new applications and services are coming, like the implementation of systems for autonomous vehicles, smart cities, factories, and the Internet of Things (IoT). Based on this scenario, the concept of Network Slicing, brought out successfully by the 5G specifications, is an important direction to properly fulfil the newly demanded services and their distinct and rigorous requirements. As a projection of this environment, the services were structured into three use cases: enhanced Mobile Broadband (eMBB), massive Machine Type Communications (mMTC), and (Ultra Reliable Low Latency Communications (URLLC).

Briefly, eMBB allows for high-speed access to multimedia resources with moderate latency, and mMTC provides connectivity to an extensive number of devices with lower concerns about delays. For the case of URLLC, there is a critical communication characteristic where stringent requirements are necessary for both latency and reliability. For this reason, correctly determining error rate boundaries for the different link configurations is a mandatory undertaking.

As expected, another aspect of the operation in these conditions is a very low error rate. This is a challenging requirement since the determination of limits where errors start to happen generally depends exactly on the occurrence of bit errors. In addition, indicators must be fast and precise to report the current quality status and provide the necessary information to make decisions and predictions. Therefore, accurate and fast indicators are essential elements to the best accomplishment of all expected services.

Taking all these aspects into account, the system has to control the parameters to meet the requirements consistently in a process known as Link Adaptation (LA). The most important parameter to be determined by this mechanism is the Modulation and Coding Scheme (MCS) index, and its selection is made by a process called Adaptive Coding and Modulation (ACM).

The MCS index refers to a combination of modulation and forward error correction (FEC) parameters that provide a balanced distribution of data rates and robustness. The appropriate index choice depends on a previous mapping of Signal-to-Interference-plus-Noise Ratio (SINR) and its correlation with the Bit Error Rate (BER) for each MCS curve.

As the receiver performs the measurements, the information about the channel state, or Channel Quality Indicator (CQI), is sent to the Base Station (BS) via the Channel State Information (CSI) control signal. Considering this prognostic and based on the information about the type of service the user needs, the BS takes its decision.

Figure 1.1 illustrates a simplified diagram of a Mobile Network and the Link Adaptation (LA) context.

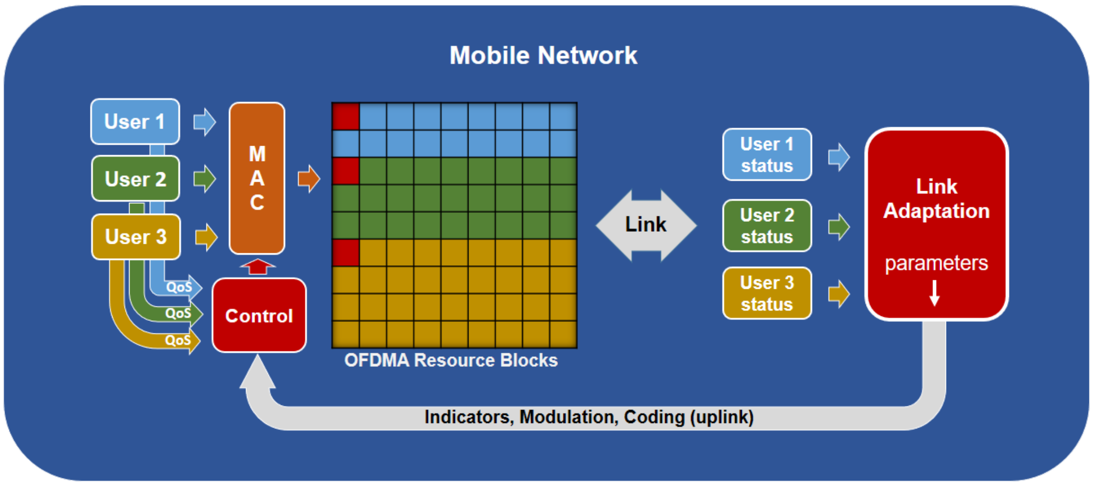


Figure 1.1: *Mobile Network System Overview.*

Divergent decisions can be addressed according to the same indicator. For instance, the selected MCS index must result in an error rate that is lower than an error-free limit condition when reliability is the main requirement. On the other hand, if the service demands throughput, this index has to be as close as possible to the highest spectral efficiency point that still achieves the maximum target BER. But in both cases, a channel state indicator needs a very accurate SINR estimation.

Since the measure of the SINR can present inaccuracies and delays, some workaround has been applied to guarantee the appropriate functioning of LA. A current strategy is an implementation of internal feedback control, or Inner Loop Link Adaptation (ILLA), complemented by an external one named Outer Loop Link Adaptation (OLLA).

Basically, the ILLA operates at a fast time scale with the MCS being increased or decreased based on the SINR to maintain the target BER for the corresponding type of service. OLLA operates at a slower time scale, being estimated over several transmissions after checking the real error rate via Acknowledge (ACK) and Non-Acknowledge (NACK) signaling. In practical terms, this process consists of an insertion of a calculated offset to the threshold in the measured SINR to achieve an average BER fitted to a defined Quality of Service (QoS).

The present work focuses on determining the best MCS choice to apply in the transceiver according to the type of service. The delay caused by information inaccuracy can be mitigated with better processing and inclusion of internal information from techniques applied in the Physical Layer (PHY) context.

For this purpose, a promising strategy that can leverage better results in the link adaptation control is the application of Machine Learning (ML) techniques. Different mechanisms and topologies can be used in this strategy. It can involve the direct treatment of an indicator to enhance its precision, or it can be employed directly in the control system itself.

Another study performed in this work is based on the concept that the internal processes present in the PHY layer can provide significant data about its condition and actions. Thus, there is a procedure to provide a method for generating information related to the decoder's status. Such monitoring leads to a precise indication of how close the decoding system is to an error occurrence.

It has to be mentioned that the investigation concerning statistics originated from inner FEC processing started with a demand exposed by the study of more informative input data that could be used for different ML topologies. However, this proposition demonstrated a very relevant strategy for LA control in an independent manner. For this reason, as a spin-off, an earlier article was elaborated on, submitted, and published.

Added to this proposal, another improvement is provided using a low-complexity Neural Network (NN) to format the indicator. This procedure allows signal conditioning to avoid dependence on the adopted MCS scheme. The results show that the NN could also generate a very consistent BER estimator. The main property of this processed signal is the fast statistic for prognostic of BER since the number of symbols needed to achieve the evaluation can be even a thousand times smaller than the normal statistic processing.

Considering that this work could not execute a complete solution for LA based on all proposed aspects and knowing that this subject comprises a vast field to be explored, the extension of research and implementation from this stage is worthwhile. Some of the methods to improve the accuracy and readiness of indicators are currently being developed. The next step should focus on the direct utilization of reinforcement learning algorithms to control the system.

1.1 Motivation

THE extensively aggregated new services and applications, expected to be present in future networks, will demand complex control systems. Traditional solutions using deterministic methods and equations are reaching a breaking point due to an ever-increasing number of metrics to be considered when making decisions. At the same time, new techniques using ML and Artificial Inteligence (AI) algorithms have been researched and applied to solve the most intricate issues in various scenarios. In all Open Systems Interconnection (OSI) layers or proposed cross-layer structures, this approach is considered the most promising and probably the unique possible solution [1] [2].

When the environment for implementing these techniques is the PHY layer, practical comparisons can proceed to better understand the complexity of ML algorithms to execute traditional Digital Signal Processing (DSP) tasks, such as adaptive filtering using Least Mean Squares (LMS) or linear estimators. Depending on the function, a NN's complexity, in terms of processing resources, tends to be much higher than some equivalent deterministic methods [3].

Thus, the emergent problem of using ML in this case relies on the processing capacity of the involved hardware used in transceivers or other connected equipment in the network. Many complex NNs need a large number of layers and nodes. This condition gets even more critical when applied to the PHY layer since DSP tasks consume a high volume of computing resources.

Bearing this aspect in mind, this work intended to create algorithms and ML topologies that can be processed into actual transceiver hardware. An interesting platform to implement these Proof of Concept (PoC) subjects is Software-Defined Radio (SDR) [4] [5] [6]. It offers excellent flexibility, but it also has the capability to carry out complex SDR tasks and seamlessly integrate ML-based functions [7].

For this reason, the challenge is finding accuracy aggregated to processing efficiency. In such a circumstance, two approaches can be adopted. The first and simplest one is an effort to employ topologies to achieve low complexity NNs. The second one is the prepossessing of information before inputting data into the learning operation. This procedure has shown better results in achieving superior precision and complexity reduction [8].

Considering these aspects, both arrangements were taken into account to propose models that can employ ML algorithms on the PHY. The chosen subject is the LA control system since it is a very important mechanism for attending to requisites of future mobile network's new applications.

1.2 Publications

The first and second papers were produced based on the research related to this work. The third one is related to the main environment used for proceeding with tests and the PoC. The fourth is related to the full implementation of the second paper.

❶ **Roberto Kagami, Felipe A. P. de Figueiredo, Victor Hugo Lázaro Lopes and Luciano Leonel Mendes. ”A Novel Technique for Link Adaptation Based on Inner Receiver Statistics”, Evento: XL Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT2022), doi: doi: 10.14209/sbrt.2022.1570808335.**

❷ **Roberto Kagami and Luciano Leonel Mendes. ”A Low-Complexity Deep Neural Network for Signal-to-Interference-Plus-Noise Ratio Estimation”, in Anais do I Workshop de Redes 6G, Evento Online, 2021, pp. 1-6, doi: https://doi.org/10.5753/w6g.2021.17227.**

❸ **W. Dias, A. Ferreira, R. Kagami, J. S. Ferreira, D. Silva, and L. Mendes, ”5G-RANGE: A transceiver for remote areas based on software-defined radio,” 2020 European Conference on Networks and Communications (EuCNC), 2020, pp. 100-104, doi: 10.1109/EuCNC48522.2020.9200925.**

❹ **João Henrique Silva Delfino, Roberto Michio Marques Kagami, Juliano Silveira Ferreira e Luciano Leonel Mendes. ”Implementação e análise de técnica para estimação de SNR baseado em Deep Learning,” Evento: XLI Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT-2023), doi: accepted.**

1.3 Document Structure

THE structure of this document is organized as follows: A description of the employed system design and architecture is given in Chapter 2, including the architecture, hardware, software, libraries, frameworks, applications, and tools utilized to implement the algorithms, control systems, quality indicators, machine learning topologies, simulations and Proof of Concept execution.

Chapter 3 considers the elementary link status indicators and techniques using ML to improve the accuracy and treatment of the data.

Chapter 4 brings the results of investigative methods to obtain complementary new indicators.

Chapter 5 will aggregate the previous indicators as part of a Machine Learning system for Link Adaptation using different types of topology and algorithms.

Chapter 6 presents the conclusion and indicates a track to future works based on the postulated systems and techniques.

The main Python and implemented machine learning codes are included in the Appendix section to illustrate in more detail the implementations.

Chapter 2

System Design and Architecture

BY virtue of Digital Signal Processing consolidation and the fast advance of computational capacity, many of the functions required by hardware in telecommunication equipment can be handled using algorithms executed by software. Nowadays, even such massive tasks, previously performed by specialized processors, like application-specific integrated circuits (ASICs), can be executed by General Purpose Processors (GPP) in the platforms of SDR. This technology provides many advantages. The strongest point is flexibility, considering the functionality can be reconfigured and updated through software changes without hardware replacement. Another advantage is the upgradability since the radio system is more adaptable to changing needs like additional functions and features.

For the purpose of implementation and PoC, the system of this work employed an SDR platform mainly based on the transceiver of the 5G-RANGE project [9]. It is a project supported by the European Research and Innovation funding program. The National Institute of Telecommunications (INATEL) took part in the Physical Layer development. The main goal of this project was the introduction of new technologies to provide coverage for mobile networks in remote areas.

Among these new technologies is the implementation of Generalized Frequency Division Multiplexing (GFDM) [10]. It is a non-orthogonal multi-carrier scheme based on a traditional filter bank scheme that provides pulse-shaped subcarriers. GFDM is a candidate for future wireless communication systems. This technique is important as long as it generates lower Out-of-Band (OoB) emission than Orthogonal Frequency Division Multiplexing (OFDM). With this characteristic, better spectral efficiency can be achieved and, at the same time, less interference is leaked into adjacent channels. Considering that one of the goals of this project is the secondary occupation of TV White Space (TVWS), this is a meaningful attribute to avoid interference in the primary channels.

Another important topic of this project is the implementation of the Polar Code [11] [12]. It is a linear block error-correcting code with low decoding complexity, high gain, low latency, and good flexibility. These aspects provide an interesting extension of possibilities for robustness and data rates.

An overview of the 5G-RANGE project and its results are described in [13]. Some new elements were included to improve the processing capacity. For the hardware, a graphic card was added to enable machine learning computing. It also demanded the creation of an environment for the utilization of ML functions, libraries, and algorithms.

2.1 Top-level Architecture

Figure 2.1 shows the top-level system diagram. From the Internet Protocol (IP) Network, the data packets are collected and delivered to the Media Access Control (MAC) layer via software sockets. As this layer is a process running outside the Physical Layer environment, a Portable Operating System Interface (POSIX) Interprocess Communication (IPC) is necessary to interface the MAC and PHY layer blocks. The Radio Frequency (RF) Front-end block comprises all the radio frequency analog circuits. Programming and data interface between PHY block and RF Front-end board are performed via a Peripheral Component Interconnect Express (PCIe) bus.

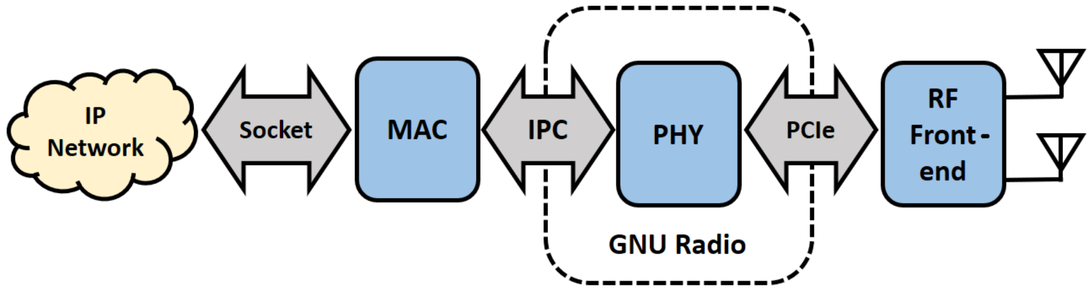


Figure 2.1: *Top-level system diagram.*

2.1.1 MAC Layer

The Media Access Control layer assembles user data packets for transmission and disassembles the received packets at the receiving end. As its name suggests, it controls access to the channel (i.e., the medium) in order to avoid collisions between the users. This procedure is achieved through the mapping of radio resources based on resource blocks distributed in time and frequency domains denominated Resource Blocks (RBs).

The number of necessary RBs per scheduled user portion is calculated accordingly to the modulation, FEC, and rate-matching parameters. The priority of occupation in the RB grid is calculated and defined by the QoS of each user. These are the main tasks to be accomplished by the MAC layer, but other functions can be additionally executed by this layer, like error control, power management, and security mechanisms.

2.1.2 PHY Layer

The Physical Layer proceeds with the conversion of mapped bits into the RBs to symbols accordingly to the modulation scheme in the transmission. It executes the inverse conversion for the reception. The bits are determined by the coding and decoding of the FEC block. From the symbols, the transmitted samples are generated after the waveform module. For the 5G-RANGE project, as previously mentioned, the GFDM technique is used.

For the reception, the recovery of synchronization, carrier frequency, and phase are performed. The physical layer also evaluates the channel characteristics, such as attenuation and distortion, to calculate an estimate of the channel and equalize the received signal. Another important assignment of PHY layer is the provision of measures to indicate the channel link status. That is the main aspect to be focused on in this work. Thus, a more detailed description of this layer will be presented in the next section.

2.1.3 RF Front-end

The RF front-end circuits can be programmed to modulate or demodulate baseband signal in order to up or down-convert it to or from the desired band-pass frequency. Analog filters and parameters for I/Q balance and gain are also provided on this board. Another interesting aspect of this hardware is the inclusion of an Field-Programmable Gate Array (FPGA) that can be configured to design custom logic and applications that require high-speed signal processing or custom DSP algorithms like the Digital Pre-Distortion (DPD), implemented in this project [14].

2.2 Physical Layer Architecture

As the subject of this work is mainly related to the PHY layer, a more detailed description will be focused on this context. Figure 2.2 shows a simplified transmission diagram. Compared to the complete scheme, the main difference is the utilization of only one user in a bridge configuration. This strategy enables more simulation symbols to be used since the data is not distributed to other users. It improves statistical precision and saves time in processing to collect results. At the same time, the link environment is analysed by considering a unique scenario. Besides, there is no need for performing on IPC tasks and more complex MAC functions.

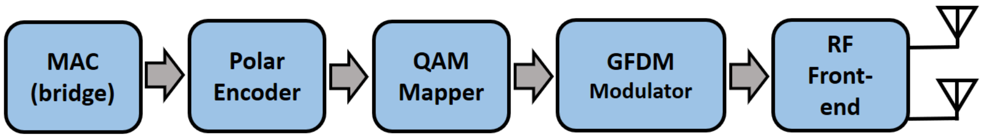


Figure 2.2: *Physical Layer transmission diagram.*

Considering that the diagram presents the basic data flow, the next task is the processing of FEC encoder. The encoder's configuration depends on the MCS index. The MCS options were previously determined according to the best distribution of spectral efficiency. The Quadrature Amplitude Modulation (QAM) mapper establishes the IQ components of the symbols corresponding to the encoded bits. Finally, these symbols are transformed into samples in the GFDM waveform block. The samples are properly synchronized and tagged to be sent to the RF Front-end.

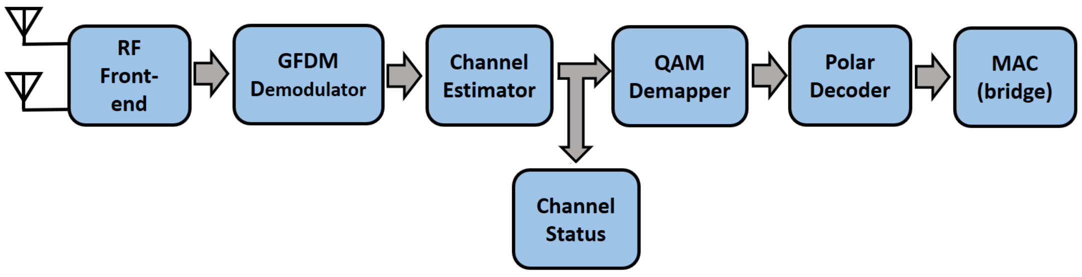


Figure 2.3: *Physical Layer reception diagram.*

The reception signal flow takes the opposite way. Figure 2.3 shows this path briefly. Starting with the RF Front-end, the first function block is the GFDM demodulator. Before all else, it is necessary to find the beginning of the frame. It can be found via the correlation between the received signal and the known sequence. The Cyclic Prefix can be used, for instance, because it is formed by the repetition of the last portion of the frame. However, the 5G-RANGE transceiver implementation inserts a time-domain preamble at the beginning of the frame. This strategy can provide better efficiency for detection. A correlation with this sequence generates the indication of a frame start. The reception detects the preamble and synchronizes the samples to introduce them into the GFDM demodulator. The order and the identification of the frame start are important due to the Fast Fourier Transform (FFT) operations present in this block.

After the demodulator, the symbols are recovered by the channel estimator. This estimation is calculated based on the pilot sequences. These sequences are composed of known symbols distributed along the frequency slots and time spacing. According to this estimation, the data symbols located between the pilot carriers are calculated using interpolation techniques.

The QAM demapper calculates the Log-Likelihood Ratio (LLR) components for the required functioning of the polar decoding. It is a soft decision, but this block can also execute a hard decision. These possibilities will be important to extract information on the decoding status that will be better described in Chapter 4.

The polar decoder receives the LLR information and proceeds with computing the received bits. As the QAM demapper, this block also receives the MCS index configuration from the control channel. With this information, the correct parameters are applied in the polar decoder processing. The decoded bits are then passed to the MAC layer that addresses the data to the respective network users.

2.2.1 Hardware Description

The main platform for implementing the transceiver physical layer functions is based on SDR, using General Purpose Processors. Such an approach allows for high flexibility, where different radio components can be used by modifying the source code. This characteristic also provides the opportunity to change radio settings during communication dynamically. SDR is a very low-cost solution when compared with dedicated hardware solutions [4]. Moreover, it also provides a fast development time. Table 2.1 describes the main hardware components used for the transceiver implementation.

Table 2.1: *Transceiver's hardware characteristics.*

Hardware Element	**Characteristics**
CPU	Intel® Xeon® Gold 5220, 3.9GHz 18 Cores/36 Threads, 64 GB of RAM
GPU	Nvidia® Geforce™ RTX 3070 8GB 5888 cuda cores
SDR	National Instruments USRP 2954R GPS-disciplined oscillator
Vector Extension	SIMD AVX-2

The Central Processing Unit (CPU) choice was based on the acquired experience with different processor configurations and generations. The Intel Core i9-9900 with 8 cores is at the edge of sufficient performance to accomplish the basic functionalities. However, a better configuration is needed to provide enough processing capacity when new features and functions using ML algorithms are added.

Single-Instruction Multiple-Data (SIMD) is a technique used in computer architectures to perform one operation on multiple portions of data simultaneously using parallelism. This allows the CPU to process large amounts of data, providing significant speed improvements compared to traditional scalar instructions. Intel® processors have an integrated set of instructions called Advanced Vector Extensions (AVX) [15]. It is a set of mathematical instructions to attend to this parallelism with different data formats, reducing the need for complex loops and branching structures in the code [16].

For the performance improvement of the ML algorithms, the employment of Graphics Processing Units (GPUs) is the best support in terms of recent platforms for AI applications. The GPU model used is the Nvidia® Geforce™ RTX 3070 [17]. Even though the proposed NN algorithms can be executed in the CPU, the readiness of this solution is interesting to create good and fast PoC implementations. It is also a strong accelerator for learning processes.

The Universal Software Radio Peripheral (USRP) is a versatile and programmable hardware platform used as a front-end on many radio communication applications. It provides a fast prototype and development of custom wireless communication systems. Its flexibility eliminates the need for multiple specialized hardware-based radios, reducing the cost of equipment and accelerating project implementations. The chosen model is the USRP 2954R from National Instruments. It fits the needs for the range of frequency, sample rate, bandwidth, Analog-to-Digital Converter (ADC) and Digital-to-Analog Converter (DAC) resolution. Besides, two channels are available to implement Multiple Input Multiple Output (MIMO) technology. Moreover, it has an FPGA Xilinx Kintex-7 used to implement a DPD scheme. The specifications are described in Table 2.2 for the transmitter and Table 2.3 for the receiver [18].

2.2.2 Software Description

The chosen operating system for implementation in this environment is Linux. The distribution is based on Debian Ubuntu version 20.04 (*Focal Fossa*). The minimum kernel version must be 5.13 for reasons of compatibility required for the Nvidia® Geforce™ RTX 3070's graphic card that will be used.

Table 2.2: *SDR USRP 2954 - Transmitter specifications.*

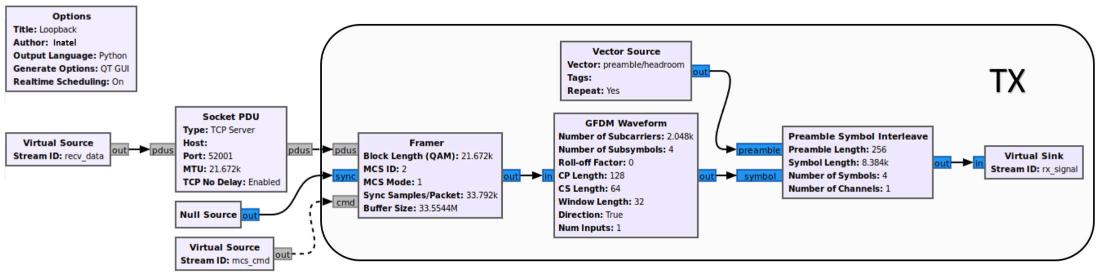
Characteristics	**Value**
Number of channels	2
Frequency range	10 MHz to 6 GHz
Frequency step	$<$ 1 kHz
Maximum output power	20 dBm
Gain range	0 dB to 31.5 dB
Gain step	0.5 dB
Maximum instantaneous bandwidth	160 MHz
Maximum I/Q sample rate	200 MS/s
DAC resolution	16 bit

Table 2.3: *SDR USRP 2954 - Receiver specifications.*

Characteristics	**Value**
Number of channels	2
Frequency range	10 MHz to 6 GHz
Frequency step	$<$ 1 kHz
Gain range	0 dB to 37.5 dB
Gain step	0.5 dB
Maximum instantaneous bandwidth	160 MHz
Maximum input power	-15 dBm
Noise figure	5 dB to 7 dB
Maximum I/Q sample rate	200 MS/s
ADC resolution	14 bit

The primary platform for implementing physical layer functions is based on SDR using the GNU Radio environment [19]. It is an open-source software development toolkit that provides several blocks implemented with ready-to-use digital signal processing algorithms and other utility communication tools. The blocks are integrated into a signal flow graph. This is also a practical platform to integrate and connect proprietary Out-Of-Tree module (OOT) modules.

GNU radio modules are written in both Python and C/C++ languages. The differences between one language and another are, basically, the complexity of programming and processing performance. The Python version used is v3.8.10, and the build version for C/C++, Gnu Compiler Collection (GCC), is v.9.4.0. As an example, Figure 2.4 illustrates a partial schematic of the transceiver's transmission.



Options
Title: Loopback
Author: Inatel
Output Language: Python
Generate Options: QT GUI
Realtime Scheduling: On

Virtual Source
Stream ID: recv_data

Socket PDU
Type: TCP Server
Host:
Port: 52001
MTU: 21.672k
TCP No Delay: Enabled

Null Source

Virtual Source
Stream ID: mcs_cmd

Framer
Block Length (QAM): 21.672k
MCS ID: 2
MCS Mode: 1
Sync Samples/Packet: 33.792k
Buffer Size: 33.5544M

Vector Source
Vector: preamble/headroom
Tags:
Repeat: Yes

GFDM Waveform
Number of Subcarriers: 2.048k
Number of Subsymbols: 4
Roll-off Factor: 0
CP Length: 128
CS Length: 64
Window Length: 32
Direction: True
Num Inputs: 1

Preamble Symbol Interleave
Preamble Length: 256
Symbol Length: 8.384k
Number of Symbols: 4
Number of Channels: 1

Virtual Sink
Stream ID: rx_signal

Figure 2.4: *GNU Radio system diagram of transmission blocks.*

The Python language is an option that facilitates programming, but it is also better suited and handy for integration with GNU Radio. However, it presents some drawbacks regarding processing power and particular issues like thread prioritization restrictions.

Some proprietary blocks were originally written in Python, motivating a necessary acceleration in real-time operation. Two procedures can be performed for this purpose: the inclusion of libraries such as Numba [20] or code rewriting into C/C++.

For the C/C++ language, other acceleration libraries must also be employed. Regarding CPU usage, the improvement using the parallel processing option is performed via the SIMD methods. This method relies on executing a single instruction to process vector operations. For this, two options exist.

The first one is the Volk library [21], which automatically finds the most efficient way to perform the required operation using parallelism and memory alignment according to the processing platform. Initially, a command of this library (volk_profile) must be executed so that the best performance solution found for a particular arithmetic operation is identified, pointed out, and registered.

The second option is the utilization of these parallelization solutions in a direct way, proceeding with intrinsic function calls (SIMD intrinsics) explicitly. The syntax of these functions varies accordingly to the employed processor. It also depends on the type and dimension of the variables used in the mathematical operation. As the transceiver's hardware definition establishes the use of an Intel processor, the instructions are defined by the specifications in the Intel Advanced Vector Extensions Intrinsics (AVX) documentation [22].

Some FFT algorithms used in the transceiver's implementation are based on the GNU radio block set. Another employed library is FFTW3 [23]. This open-source software for computing provides good flexibility and adaptation to the hardware for performance optimization, executing even discrete Fourier transform (DFT) operations with prime size data, maintaining a good performance.

The Polar Encoder [24] is implemented using the open-source library AFF3CT [25], which is also quite flexible. It allows a reduced-complexity version of a Successive Cancellation Decoding (SCD) decoder, proposed in [26]. This method reduces the number of mathematical operations when prior knowledge of the frozen bit positions exists.

The SIMD technique is also used for decoding, with operations that can be done in parallel, decoding several codewords simultaneously. The library also allows the use of the unrolled decoders [27] generation technique, which takes advantage of the architecture of modern processors, making better use of instruction-level parallelism and pipeline.

Some of the tools and libraries can be compiled directly from source code, allowing greater control over the software version and environment. For example, table 2.4 provides repositories for some of the leading software used in transceivers.

Table 2.4: *Open-source Libraries.*

Library	**Version**	**Repository**
GNU radio	v3.9.2	https://github.com/gnuradio/gnuradio.git
Volk	v2.5.0	https://github.com/gnuradio/volk.git
AFF3CT	v2.3.5	https://github.com/aff3ct/aff3ct.git
USRP	v4.1.0	https://github.com/EttusResearch/uhd.git

.

For increasing the processing power in AI algorithm implementations, employing a graphics card, as a General Purpose Graphics Processing Unit (GPGPU), is crucial for a good performance of the tools applied to ML algorithms, whether for learning or inference. The primary tool employed for leveraging ML models is TensorFlow [28]. TensorFlow provides high-level Application Programming Interfaces (APIs) that make it easy for users to define and train ML models. In addition, it has a large community of developers contributing to the development of libraries and support for the users [29].

Table 2.5: *Software installations for GPU usage.*

Software	**Version**	**Repository**
Tensorflow	v2.7.0	https://www.tensorflow.org/install/source#gpu
Nvidia Driver	v510.47	https://www.nvidia.com/Download/driverResults.aspx/186156
CUDA	v11.6	https://developer.nvidia.com/cuda-downloads
cuDNN	v8.4.0	https://developer.nvidia.com/cudnn

Applications that facilitate their use can be bundled, such as Keras [30]. This framework is built on top of TensorFlow in this implementation. Keras simplifies the process of setting up and training neural networks, allowing users to easily create complex neural network architectures with a few lines of code.

Other tools can also support visualization and acceleration, such as TensorBoard and the Accelerated Linear Algebra (XLA) compiler [31], which optimizes TensorFlow calculations. This compiler can simplify code by automatically combining operations, eliminating expendable computations. However, it must be observed that this tool can face some compatibility issues with the Keras library. For the proper execution acceleration, the function to be compiled must access only elements that interface directly with the TensorFlow platform.

The most powerful platforms, systems, hardware, frameworks, and libraries were presented in this chapter. However, various other software, applications, and tools, such as data analysis and graphics, were also installed for the transceiver's best functioning and support.

Chapter 3

Elementary Link Status Indicators

AS the future wireless communications systems spread and provide a wide variety of applications and use cases, the better achievement of new requirements demands a high accuracy of indicators providing information about the link status. The most important parameter in Long Term Evolution (LTE) and 5G standards, on this aspect, is the CQI. Furthermore, when generating this parameter, accurately measuring the Signal-to-Noise Ratio (SNR) is crucial in determining the optimal MCS value. For this reason, this elementary link status indicator will be analyzed first in a way that the main issues related to the lack of preciseness can be mitigated. Beyond this, a proposition of techniques without using high-complexity algorithms is detailed in this chapter.

The concept of this study considers additional imperfections to be taken into account, such as interference and even the internal inaccuracies of processing, algorithms, and hardware issues. For this reason, the term SINR will be used to include and characterize better the impairments to be considered.

In this chapter, some methods to estimate SINR are examined, and, at the same time, new conceptions and techniques like Machine Learning are implemented to precisely calculate this measurement.

3.1 Related works

In OFDM systems, most of the methods used to measure the SNR are data-aided based on known sequences like pilot carriers [32] [33], CP [34] and preambles [35]. This approach is worthwhile, considering that no hard decisions are necessary to establish the reference of the signal.

On the other hand, some aspects point to drawbacks. One is related to the fact that the number of these pre-known sequences in an OFDM frame or subframe is much lower than the number of payload symbols. The reliability of statistical data increases when the number of elements used to generate the information is greater. Secondly, if impairments like fading and impulsive noise hit these sequences in their exact position, the correct estimation of an entire frame is undermined. On the other hand, if some imperfection affects only the payload symbols portion, a correct evaluation will not be achieved. Thus, these propositions do not take into account degradation and imperfections occurring in the data subcarrier's location, and that is exactly the point where the SINR estimation should be performed.

Some researchers have proposed measuring SNR based on preamble transmission before channel estimation. Boumard's algorithm [36] uses two identical consecutive preambles assuming that the channel varies slowly between them in the frequency and time domains. However, significant changes in the channel lead to errors in the estimation. This situation can become worse when higher frequencies are used.

Guangliang Ren mitigates this factor in Boumard's equations by removing the estimated noise power from the total received signal power [37] [38]. Milan Zivkovic proposes a more complex structure of preambles to achieve better results. More subcarriers are divided into identical parts in a way they appear periodically between null subcarriers. After the received signal passes through the FFT process, the noise power can be easily estimated from these null portions [39].

Non-data-aided SNR estimation has been receiving more attention [40], mainly with a combination of fast-growing novel ML techniques. Interesting approaches, like the one presented in Xiaojuan Xie's work [41], would not be immediately considered some years ago due to their complexity. In this proposition, the evaluation of noise is based on constellation image recognition. Offline training is executed using a signal generated by a known modulator and a known Additive White Gaussian Noise (AWGN). Options of fading channels, Rician or Rayleigh, are also added. Various formats of constellation diagrams are exploited. Supervised learning is performed to recognize the constellation. Three different Convolutional Neural Network (CNN) are implemented to proceed with feature extraction processing, including AlexNet, GoogLeNet, and VGG16. These are preeminent CNN for Visual Recognition. All techniques showed accurate results, no matter what is the constellation diagram. The computing is executed via a Graphics Processing Unit since it demands a high volume of processing.

Another ML-based proposition is demonstrated in [42]. It employs a combination of CNN and Long Short Term Memory (LSTM) layers. The characteristic of LSTM layers is their efficiency to make predictions based on historical information. Thus, more than a precise SNR estimation, this model can also provide information about channel behavior tendencies.

One simple method to estimate SINR is the Mean Square Error (MSE) based on hard decisions. However, this calculation introduces non-linearities that result in inaccurate SINR estimation. The adoption of classical MSE can be a viable solution if these imprecision issues are considered and compensated. In order to achieve an acceptable accuracy of the SINR over channel conditions, it is necessary to compensate for the error of the estimator, and the use of Deep Neural Networks (DNNs) is an interesting strategy [43].

3.2 Mean Square Error for SINR estimation

The data-aided methods, based on the use of pre-known transmitted information to calculate the SINR, are advantageous because they do not need channel estimation and hard decision processing. However, only small portions of the frame are taken into account for this purpose. Statistically, the larger the amount of data used for estimation, the better the accuracy and predictability of results. The computation of all sub-carriers provides a superior evaluation of what is occurring in the data region. This approach can also consider inaccuracies in respect of channel estimation, interpolation, quantization, and other internal processes that could not be ignored.

On the other hand, the characteristics present in the recent mobile communication standards can impose some extra difficulties in the utilization of the traditional MSE method. As robust channel coding schemes are used, such as Polar Code, Turbo Code, and low density parity check (LDPC), these techniques allow the system to operate with very low SINR, including negative values in logarithmic scale [44]. Along with that, the normal operation generates a meaningful number of symbols crossing the grid limits. Thus, the inaccuracies are much higher when estimation is calculated using grid-based and hard decisions. Additionally, if a link's condition worsens quickly, the current modulation order can lead to very high BER, which also means that the estimation based on Euclidean distance on the receiver side is even poorer.

3.3 Modified Mean Square Error Function

The first step to mitigate the mean square error calculation's deficiencies is described in this section. A modified MSE function is proposed, where some assumptions and adjustments are used to improve the precision of the traditional MSE calculation. Figure 3.1 shows the instant error vector.

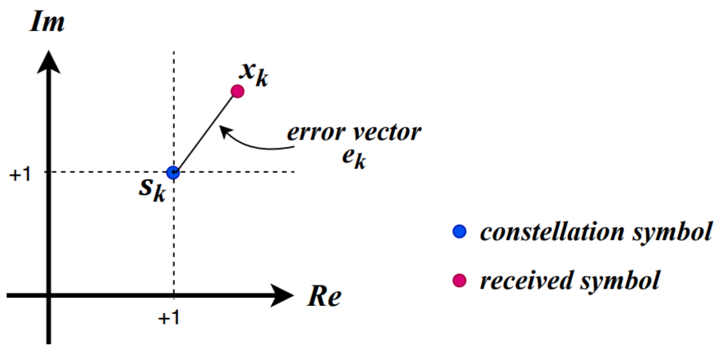


Figure 3.1: *Instant error vector.*

The Modified Mean Square Error Function will be explained by taking into account the following equations. By considering the k-th received symbol $\mathbf{x}_k$, correspondent to the transmitted symbol $\mathbf{s}_k$, and applying the vector amplitude operator ($||$) it can be stated as

$$||\mathbf{e}_k||^2 = ||\mathbf{x}_k - \mathbf{s}_k||^2, \tag{3.1}$$

where $\mathbf{s}_k$ is a symbol pertaining to the sample space of a defined constellation. If the receiver applies a grid-based hard-decision for an M-QAM scheme, where M is the modulation order, d_{min} is the minimum distance between adjacent symbols, the absolute values of coordinates of estimated transmitted symbols $\hat{\mathbf{s}}_k$ can be determined using the floor operator ($\lfloor\ \rfloor$)

$$||\Re(\hat{\mathbf{s}}_k)|| = d_{\text{min}} \left\lfloor \frac{||\Re(\mathbf{x}_k)||}{d_{\text{min}}} \right\rfloor + 1, \tag{3.2}$$

$$||\Im(\hat{\mathbf{s}}_k)|| = d_{\text{min}} \left\lfloor \frac{||\Im(\mathbf{x}_k)||}{d_{\text{min}}} \right\rfloor + 1. \tag{3.3}$$

Hence, depending on the constellation, the maximum value of a $||\hat{\mathbf{s}}||$ coordinate is

$$S_{\text{max}} = \frac{d_{\text{min}}}{2}(\sqrt{M} - 1). \tag{3.4}$$

If an absolute value of a received symbol coordinate crosses the most external decision limit, i.e., $S_{\text{max}} + d_{\text{min}}/2$, we can assume that the most probable absolute coordinate is S_{max}. Furthermore, this occurrence is the key event for the following considerations:

i) If there are no events, then the results of (3.2) or (3.3) do not need correction.

ii) If the value S_{max} is attributed to the respective coordinate of $||\hat{\mathbf{s}}||$ when an event occurs, the calculated error is closer to the correct value.

iii) On condition that the symbols are equiprobable and equidistant, the frequency of this detected error is statistically the same as undetected errors for this coordinate and for each one of the internal symbols ($N/2$, where $N = \sqrt{M}$).

By considering a first-level event as the detected component that does not surpass S_{max} + d_{min} and if d_{min}=2, a similar undetected error δ_k inserts an underestimation calculated as

$$\xi_k = (1 + \delta_k) - (1 - \delta_k) = 2\delta_k. \tag{3.5}$$

A second level event is a detected component that exceeds S_{max} + d_{min} but does not surpass the next hard decision limit. The undetected inserted error, in this case, is given by

$$\xi_m = (3 + \delta_m) - (1 - \delta_m) = 2(1 + \delta_m). \tag{3.6}$$

As the SINR decreases, higher levels of hard decision limits are overstepped, and higher values of errors are inserted. Compensation factors could be used to adjust each level of outdistancing based on its statistical rate and error amplitude. But this estimation would comprise the computation of a complex arrangement.

Taking into account only the first level event, if an α factor is used to represent the rate of undetected events and $\beta = 1 + \alpha$, the modified function for a k-th received symbol

$$||\mathbf{e}_k||^2 = a_k + b_k, \tag{3.7}$$

where

$$a_k = \begin{cases} (||\Re(x_k)|| - ||\Re(\hat{s}_k)||)^2, & \text{if } ||\Re(x_k)|| \leq S_{\text{max}} \\ \beta(||\Re(x_k)|| - S_{\text{max}})^2, & \text{otherwise,} \end{cases} \tag{3.8}$$

$$b_k = \begin{cases} (||\Im(x_k)|| - ||\Im(\hat{s}_k)||)^2, & \text{if } ||\Im(x_k)|| \leq S_{\text{max}} \\ \beta(||\Im(x_k)|| - S_{\text{max}})^2, & \text{otherwise.} \end{cases} \tag{3.9}$$

Eq. (3.7) can be used to compute the average MSE for the lth user as

$$\bar{\varepsilon}_l = \frac{\sum_{k \in \mathcal{L}} ||\mathbf{e}_k||^2}{L}, \tag{3.10}$$

where $\mathcal{L}$ is the set of symbols received by user U_l and L is the dimension of $\mathcal{L}$.

3.4 System Model for Data Analysis

The mobile network system considered in this paper is based on the PHY layer description provided by 3GPP for the 5G in Release 15 [45]. Using the OFDM technique, the data RB are transmitted employing M-QAM with $M \in \{4, 16, 64, 256\}$. The data symbols are mapped into the OFDM subcarriers, and the inverse Fast Fourier Transform (iFFT) is used to generate the time-domain OFDM block. A CP is inserted to protect the data from the time-dispersive channels. On the receiver side, the CP is removed after the time-domain synchronization. The FFT is used to obtain the received symbols in the frequency domain.

Since the estimator only employs the M-QAM symbols to evaluate the MSE, this estimator can also be employed in Single Carrier (SC) schemes. The setup will consider a system with perfect synchronization, carrier recovery, and equalization in order to generate data and evaluate the function behavior with pure AWGN. The system model considered in this paper assumes the M-QAM symbols in the frequency domain after the equalization, then the scheme depicted in Figure 3.2 can be properly used.

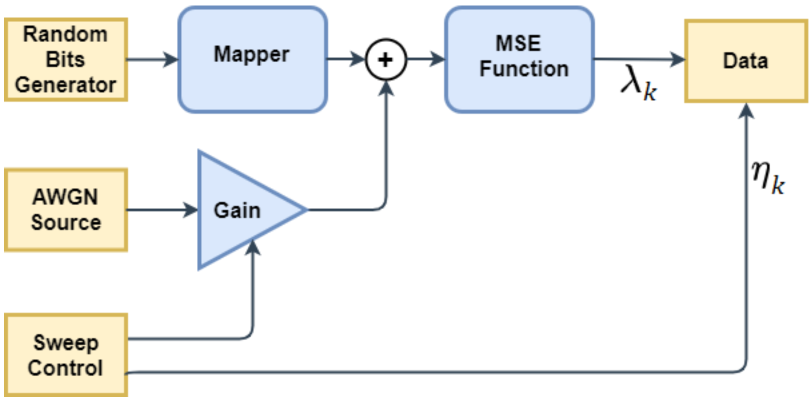


Figure 3.2: *Data set generation diagram.*

The sweep control block selects the gain values of the AWGN source and provides the specified SINR to the data set block (η_k). A random bits generator provides the M-QAM block with input data. This block, in turn, produces the symbols to be added to the noise. Subsequently, the signal is submitted to the function in which output (λ_k) is assigned to the data set block. The adjusted range is from -40dB to +40dB with steps of 0.1 dB. The step time duration depends on the required number of elements to calculate average values. Figures 3.3, 3.4, 3.5 and 3.6 show the MSE obtained by applying (3.10) and making L equal to 1000.

Three scenarios were used to compute the SNR estimation in the MSE function. As presented, the Modified MSE applies a factor considering non-detected events. This parameter establishes $\beta = 1 + N/2$, where $N = \sqrt{M}$, and the approximation is based on the first level event. The results for $\beta = 1$, where only the saturation condition is considered is the second option. The results assuming the unconditional grid-based method for evaluating the MSE are presented as the last scenario. The neural network proposed next will further estimate the difference caused by the omission of other levels in the Modified MSE function.

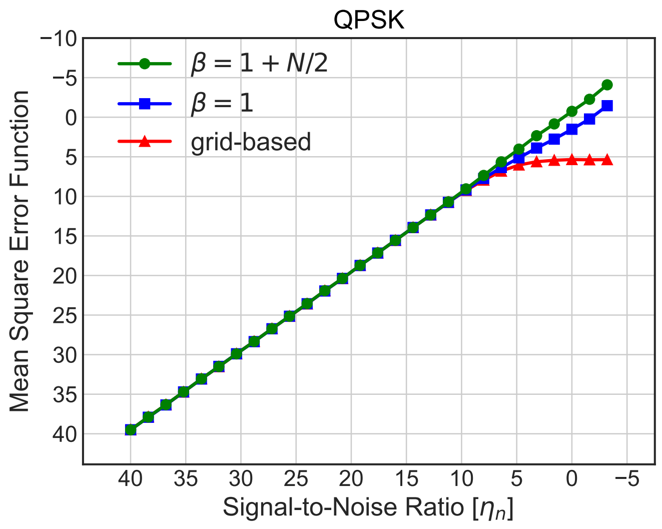


Figure 3.3: *Modified MSE vs. SNR with AWGN channel - QPSK.*

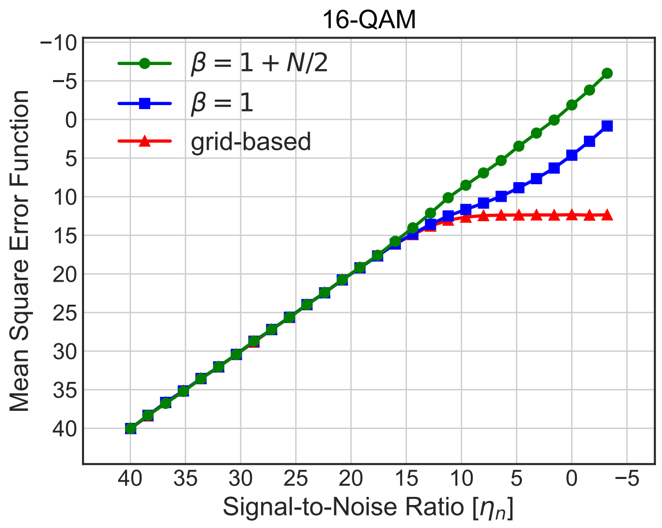


Figure 3.4: *Modified MSE vs. SNR with AWGN channel - 16-QAM.*

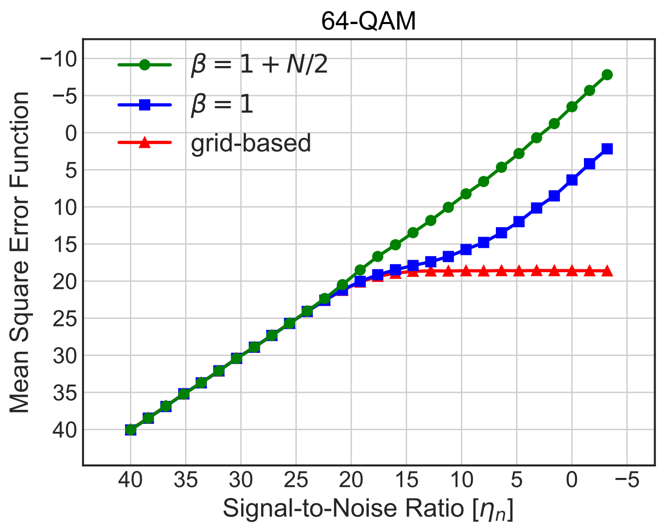


Figure 3.5: *Modified MSE vs. SNR with AWGN channel - 64-QAM.*

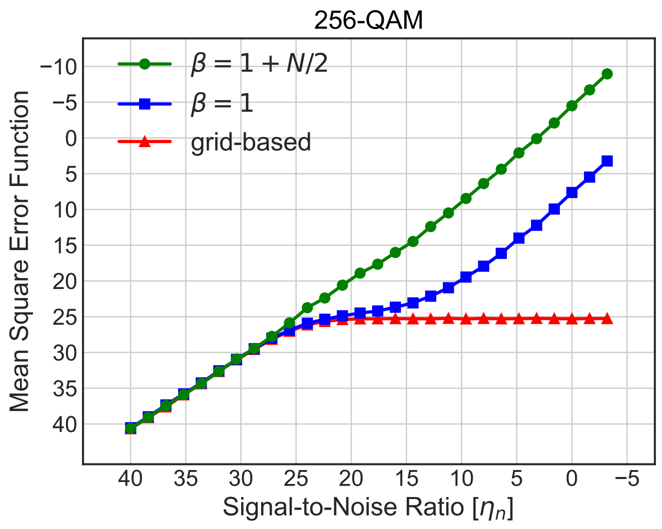


Figure 3.6: *Modified MSE vs. SNR with AWGN channel - 256-QAM.*

Analyzing the results, mainly in the 256-QAM case, it is possible to observe that the modified mean square error function can provide better proximity to an ideal measurement even with low SNR. Anyway, a nonlinear error component is still present in this function, because the used approximation does take into account only the first level of crossing grid events. When the noise is higher enough to generate other levels of crossing, the error increases. Statistical equations could be employed to provide a deterministic estimation considering all possible cases. It also depends on the modulation option, then these are very complex equations to be determined. Therefore, the next sections will consider options to execute this calculation.

3.5 Nonlinear regression function

The samples provided by (3.10) are compared to the target (η_k) in order to produce data for the regression function. From the difference vector ($\lambda_k - \eta_k$), a function can be created via curve fitting techniques [46]. Traditional polynomial regression is an option.

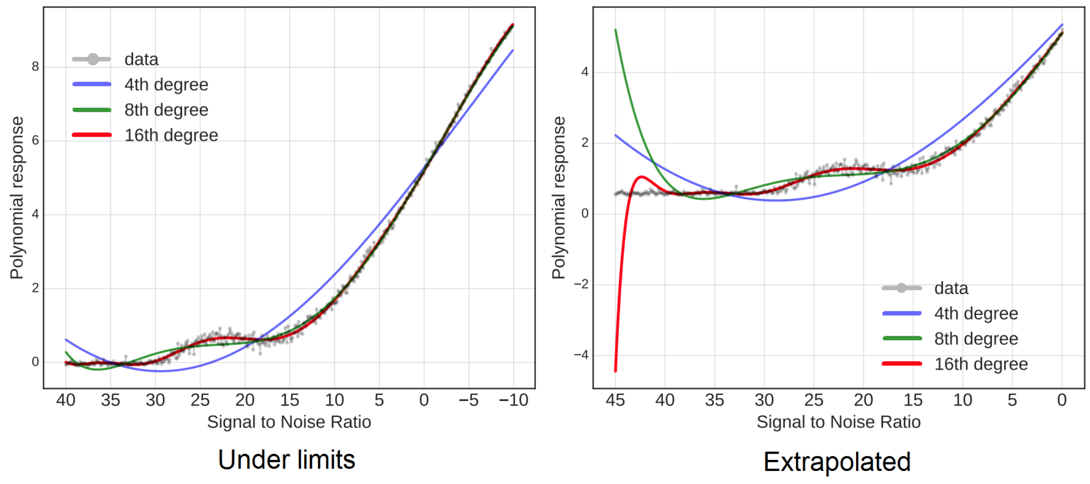


Figure 3.7: *Polynomial nonlinear regression.*

However, in this case, the result is not satisfactory. A high-order polynomial is necessary to fit the curve, as seen in Figure 3.7. Moreover, the curve fitting diverges from the observed data samples for values at the edge and outside of the range.

Neural networks are excellent at finding complex nonlinear relationships between input and output variables. They can model highly intricate nonlinear patterns in the data, making them also particularly suitable for regression issues. A Deep Neural Network with few elements can provide a precise, low complexity, and very satisfactory function. The description of the used DNN is performed in the next section.

As the DNN can precisely fulfil the nonlinear regression, it can be used for the inference of imprecision given by the modified MSE function, as shown in Figure 3.8.

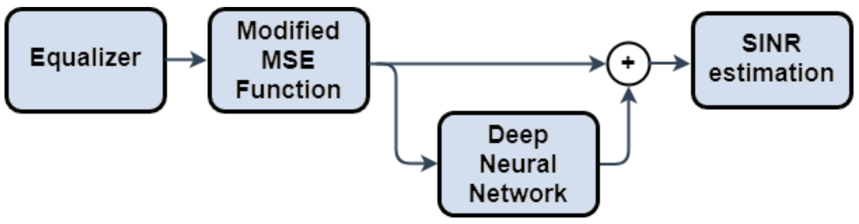


Figure 3.8: *Modified MSE plus DNN scheme.*

3.6 Deep learning network description

Deep Neural Networks are the core of many Machine Learning topologies. It is fundamental for ML, achieving excellent results across different applications. One of its properties is good performance in handling nonlinear regression functions. This characteristic was the trigger to utilize the approaches in the proposed implementations of this research.

The basic element of a Deep Neural Network is perceptron. It is composed of an input, a weighted sum, a bias, and an activation function. Figure 3.9 shows the diagram and equation to produce an output.

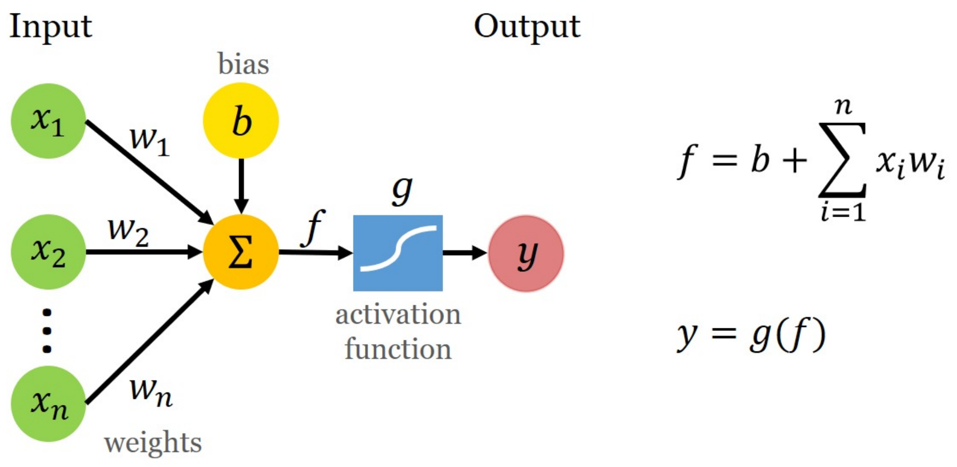


Figure 3.9: *Perceptron architecture.*

In order to create a DNN model multiple layers of neurons are arranged in a way one neuron of a layer combines with all other neurons in the subsequent layer. The concept of a low-complexity DNN is necessary to avoid a processing overload since it will be inserted in a massive computing environment. After some tests, a minimal structure was defined to achieve a satisfactory outcome. The configured DNN topology is shown in Figure 3.10.

It has two hidden layers with four neurons per layer and a singular input. All layers use linear activation functions, except the hidden layer 2, which applies a sigmoid activation function. The bias $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are vectors containing 4 elements.

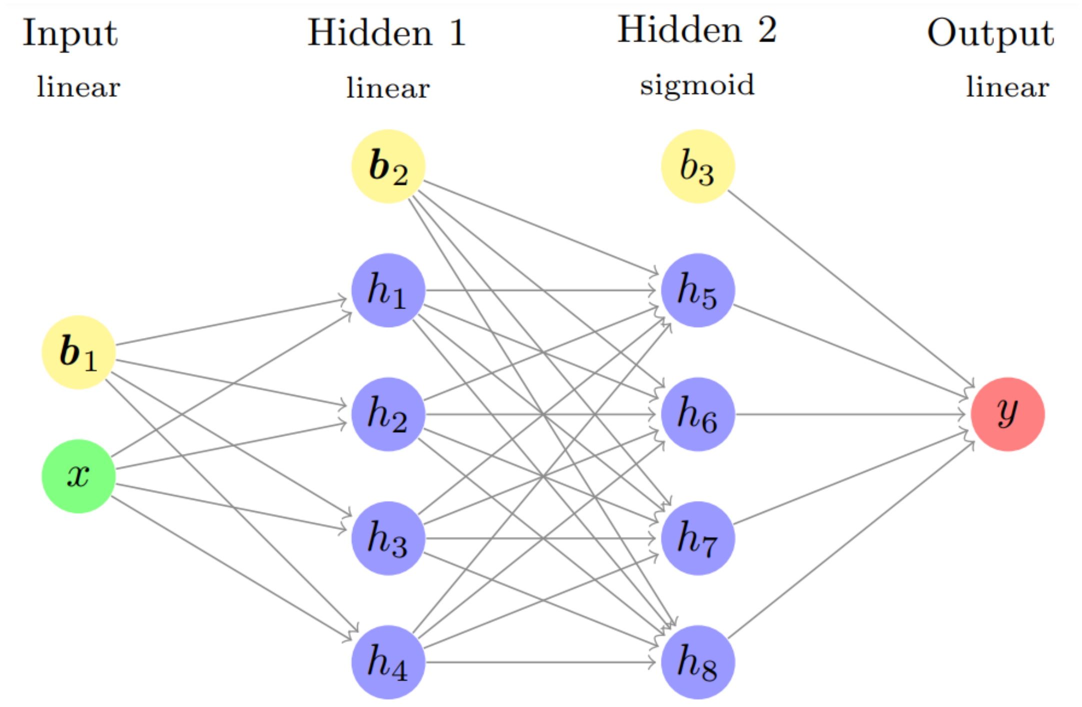


Figure 3.10: *Implemented Deep Learning Neural Network topology.*

In the execution of the learning process, some different methods can be used to adjust the parameters. All algorithms try to minimize the error or loss function. An optimizer's goal is to find the best set of weights and biases to approximate the predicted output and the actual target values. The employed optimizer for this DNN is Adam. It is frequently used due to its fast convergence.

One of the best platforms for Machine Learning is Tensorflow. At the same time, it can run on various hardware, including CPUs and GPUs. It also has a large user community. For these reasons, it was chosen as the framework to create the presented DNN.

All descriptions for options for this topology, according to the modulations, are presented in the Appendix section. The learning process considered an SINR = -10dB as the worst scenario. After this value, the reception process starts to fail in the carrier frequency and timing recovery.

Figure 3.11 shows the biases and weights for 256-QAM model just for exemplification.

$$input_w = [-1.237649]$$

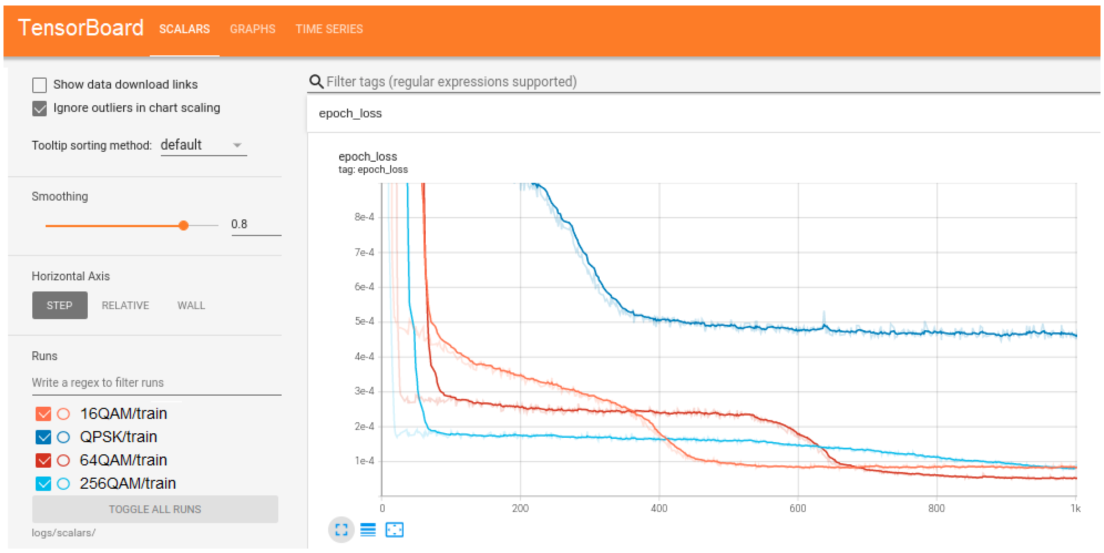
$$\begin{aligned}
x &= [1.801618, -3.228011, -1.409899, 1.187602]\\
h1 &= [-1.786158, 2.077680, -2.200391, -0.523477]\\
h2 &= [0.745564, -4.046422, 0.245129, 5.835315]\\
h3 &= [1.856393, -0.630510, 1.685459, 0.597475]\\
h4 &= [-0.889887, 0.368561, -2.398472, -1.104986]\\
h5 &= [0.489973] \qquad h6 = [0.190718]\\
h7 &= [0.630334] \qquad h8 = [0.206994]
\end{aligned}$$

$$input_b = [0.545633]$$

$$\begin{aligned}
b1 &= [0.724800, 0.731440, -0.607540, 0.650504]\\
b2 &= [-0.186716, -1.346470, -1.704744, 1.183266]\\
b3 &= [-0.184764]
\end{aligned}$$

Figure 3.11: *Biases and weights for for the 256-QAM model.*

Adopting these parameters, the learning process was successfully accomplished, as can be visualized in Figure 3.12 provided by the TensorBoard utility tool. Even in the worst case, which is the 256-QAM option, the error loss ($< 5e^{-4}$) is sufficient to provide a very precise result.



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.8
Horizontal Axis [✓ STEP] RELATIVE WALL
Runs
Write a regex to filter runs
☑ 16QAM/train
☑ QPSK/train
☑ 64QAM/train
☑ 256QAM/train

Figure 3.12: *DNN training (TensorBoard).*

3.7 Results

Figures 3.13,3.14, 3.15, 3.16 show the achievement of a trained neural network. As can be seen, the values of predictions close to the edge are also consistent.

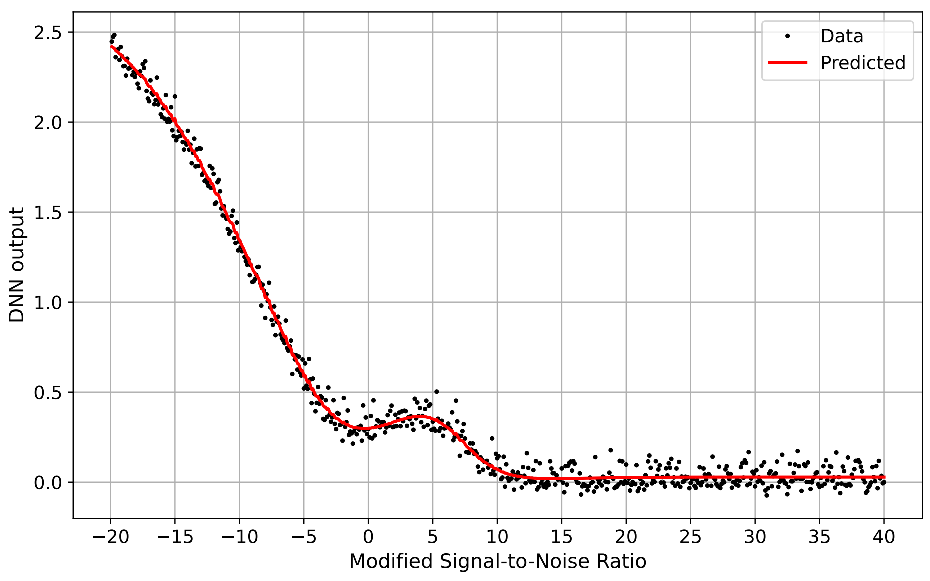


Figure 3.13: *QPSK - DNN output.*

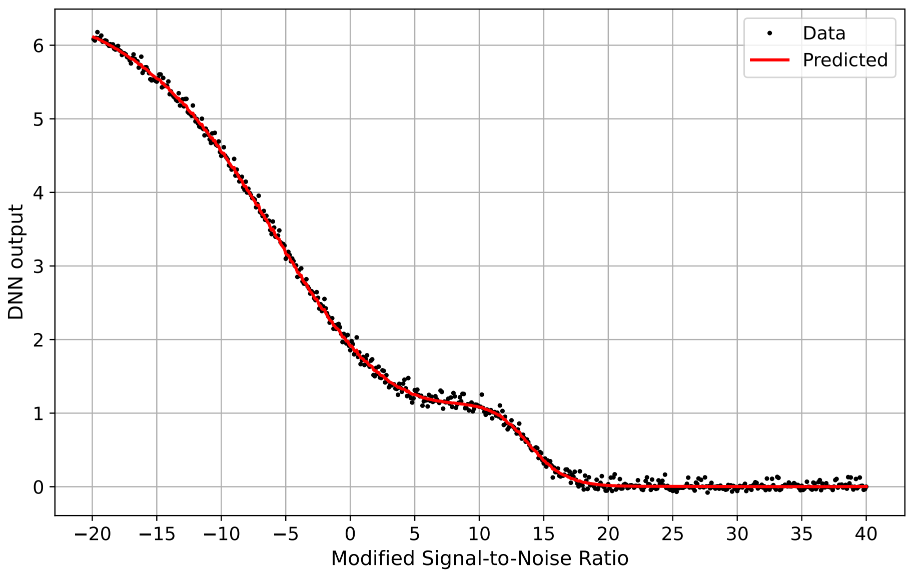


Figure 3.14: *16QAM - DNN output.*

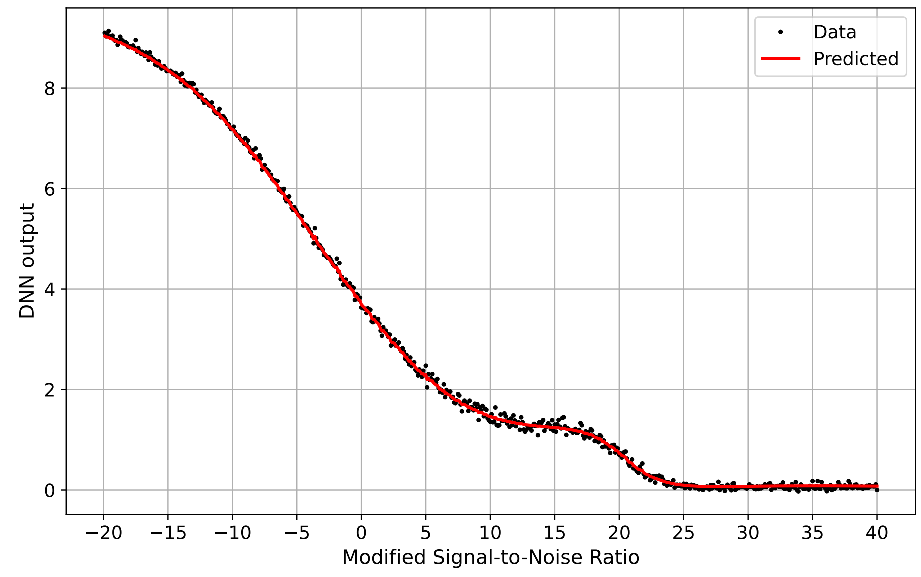


Figure 3.15: *64QAM - DNN output.*

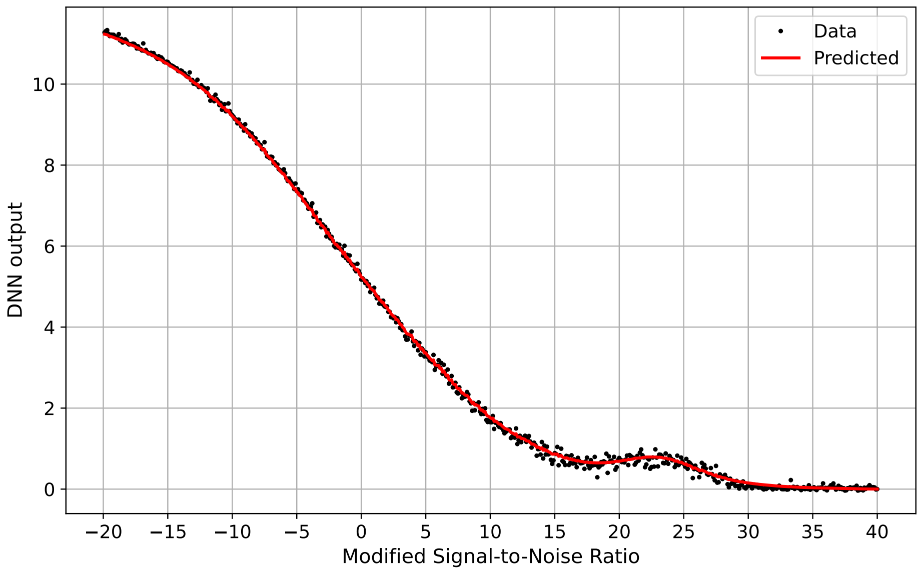


Figure 3.16: *256QAM - DNN output.*

Figure 3.17 shows the achieved results for the proposed scheme. The chosen modulation is 256-QAM since it is the most complex case. As can be seen, the estimated SINR values are very close to the ideal ones.

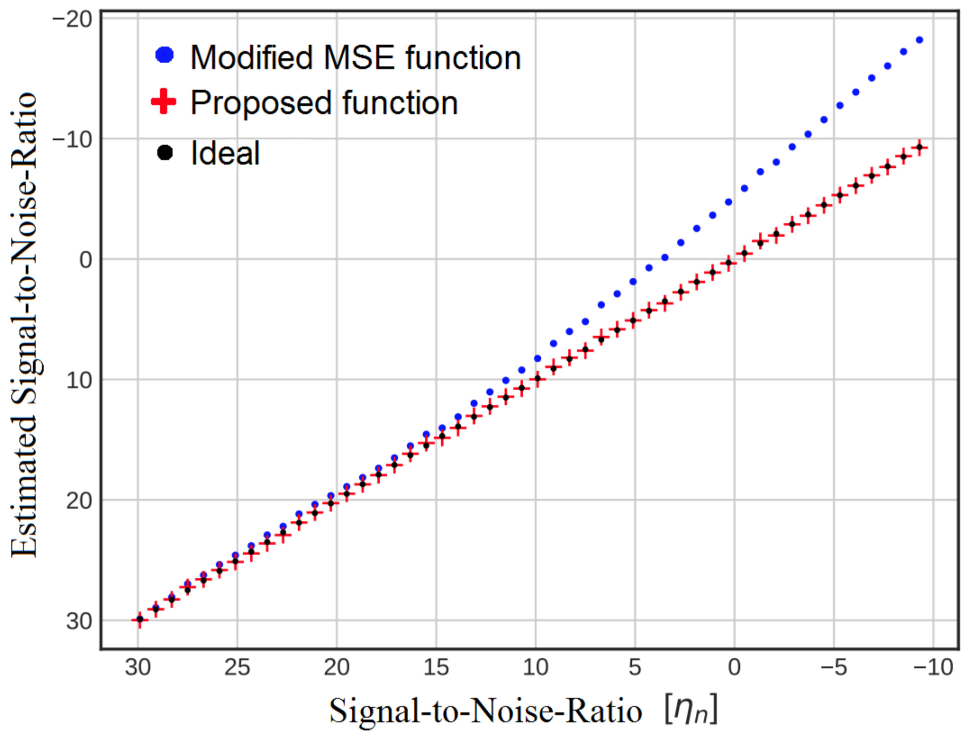


Figure 3.17: *Proposed function response.*

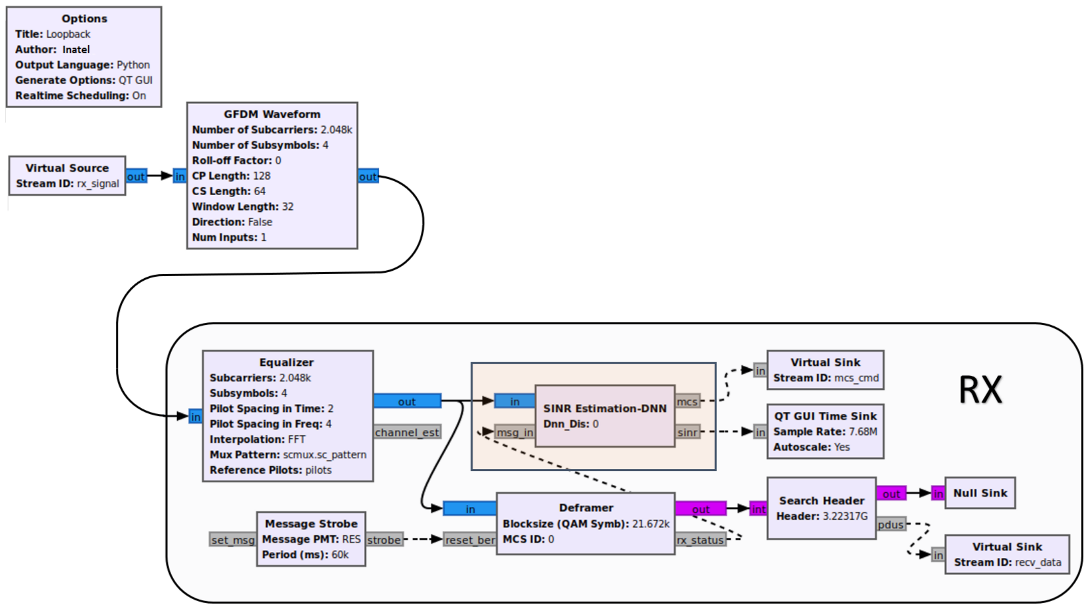
The mean squared error of 400 measurements was calculated by the proposed function. By considering ρ_k as the k-th measure of this vector and η_k as the respective ideal target, the value is given by

$$M = \frac{1}{400}\sum_{k=1}^{400}(\rho_k - \eta_k)^2 = 0.00608495. \tag{3.11}$$

Predictive strategies are interesting when the link condition indicates a fast-changing situation. Then, this DNN can also be connected to other neural network topologies, like Long Short Term Memory (LSTM), for prediction purposes. This combination, for instance, is proposed in [42].

The Proof of Concept implementation of the proposed SINR estimation method, presented in this chapter, took perfect conditions under consideration. However, real operational conditions can lead the SINR estimation to different results.

The optimal solution also depends on the effects of channel estimation and interpolation techniques, synchronism algorithms, and carrier recovery mechanisms. These factors can generate noise propagation, then a DNN retraining within the real environment is an important strategy to achieve better accuracy. Figure 3.18 shows the diagram with the DNN block integrated into the GNU Radio system.



Options
Title: Loopback
Author: Inatel
Output Language: Python
Generate Options: QT GUI
Realtime Scheduling: On

Virtual Source
Stream ID: rx_signal

GFDM Waveform
Number of Subcarriers: 2.048k
Number of Subsymbols: 4
Roll-off Factor: 0
CP Length: 128
CS Length: 64
Window Length: 32
Direction: False
Num Inputs: 1

Equalizer
Subcarriers: 2.048k
Subsymbols: 4
Pilot Spacing in Time: 2
Pilot Spacing in Freq: 4
Interpolation: FFT
Mux Pattern: scmux.sc_pattern
Reference Pilots: pilots

SINR Estimation-DNN
Dnn_Dis: 0

Virtual Sink
Stream ID: mcs_cmd

QT GUI Time Sink
Sample Rate: 7.68M
Autoscale: Yes

Message Strobe
Message PMT: RES
Period (ms): 60k

Deframer
Blocksize (QAM Symb): 21.672k
MCS ID: 0

Search Header
Header: 3.22317G

Null Sink

Virtual Sink
Stream ID: recv_data

Figure 3.18: *GNU Radio system diagram of reception including DNN block.*

As can be seen in Figure 3.19, the behaviour of the Modified MSE implemented in real-time is slightly different from the simulation considering a perfect functioning of channel estimation and reception parameters, mainly with very low SNR values.

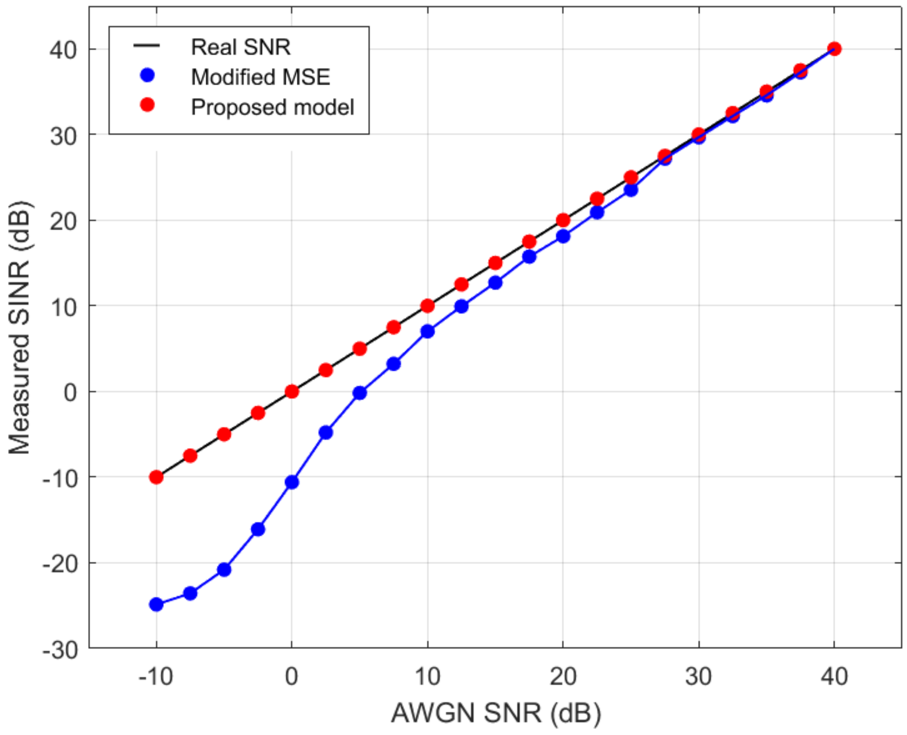


Figure 3.19: *Proposed model response for 256-QAM with real-time implementation and DNN retraining.*

When the channel estimator block generates the symbols, the involved calculation, based on the zero-forcing technique, can increase the added noise effect. This is one of the reasons the estimation tends to present more inaccuracies. However, this difference can be successfully corrected after the DNN's retraining. This procedure was executed by current Inatel's staff at the Radiocommunications Reference Center (RRC).

After the proper implementation of this retrained DNN in the 5G-RANGE transceiver, as shown in Figure 3.18, the LA control was set to non-automatic configuration. The MCS was fixed in the 256-QAM constellation to verify the accuracy of SINR estimation using DNN. Figure 3.20 demonstrates an interesting picture when an SINR of 10dB is applied. At this point, it is no more possible to identify the constellation shape. However, even in this situation, precise estimation is provided.

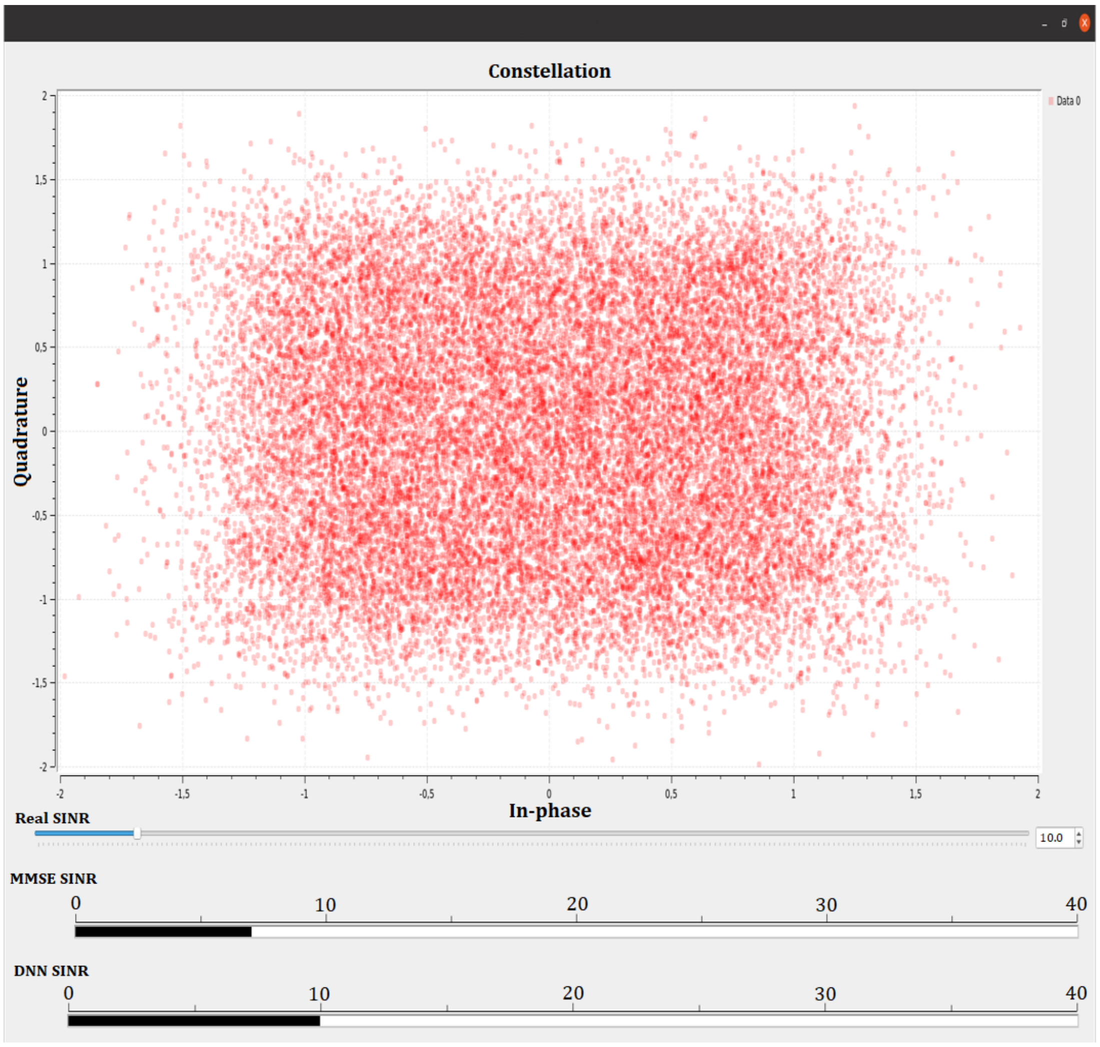


Figure 3.20: *Real-time SINR estimation of a 256-QAM constellation.*

3.8 Conclusions

A precise estimation of Signal-to-Interference-plus-Noise Ratio was efficiently executed by adopting the proposed modified Mean Square Error function plus a low-complexity Deep Learning Neural Network to avoid the intricacy of statistical computation. From this point on, different strategies can be performed to achieve better accuracy, performance, and prediction.

As demonstrated, this method also provides adaptability to different hardware, systems, and architectures by executing proper learning of the DNN. Thus, the designed solution accomplished the intended purpose. The same neural network topology can be easily applied to different modulation options using predefined bias and weights.

The low complexity of DNNs can enable a suitable solution in real-time platforms like Software-Defined Radio. The inference processing is not costly and the implementation was successfully executed jointly with all DSP algorithms.

In spite of the achieved precision in the SINR estimation, more information must be provided to the system to fulfil all the demanded requirements in future mobile network systems. Moreover, such strategies must consider the concerns about real-time processing. Pondering these aspects, the next chapter will introduce some proposals in terms of complementary indicators.

Chapter 4

Complementary Link Status Indicators

FOR the next generations of mobile networks, a large number of parameters are foreseen to manage the diverse new use cases. In this scenario, more complete and accurate information must be generated to provide consistency when defining these parameters. Depending on the required QoS, and a better detailing of the link conditions, the decision-making for each application will be appropriately oriented.

As seen previously, the main parameter to be determined in a Link Adaptation is the MCS index. Then, establishing the currently adopted indicators relies primarily on detecting bit errors. Many link adaptation control methods make use of error occurrences or ACK/NACK statistics to estimate error rate limits [47]. However, this procedure is contradictory when one of the quality requirements is to achieve high levels of reliability.

The primary indicator, also spotted in the previous chapter, is the SINR. It could be evaluated that some estimations generally depend on the assumption that the processes are nearly ideal. It was also evinced that data-aided algorithms do not take into account channel estimation and other reception imperfections affecting the subcarriers with the data. However, these imperfections can be quite significant depending on the channel type and the pilot channel spacing [48].

In practical terms, the limits for Link Adaptation tend to be based on tables with fixed safety margins and hysteresis [49]. Even so, these margins are not precise. Mechanisms to compensate for this error and lack of accuracy are executed by an external control and supervision known as Outer Loop Link Adaptation. This procedure works relatively well, but it can insert a significant delay until complete convergence is achieved. Along with it, slow action triggering is not recommended in control systems [50] [51].

Based on all these reasons, it is clear that complementary indicators can help considerably in these subjects. An interesting indicator would be the capacity to report a probability of error instead of its occurrence. Therefore, in this chapter and the next one, some propositions and methods will be presented to improve ILLA control actions in such a way that Link Adaptation can be performed without the OLLA control.

4.1 Related works

Since the advent of 5G networks, with the introduction of network slicing and multiple scenarios, researchers are contributing with new methods to improve the CQI estimation. Apart from the SINR evaluation and CQI definition, which take part in the called ILLA, the link adaptation control can consider an OLLA mechanism to compensate measurements and processing inaccuracies [52] [53]. A final definition criterion depends on the Bit Error Rate or Block Error Rate target and throughput maximization.

The conventional OLLA evaluation establishes margins that are conditioned by statistics. If the discrepancy between the estimation and the target is high, the process corrects the difference, but it may present a slow convergence. It can also face an even worse scenario depending on the channel variability [54]. Some notable operational results confirm the aspects addressed here. The offset margins introduced by OLLA to compensate ILLA inaccuracies can establish considerable variations (from 10% to 30%) [55].

Other algorithms available in the literature can achieve better performance by using the statistics of the received data. One example is the algorithm based on Bayesian learning proposed in [56]. However, in this case, the simulated target BER values (0.1 to 0.3) are far from what could be considered a high-reliability service category. The OLLA convergence speed can also be accelerated by adjusting the initial offset parameter, as described in [57]. This approach indicates how important is a precise and successful initial CQI estimation in the ILLA.

Proposed methods applying ML have emerged to improve link adaptation efficiency, achieving successful results [58]. In [59], the Multi-Armed Bandit (MAB) algorithm and the reinforcement learning (RL) techniques are applied. Many other machine learning procedures are used to optimize the OLLA convergence and performance [60] [61] [62].

4.2 Error Correction Amplitude Indicator

The typical Inner Loop Link Adaptation feedback is based on a lookup table where an SINR measure determines a CQI index. Empirically, the breaking point SINR for each MCS index can be found by varying the SINR while computing the average bit or block error rate, as can be seen in Figure 4.1. When the SINR approaches its breaking point, the error probability increases, indicating a decrement in the QoS.

After mapping all possible options, safety margins and hysteresis can be applied according to the degree of reliability, required data rate, and QoS factors that supervise the current service. As can be seen, the lack of information about what is happening immediately before the occurrence of errors creates a blind region to estimate the error probability and the reliability degree.

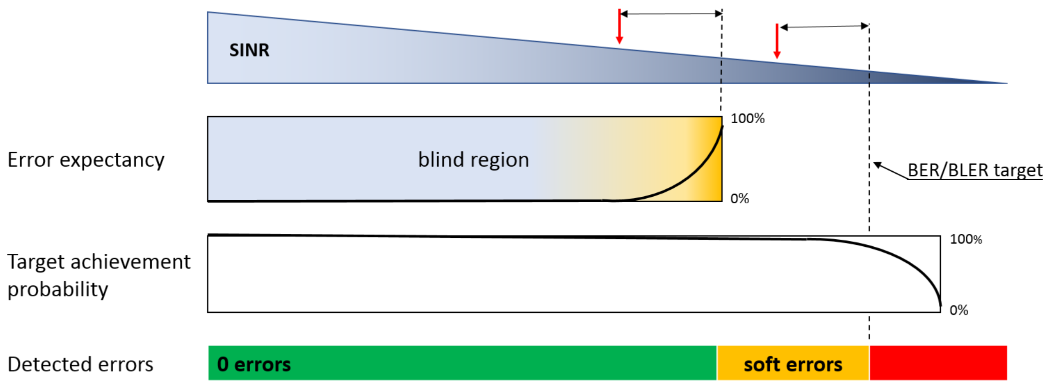


Figure 4.1: *Breaking point estimation based only in the SINR.*

In this chapter, we propose using a complementary indicator, which is based on statistical information available on the receiver side, to establish a better mapping of the link status as depicted in Figure 4.2.

In the bit detection processing, the involved operations carried out by the PHY decoder are capable of generating information about corrected errors. When a FEC scheme indicates the number of bits successfully corrected, this data is directly provided. Some Reed-Solomon encoding implementations, for instance, can generate this information directly.

Otherwise, in schemes like LDPC and Polar Code, data preprocessing is necessary to calculate this value. In these cases, considering hard decoding as a detection processing without correction algorithms, a signal can be created comparing this detection with the complete algorithm detection to estimate a reliability level.

According to this approach, an indicator named Error Correction Amplitude (ECA) is introduced. As a definition, it can be stated as:

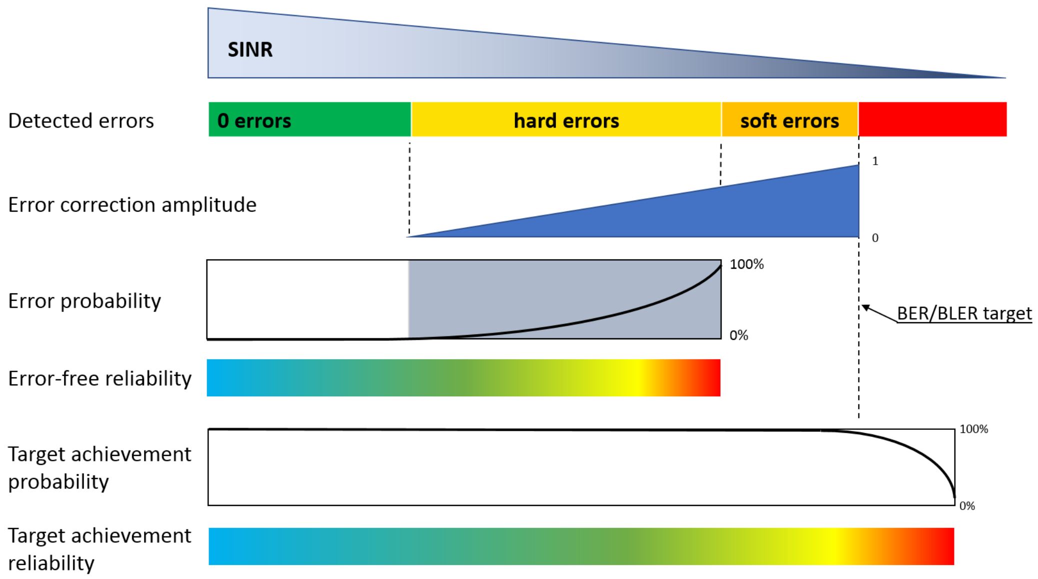


Figure 4.2: *Breaking point estimation based on the inner receiver statistics.*

ECA is the intervention degree applied by the decoding system compared to the detection without correction algorithms or probabilistic analysis techniques.

In practical terms, this indicator can be used to measure the proximity that the decoding system is from an error occurrence or from an expected error rate value.

4.3 System Model for Data Analysis

Actually, polar encoding is used in 5G channel control messages. However, it is a promising technique to be employed in data channels for future mobile networks. Hence, a model to generate the ECA index is proposed based on the Polar Code technique using the block diagram shown in Figure 4.3.

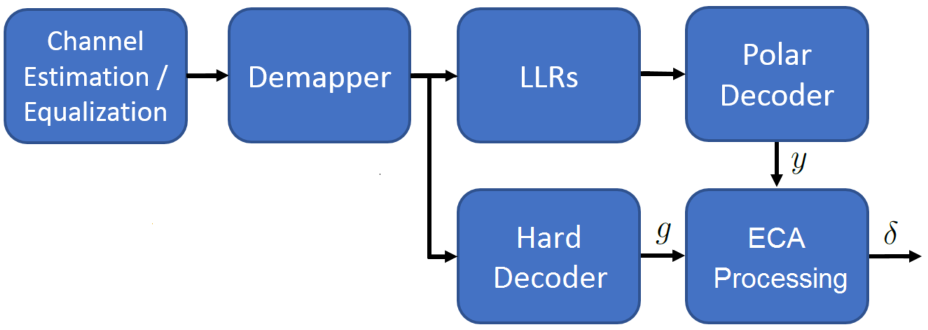


Figure 4.3: *Block diagram for generating the ECA index.*

After the channel estimation and the equalization process, the demapper retrieves the data for the LLR evaluation, which will be used by the Polar decoder to recover the transmitted bit-stream. In parallel, the hard decoder provides non-corrected data. It is a simple task, demanding very low processing resources. Lookup tables can be used, for instance. The ECA processing block compares the sequences obtained in each chain and evaluates the Hadamard distance between them. The measure, in this case, is a particular case of the Hamming distance since it is simply the sum of all different bits between two codewords.

The Polar Coding systems were implemented using the AFF3CT library [63]. It is open-source software coded in C++, licensed by Massachusetts Institute of Technology (MIT), which supports a wide variety of FEC algorithms. Four modulation orders, varying from Quadrature Phase-Shift Keying (QPSK) up to 256-QAM, are combined with 9 different code rates to generate 22 MCSs values. This distribution scheme was elaborated only for validation and proof of concept purposes. Table 4.1 presents all MCSs values considered in this study.

Table 4.1: *MCS definition and breaking point SINR, assuming a codeword length of* $N = 2048$ *bits.*

MCS	**Modulation**	**K**	**Spectral Efficiency**	**SINR** (10^{-6})
1	QPSK	1024	1.0000000	3.8 dB
2	QPSK	1192	1.1640625	4.6 dB
3	QPSK	1368	1.3359375	5.5 dB
4	QPSK	1536	1.5000000	6.2 dB
5	QPSK	1704	1.6640625	7.3 dB
6	QPSK	1792	1.7500000	7.8 dB
7	16-QAM	1024	2.0000000	10.3 dB
8	16-QAM	1192	2.3281250	11.3 dB
9	16-QAM	1368	2.6718750	12.2 dB
10	16-QAM	1536	3.0000000	13.2 dB
11	16-QAM	1704	3.3281250	14.3 dB
12	16-QAM	1792	3.5000000	14.9 dB
13	64-QAM	1280	3,7500000	16.1 dB
14	64-QAM	1368	4,0078125	16.8 dB
15	64-QAM	1480	4.3359375	17.9 dB
16	64-QAM	1600	4.6875000	18.9 dB
17	64-QAM	1704	4.9921875	19.7 dB
18	64-QAM	1792	5.2500000	20.4 dB
19	256-QAM	1368	5.3437500	24.4 dB
20	256-QAM	1536	6.0000000	25.3 dB
21	256-QAM	1704	6.6562500	26.4 dB
22	256-QAM	1792	7.0000000	27.1 dB

As in the PoC of the previous chapter, a transceiver was implemented using the same platform to evaluate the performance of the proposed indicator under real operating conditions. Figure 4.4 depicts the block diagram of the test setup.

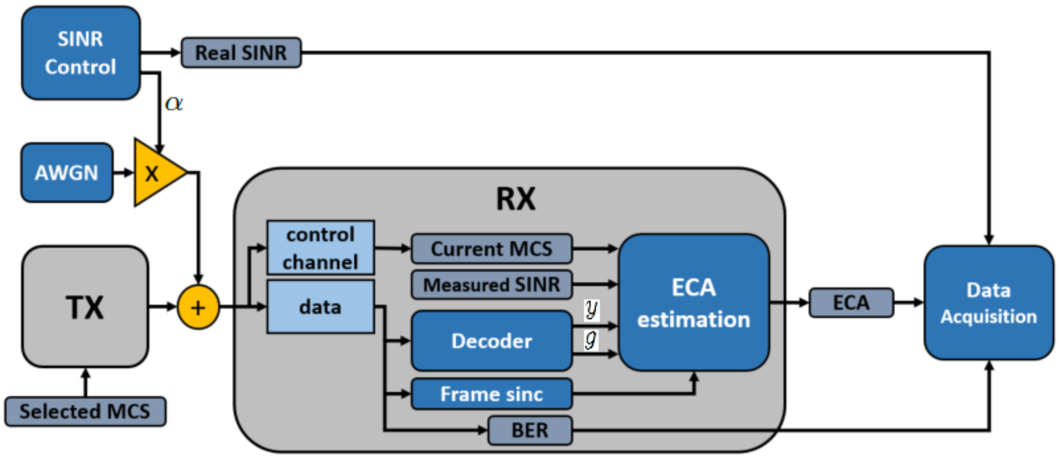


Figure 4.4: *Data collection diagram.*

For each fixed MCS index, the SINR control block automatically sets the AWGN power level by sweeping the α parameter and sending the real SINR value to the data acquisition block. The reception block processes the ECA and the average BER, delivering the results synchronously to the data acquisition block. The reception block also configures its current MCS index via the control channel, and the payload data is used to calculate the BER statistics. This collected data is shown in Figure 4.5.

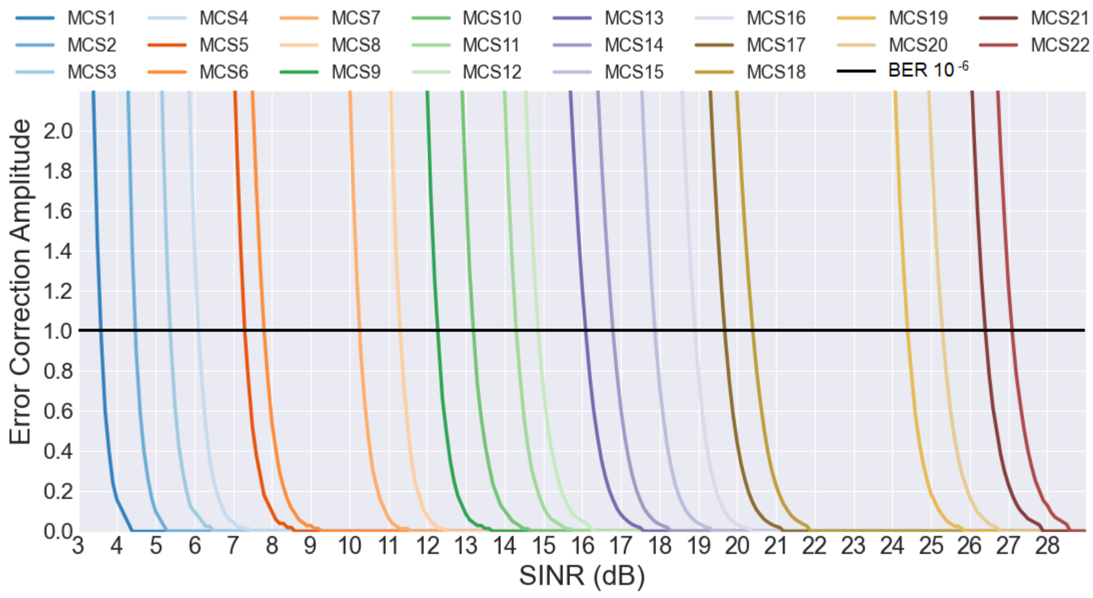


Figure 4.5: *ECA vs. SINR.*

The ECA calculation is executed for each frame and can be determined using (4.1).

$$\delta = \frac{\sum_{i=1}^{m} |y_i - g_i|}{\Lambda}, \tag{4.1}$$

where g_i is the i-*th* bit produced by the hard decoding, y_i is the i-*th* bit produced by the soft decoding and m is the number of payload bits in the frame. The Λ parameter is the maximum number of hard bit errors allowed in the frame to achieve the soft BER target. This parameter is obtained by applying a fixed SINR (column 5 of Table 4.1) for each MCS index. Considering the lower spectral efficiency, $\approx 10^9$ bits were computed to establish a reasonable average value of Λ. This procedure links the ECA index to a target BER of 10^{-6}. Thus, the Λ value is $\approx 10^6$ in this situation. Lower target BER could be used, but it would spend longer-lasting simulations.

In order to evaluate the performance of the proposed SINR approach, errors were artificially introduced in the SINR measurements. The simulations do not include other impairments. To provide a controlled environment, only white Gaussian noise is added. Two types of distortions were inserted.

When SINR estimation is obtained using pilots, preambles, or cyclic prefixes, e.g., the imprecision of channel estimation and interpolation processes are not taken into account. In this case, the overestimated SINR option simulates a situation where its value, measured by the reception block, is 0.5 dB more optimistic than the real one provided by the control block.

The nonlinear distortion comes from the calculation using the average ratio of the symbol power to the error power, where the received error magnitude is the Euclidean distance between the received data symbols and the decided symbols (hard decision based on the constellation grid). Using (4.2), the SINR value of a kth received symbol is based on the squared error, given by [64]:

$$\|\mathbf{e}_k\|^2 = a_k + b_k, \tag{4.2}$$

where

$$a_k = \begin{cases} (|\Re(x_k)| - |\Re(\hat{x}_k)|)^2, & \text{if} |\Re(x_k)| \leq d_{\text{max}} \\ (|\Re(x_k)| - d_{\text{max}})^2, & \text{otherwise}, \end{cases} \tag{4.3}$$

$$b_k = \begin{cases} (|\Im(x_k)| - |\Im(\hat{x}_k)|)^2, & \text{if } |\Im(x_k)| \leq d_{\text{max}} \\ (|\Im(x_k)| - d_{\text{max}})^2, & \text{otherwise}, \end{cases} \tag{4.4}$$

with x_k is the kth received symbol, $\hat{x}_k$ is the closest symbol in the M-QAM constellation, and d_{max} is the maximum absolute value of the current constellation coordinate.

The average SINR per frame is defined as

$$\gamma_{\text{dB}} = 10 \log \left(\frac{\sum_{k=1}^{n} \|\mathbf{e}_k\|^2}{n} \right), \tag{4.5}$$

where n is the number of received symbols in the frame.

When a received symbol crosses the constellation grid, an incorrect error value is generated. Figure 4.6 shows the difference between the real SINR, provided by the control block, and the measured SINR with imperfections, provided by the reception block.

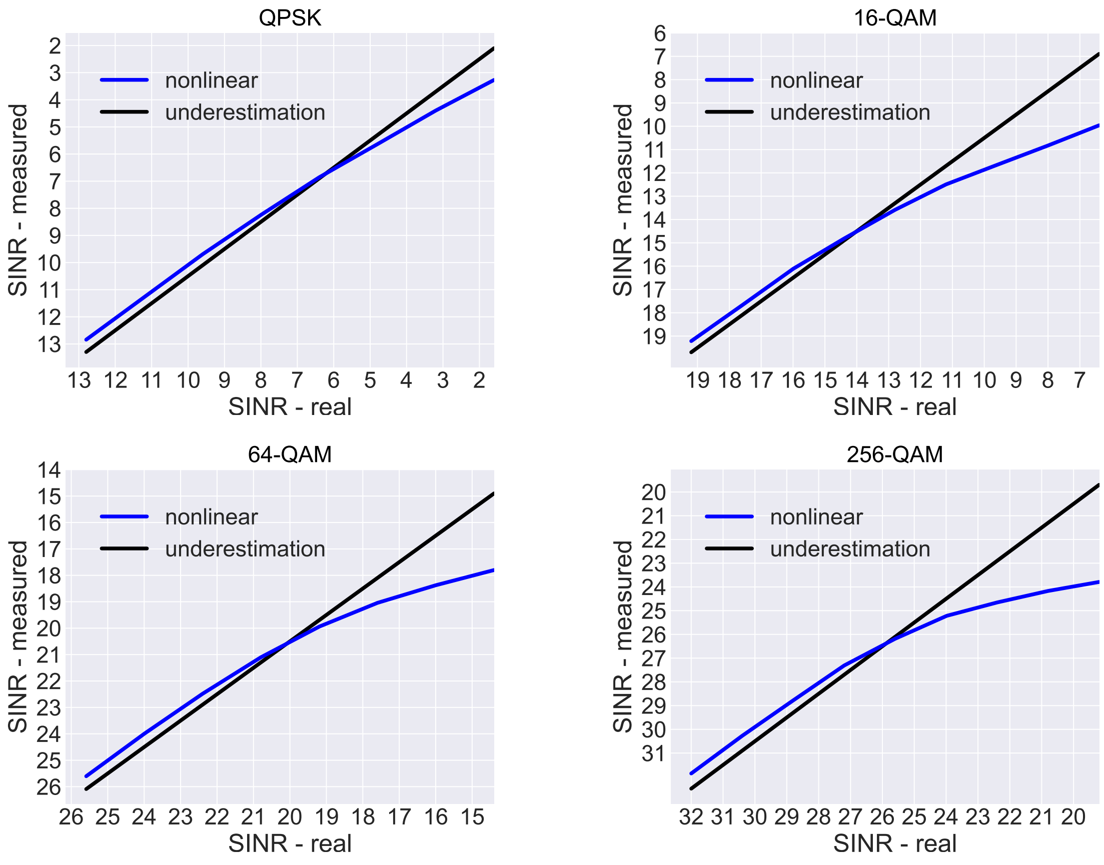


Figure 4.6: *Measured SINR with imperfections.*

For comparison purposes, three different modes were implemented to control the link adaptation process. The system diagram is shown in Figure 4.7. To evaluate only the results provided by the ILLA, there is no OLLA scheme. A plain operation is adopted, where the MCS index can remain unchanged, or it can be increased or decreased by one unit. To reduce the control complexity, high mobility characteristics were not considered. The decision is processed at the end of each frame.

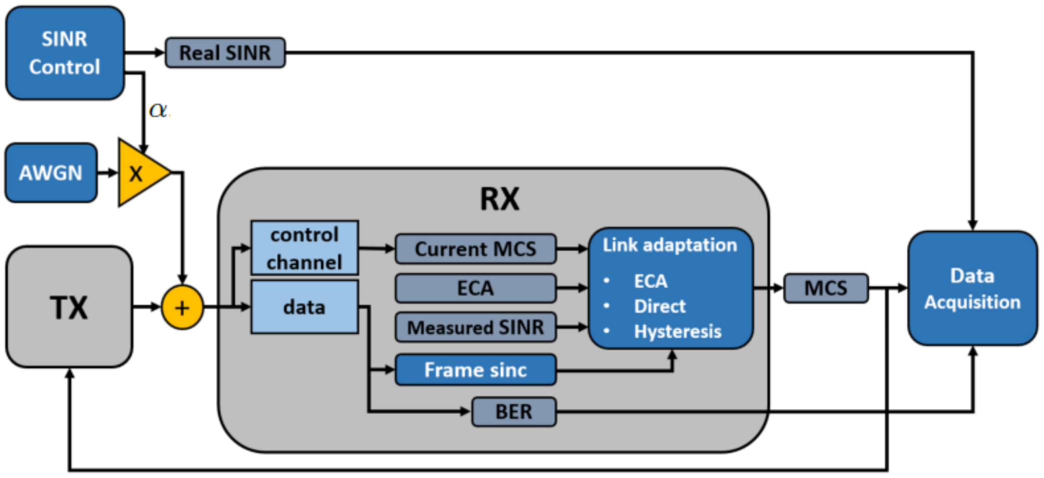


Figure 4.7: *System diagram.*

To proceed with the link adaptation control, the estimated MCS index is sent via the loopback path to the transmission block. The MCS index configuration is synchronously applied during one entire frame length.

The direct control mode chooses the MCS index based on the breaking point SINR according to the lookup table (column 5 in Table 4.1). The hysteresis control mode also employs the same lookup table. However, an offset (+1dB) is added to the breaking point SINR to establish a fall-forward action threshold. An offset (+0.5dB) is also added to the breaking point SINR to establish a fallback action threshold. It creates a safety margin and lower variance.

The ECA control mode defines an immediate fallback action if its value is greater than 1.0. The fall forward action, on the other hand, depends on the lookup table based on data shown in Figure 4.5. The ECA value is estimated by addressing this lookup table with the current MCS index increased by one unit. The fall forward is executed whenever this estimated ECA value is smaller than 1.0.

4.4 Results

The SINR control block automatically provides the α and real SINR values, as can be seen in Figure 4.7. The setup range is from 4.2 dB to 27.5 dB with steps of 0.1 dB. A payload data comprises 21670 symbols per frame. The number of elements used to establish average values is 600 frames per SINR step. The achievement index is the average ratio of successful target achievement occurrence (BER $< 10^{-6}$) to the total events. One evaluation is executed per SINR step. As the channel is AWGN, parameters like the number of pilots, CP, and subcarrier frequency spacing do not affect the results.

Figure 4.8 shows the results for the three control modes using SINR values with overestimation. Figure 4.9 exhibits the obtained results for the three modes using SINR values based on the grid-based decision method, which generates a nonlinear response. From the simulations in these two scenarios, the total transmitted bits for each type of MCS index control were computed, as well as the BER and respective target achievement index. The tables 4.2 and 4.3 present these results.

Table 4.2: *Transmitted bits with overestimated SINR.*

Mode	**Transmitted bits (Mb)**	**Achievement %**
Direct	11.134	0.43
ECA	10.918	99.57
Hysteresis	10.372	100.00

Table 4.3: *Transmitted bits with nonlinear measured SINR.*

Mode	**Transmitted bits (Mb)**	**Achievement %**
Direct	10.982	30.34
ECA	10.826	95.30
Hysteresis	10.172	100.00

As can be seen in Figures 4.8 and 4.9, most of the time, the direct control mode exceeds the target limit. Collecting the number of transmitted bits, for the case of SINR measurement with overestimation error, the ECA control mode provided a 5.26% gain compared to the hysteresis control mode.

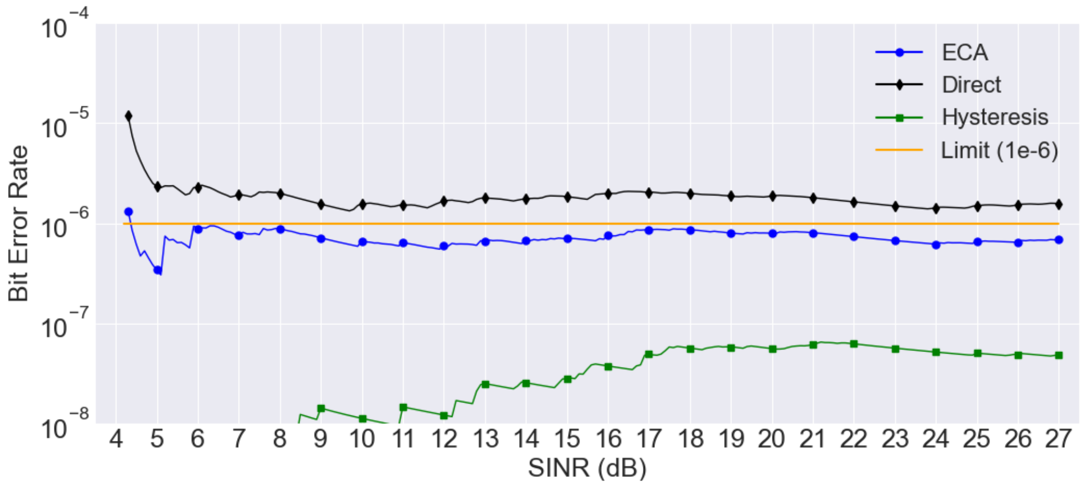


Figure 4.8: *BER using the overestimated SINR (0.5 dB).*

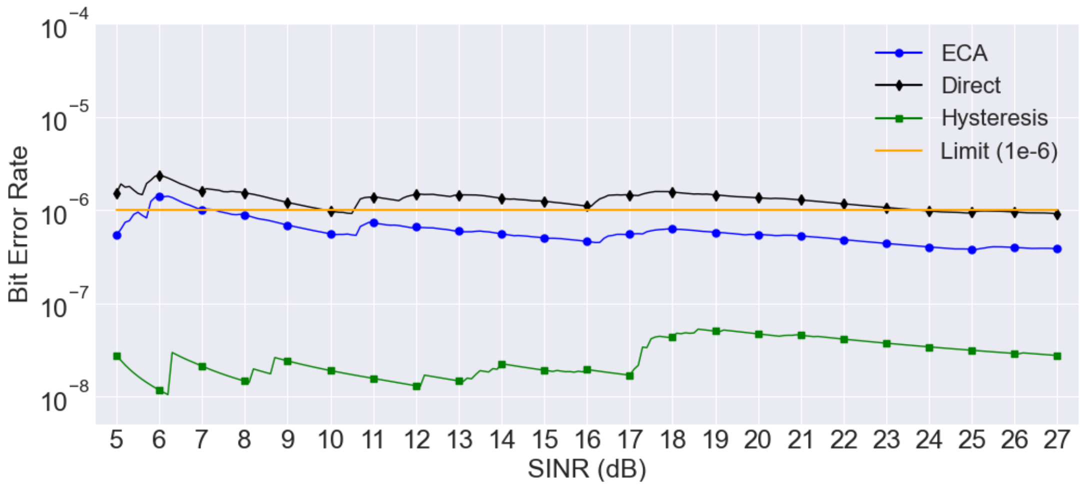


Figure 4.9: *BER using SINR with a nonlinear response.*

Similarly, when a nonlinear error is applied, a 6.43% gain is verified. The direct and hysteresis control mode could be tuned to equalize the gains. However, this procedure would only fit these particular situations. Using the ECA, the target is automatically achieved for different scenarios, different MCS indexes and different receiver implementations.

As can be seen in Figure 4.6, the estimation error is significant in regions after fallback limits. A fast-changing link condition and delayed fallbacks can establish even greater gains. Examining the data close to the BER target limit, it is displayed that the number of hard-bit errors is hundreds of times greater than the number of soft-bit errors. As a result, this attribute provides faster statistics about low error rates.

4.5 Conclusion

The simulations demonstrate that the MCS control using the Error Correction Amplitude indicator for the link adaptation improves the throughput by keeping the demanded reliability strictly. The proposed indicator is consistent with the expected attributes. It estimates the actual error tendency and indicates a faster and more precise sample of low error rates. It can infer error probability before the occurrence, extracting BER statistics with low processing and without payload consumption.

Chapters 3 and 4 proposed different strategies for precision improvements and for providing complementary information. Anyway, Link Adaptation will require more control mechanisms to take into account the PHY provided information, mainly considering the entire system status. The next chapter brings out decision-making issues and propositions keeping in mind a more systemic point of view.

Chapter 5

Link Adaptation Control Using Machine Learning

THE employment of effective indicators is an essential groundwork for Link Adaptation process. For this reason, the subjects of the previous chapters were mainly based on the improvement of the elementary one (non-data-aided SINR estimation) and the proposition of a complementary link status indicator (Error Correction Amplitude).

Many techniques have been proposed to perform Link Adaptation control. Among these propositions, there are several using Machine Learning algorithms. As could be seen in Chapter 3, this technology can also help with accuracy issues. However, the decision-making also needs more information about the demanded constraints and Quality of Service requirements.

This chapter proposes some architectures for the use of LA control systems employing ML techniques. The first focuses on ECA as a sufficient standalone element. The second is the simplest type of control based on the reinforcement learning technique. This is a basis for other topology propositions that are presented as schemes for research and Proof of Concept in future works for LA control using Machine Learning.

5.1 Related works

Many alternatives have been employed to enhance the efficiency of Link Adaptation. In a straightforward manner, this improvement resides basically on a solution providing the higher possible data rate accordingly to a required error rate limit or a reliability level. In the context of system control and probability theory, the principle of maximizing results from a set of actions is known as Multi-Armed Bandit (MAB) problem [65], which is also the perspective of RL techniques. In [66], the criterion of MAB is employed to mitigate the problems of slow convergence and sub-optimal throughput present in the Outer Loop Link Adaptation. In this scheme, the bandit arms, or action options, are responsible for mapping a context vector that comprises the observed link state to the success probability associated with each available MCS. An Artificial Neural Network (ANN) is modeled to predict the transmission success probability.

The MAB is inherently performed by reinforcement learning techniques, like in [67]. This proposal is based on error clustering, which refers to the event where consecutive transmission errors tend to occur in groups due to the presence of bursty noise or fading. Named Reinforcement Learning Link Adaptation (RLLA) in [62], this proposal uses RL with the Thompson Sampling technique. This is a well-known algorithm used in the field of sequential decision-making. It is applied to solve the explore-exploit trade-off problem, where the decision-maker balances between gathering more information (exploration) and exploiting the currently best-known option (exploitation). This is executed by incorporating uncertainty through Bayesian inference, where exploration actions have uncertain reward distribution, and exploiting actions use the knowledge of rewards gained in the previous rounds.

Bayesian inference is a good strategy to combine preexisting knowledge and update beliefs with new evidence, quantifying uncertainty [68]. In this specific case, it models the ACK probability associated with each candidate MCS value. This online learning method is also used in [56], receiving the denomination BayesLA. Both propositions are similar and achieved better results compared to traditional OLLA algorithms. However, in BayesLA, a Rayleigh fading wireless channel is included in the simulations.

One of the most applied and straightforward techniques of reinforcement learning is Q-Learning, introduced by Watkins in [69]. It is based on an environment with discrete states mapped in a matrix. This is basically a table containing the rewards collected when an action produces a state transition. The algorithm's objective is to learn the optimal action-value function to estimate the expected cumulative reward. One drawback of this technique is the discretizing of states. If the granularity is high, meaning a low-dimensional environment (i.e., a few states), the quantization is better, and the technique converges fast. However, a high-dimensional environment generates a long-lasting time of convergence in the learning process.

In Deep Q-Learning (DQL), the state table can be replaced by a Deep Neural Network [70]. This way, in high-dimensional environments, the learning convergence time is shorter, and there are no discrete levels for the states [71]. This method is explored in [72], and it is denominated as Deep Reinforcement Learning for Link Adaptation (DRLLA). Three different types of channels were used to evaluate this proposition: AWGN, Raileigh, and pre-recorded real-world channels [73].

The related works in this chapter and in the previous ones highlight the importance of accurate link adaptation in mobile networks to achieve higher throughput and reliability. Machine Learning algorithms with incorporated state-of-the-art indicators are promising tools for optimizing link adaptation in future wireless networks.

5.2 Link Adaptation Control Topologies

The control architecture for LA can employ the primary and the complementary indicators as they were previously defined in this dissertation. However, distinct topologies can be used to provide actions or statuses as output. The first case will be referred to as Direct Decision since the information to be sent to the Base Station via Channel Quality Indicator is the MCS index itself.

The second case is referred to as an Indirect Decision topology since the ML output can contain statuses or probability information. In this proposed scheme, this data is sent to the BS, where the system can better decide the actions based on the QoS contract specification [74]. Figure 5.1 shows an overall perspective diagram, including some of the introduced elements. An implemented topology can utilize any of these components.

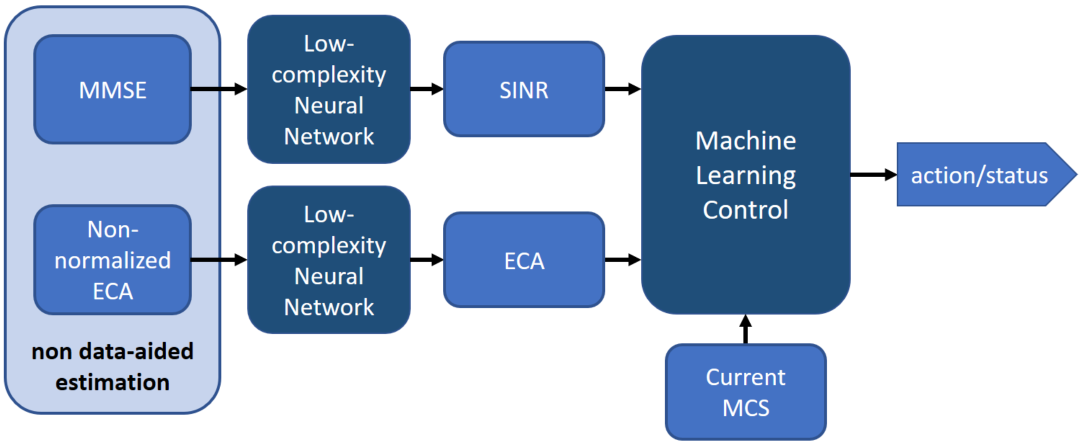

MMSE
Low-complexity Neural Network
SINR
Non-normalized ECA
Low-complexity Neural Network
ECA
non data-aided estimation
Machine Learning Control
action/status
Current MCS

Figure 5.1: *Link adaptation control overall perspective.*

In the Direct Decision topology, the actions that must be informed by the ML for the LA control are the remaining at the current MCS index or an indication of change. This adjustment can be a positive (*fall forward*) or negative (*fall back*) jump. It can also inform a unitary or a multiple-step. In the implemented PoCs, the process increments or decrements the MCS index at each frame.

Depending on the ML algorithm, the output can indicate an immediate jump to the value that best satisfies the required conditions. From this context, some approaches can be initially indicated:

- Direct Decision with Standalone ECA;
- Direct Decision with Q-Learning;
- Indirect Decision with achievement probability output.

The Direct Decision items and related implementations will be considered in this chapter. The Indirect Decision proposition is intended to be evaluated in future works. Thus, it will be better explained in the final chapter.

5.3 Direct Decision with Standalone ECA

The Error Correction Amplitude concept constitutes the basis of this proposal, where the system goal is to determine the limits of reliability with the best precision for the achievement of QoS. As could be seen, if the SINR estimation is not based on the model presented in Chapter 3, the correlation between the SINR measurement and the error rate is not straightforward and linear since it presents unidentified inaccuracies. More than that, it can be based on data-aided estimations. In this case, as previously mentioned, the estimation results do not take processing imperfections into account in the payload portion of an OFDM frame.

In consequence, the application of the ECA indicator provides more precise information about the real state of the link regardless of all these factors. Figure 5.2 is a diagram representation of a topology using only the ECA.

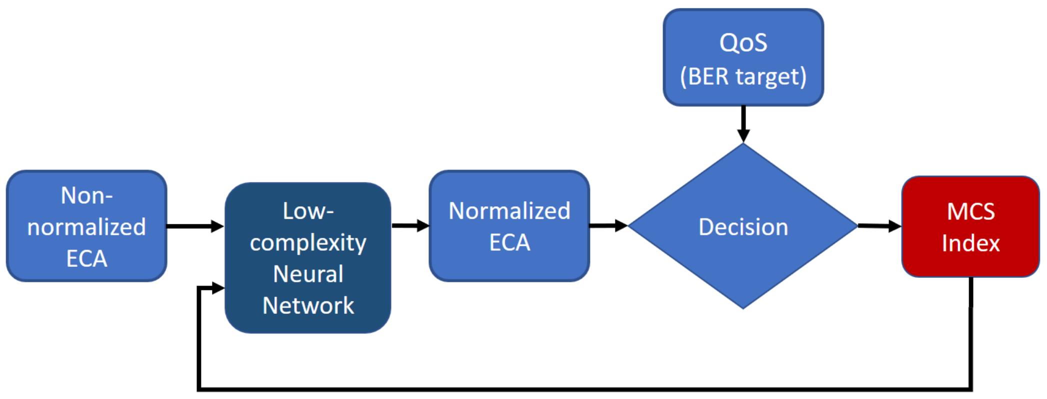


Figure 5.2: *Link Adaptation Control scheme based on standalone ECA.*

The ECA normalization process was implemented using a Deep Neural Network, and this process will be described later on in this section. Some initial considerations must be taken into account with respect to ECA and BER. As expected, the correlation between ECA and hard BER (error occurrence with the hard decision) is always linear and direct. It can be seen in Figure 5.3.

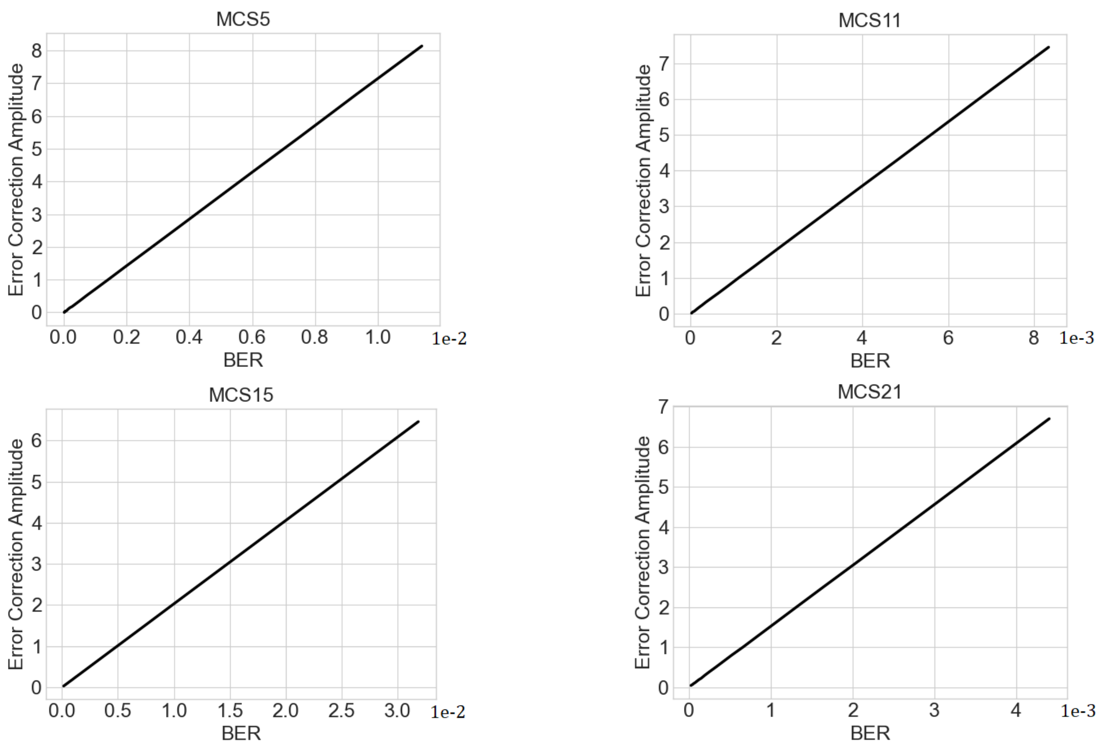


Figure 5.3: *ECA vs. hard BER.*

However, some differences can be found in the correlation between the ECA index and the soft BER (error occurrence with the soft decision) accordingly to the MCS index. This variation can be originated by characteristics such as the constellation shape, code rate, and FEC efficiency. It can be seen in Figure 5.4.

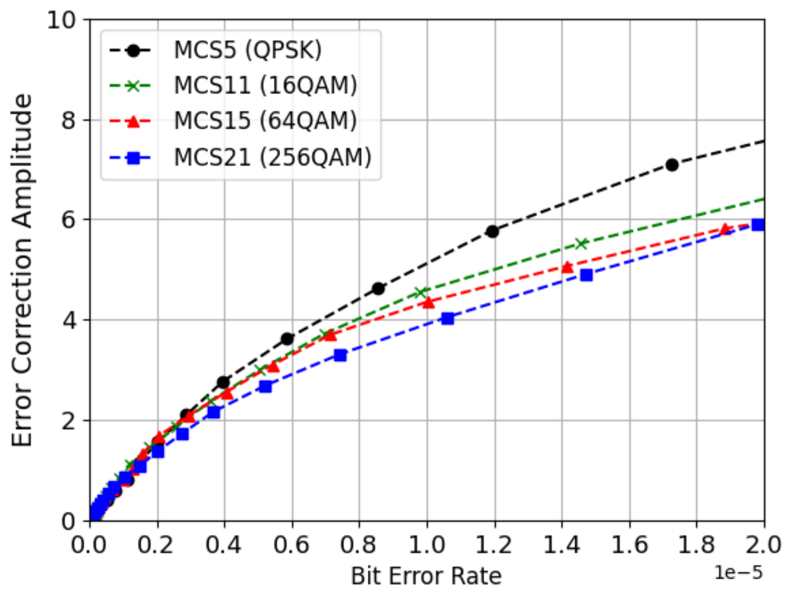


Figure 5.4: *Measured ECA vs. soft BER.*

To normalize the curves of the ECA versus the BER, a low-complexity Neural Network can also be implemented. For this proposition, a similar Deep Neural Network topology used in Chapter 3 was trained. The difference is the addition of an input. Figure 5.5 shows the model where x_1 is the non-normalized ECA and x_2 is the current MCS.

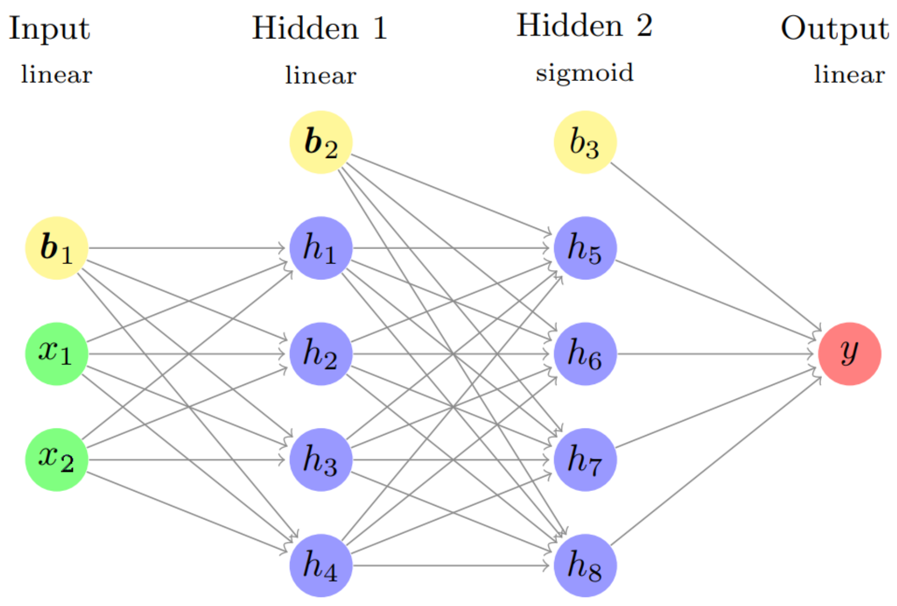


Figure 5.5: *Neural Network topology for ECA normalization.*

The result was successfully achieved. Some ECA values were used as references and indicated a very balanced output for soft BER estimation, as can be seen in Figure 5.6.

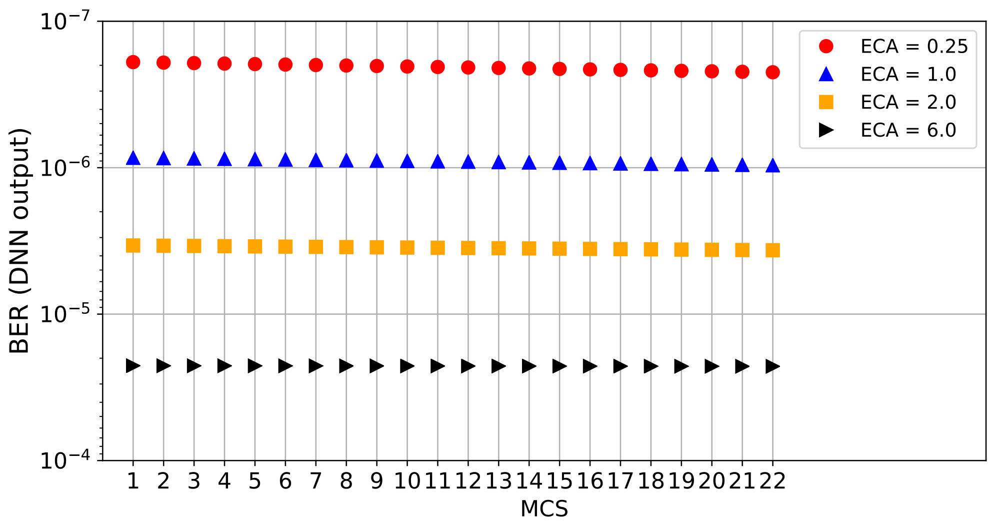


Figure 5.6: *Normalized Error Correction Amplitude output accordingly to MCS index.*

Employing the normalized and calibrated ECA, a simple control scheme can be implemented based on the BER target for the QoS value established by the system.

Furthermore, the ECA vs. soft BER curve differences will depend on the type of techniques and algorithms used by the different implementations of transceivers. The application of the adaptive and learning characteristics of ML algorithms is quite advantageous for the determination of the intended results.

5.4 Direct Decision with Q-Learning

In this model, shown in Figure 5.7, the reinforcement learning block's output indicates the immediate action to be taken, that is, the MCS index to be used.

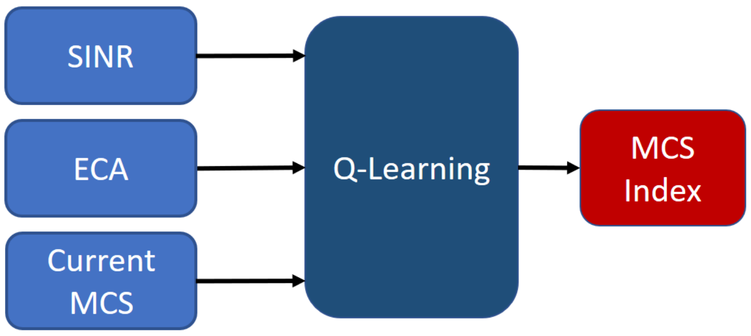


Figure 5.7: *Direct decision with Q-learning diagram.*

Among the most successful algorithms for decision-making based on reinforcement learning is the Q-Learning scheme [75]. It is a method based on creating a flowchart of actions based on rewards, which are determined according to the relevance of choices necessary to reach a final state that satisfies certain requirements. In this way, a mapping of actions is processed according to each state through an array of values (Q-values). This method is interesting because it is a low-complexity system.

The principle of obtaining a convergent result is based on Bellmann's Equation [76]. In conformation with the reinforcement learning approach, the statement indicates that a long-term reward of a current action is the combination of a reward from the current action and the expected reward from future actions. Thus, a fitted equation to compute the Q value, considering actions and states, can be written as

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha[R_{t+1} + \lambda Q(s_{t+1}, a) - Q(s_t, a_t)], \tag{5.1}$$

where α is the learning rate, λ is the discount rate, R_{t+1} is the reward that depends on the transitioning from the time t to t_{+1}.

States: For the execution of the algorithm using this method, the current MCS index, the value of SINR, and the value of ECA were used as input parameters.

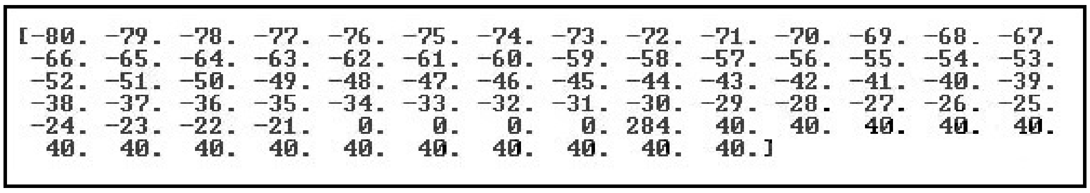
Actions: The actions to be performed are arranged in a vector with 3 commands: ”stay”, ”up”, and ”down”, indicating respectively that the system has to stay in the current MCS index, it must change to one index higher (fall forward), or it must change to one index lower (fall back), indicating the best-suited index of MCS according to the current condition.

To achieve better convergence, debouncing is included since the system delay can insert some instability. The confirmation of a state, or the current MCS index decision, is executed after consecutive iterations. The number of iterations was specified as 20 during the training process and as 2 in the inference execution. Other values can be used without compromising the functioning, but empirical tests pointed to a good convergence utilizing this strategy.

The inclusion of SINR was necessary due to the characteristics of the performance curves when using constellations 2^N with N assuming the values 2, 4, 6, and 8. These are configurations found in standards such as LTE and 5G. However, the exclusion of odd values of N generates performance gaps between the considered modulation schemes.

When polar codes are configured to fill these gaps, the resulting spectral efficiency using consecutive MCS index with different modulations does not present a linear response. One reason probably resides in soft decision issues since the quantizing is as better as higher is the constellation order. In future works, modulation schemes using N with odd values or even multidimensional constellations could be used to establish a better distribution of spectral efficiency into these gaps.

The number of MCS is 22, while the number of SINR positions is 280, considering the range from 0 to 28 dB with steps of 0.1 dB. The maximum value used for the index of ECA is 5.0, where the unit value corresponds to a target rate of BER equal to 10^{-6}.



```
[-80. -79. -78. -77. -76. -75. -74. -73. -72. -71. -70. -69. -68. -67.
 -66. -65. -64. -63. -62. -61. -60. -59. -58. -57. -56. -55. -54. -53.
 -52. -51. -50. -49. -48. -47. -46. -45. -44. -43. -42. -41. -40. -39.
 -38. -37. -36. -35. -34. -33. -32. -31. -30. -29. -28. -27. -26. -25.
 -24. -23. -22. -21.   0.   0.   0.   0. 284.  40.  40.  40.  40.  40.
  40.  40.  40.  40.  40.  40.  40.  40.  40.  40.]
```

Figure 5.8: *Reward vector: MCS 21 from 20dB to 28 dB.*

The reward vector, partially shown in Figure 5.8, considers the spectral efficiency, which increases with the MCS index. Regarding the SINR, there is a gradual decrease in negative rewards until zero values are reached in the region that will be controlled by the ECA indicator.

The target of BER = 10^{-6} was taken as an example in this implementation, that is when the value of ECA = 1.0. Finally, the state with a peak reward value is fixed exactly at this point, aiming to create an array of *Q-values* that generates an inference of actions in order to converge to this state.

For the simulation and execution of the Q-Learning algorithm, the Python language was used. The maximum value of ECA was set to 5.0, and the curve of MCSs decided by the algorithm, shown in Figure 5.9. The actions were obtained with the target value of ECA = 1.0, and it shows that the values obtained are adjusted with the target rate of BER = 10^{-6}.

The results demonstrate a consistent convergence of the Q-values. One drawback of this implementation is the quantized values of ECA and SINR. This aspect is intrinsic to the Q-Learning method. If more levels of states are aggregated, the lasting time to a fitted learning process can be very high.

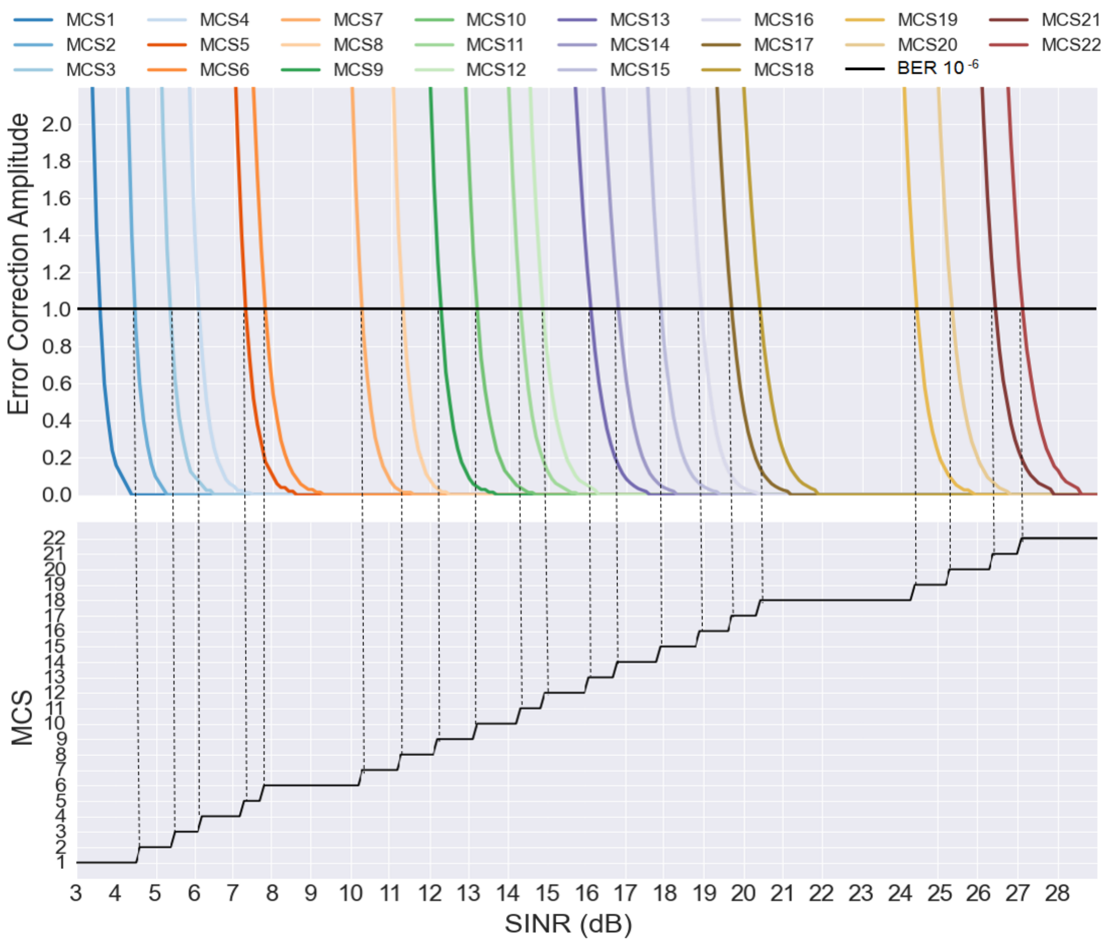


Figure 5.9: *Q-Learning algorithm response & ECA vs. SINR*

The best approach to optimize the LA control starting from this technique is the Deep Q-Learning [77]. In this case, the table of states and rewards is replaced by a Fully-Connected Neural Network (FCNN). Another convenience of this technique is the possibility of generating more information about multiple actions, which is a useful prerequisite for future proposals.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The complexity of the current and future mobile network slicing infrastructure will push on a full awareness of all involved processes. Traditional procedures and controlling will demand improvements and new methods to achieve the envisioned requirements.

The Machine Learning (ML) technology has been an important support for the evolution of diverse issues and fields. Certainly, it would not be different in the telecommunication scenarios. Since the high processing of some algorithms is a drawback in this area, mainly considering the Physical Layer environment, some solutions call for a low-complexity attribute.

Considering these aspects and the necessary balance of throughput and reliability for the Quality of Service (QoS) requirements, this work focused on Link Adaptation, whether proposing solutions in control systems or the accuracy of the involved indicators. The propositions related to these issues were implemented and tested.

As a result, a precise estimation of the Signal-to-Interference-plus-Noise Ratio, or primary indicator, was efficiently executed by adopting the proposed Modified Mean Square Error function plus a low-complexity Deep Learning Network. As could be seen, a Deep Neural Network (DNN) was successfully used to supersede a complex statistical computation even in low SINR. Also, it was verified that the same neural network topology could be easily applied to different modulation options using the respective trained biases and weights.

The subsequent proposition, referred to as the complementary indicator Error Correction Amplitude, presented a noticeable response in a way it can be used as a standalone control scheme. Again, a low-complexity DNN similar to the one used in the primary indicator was capable of establishing a normalized indicator that is not dependent on the current Modulation and Coding Scheme (MCS) parameter. Moreover, this Neural Network can also be fitted and calibrated to different transceiver implementations and different Bit Error Rate (BER) targets.

The PoC's simulations show that with the utilization of the ECA indicator for the link adaptation and a straightforward control, there is an improvement in throughput by keeping the demanded reliability strictly. Its application is also an interesting strategy to deal with low error rates since it can estimate error probability without bit error occurrence. Other verified benefits of this indicator are low processing and no payload consumption.

Finally, to integrate the proposed indicators into a control structure, some schemes were suggested either conceptually or also using ML techniques. The minimal one, using the ECA indicator as a unique control input, led to an implementation of a preprocessing block applying another low-complexity DNN. This procedure was successfully achieved, indicating the versatility of ML techniques.

A reinforcement learning (RL) scheme was also implemented using a basic Q-Learning method. A consistent result was carried out, which is a prevalidation of the use of a more efficient algorithm using Deep Q-Learning (DQL). This enhanced technique is key to triggering a proposition for future work where the concept of Indirect Decision for the MCS is established. In this scheme, the Channel Quality Indicator (CQI) provides probability options of target achievement instead of a unitary MCS index. It will be better described in the next section.

6.2 Future Work

The improvements of LA control capabilities go through providing more complete data for the Base Station (BS) about the conditions of the receiver on the link. In this way, decisions can be better taken when other statistics of the entire environment are gathered to this information. The system can assume greater or lesser risks, as well as higher or lower data rates, depending on the slice of service and the current usage of the network resources. It is a complex task to be executed. Anyway and again, ML technique is a candidate to support this process in a successful arrangement.

More complete information related to this approach can only be provided in terms of Physical Layer environment. Once this data compilation is available, processing methods are necessary to describe the link status better. Sending this processed info to the BS instead of a decided MCS index generates a more complete insight, providing a better scenario for decision-making. A succinct diagram is shown in Figure 6.1.

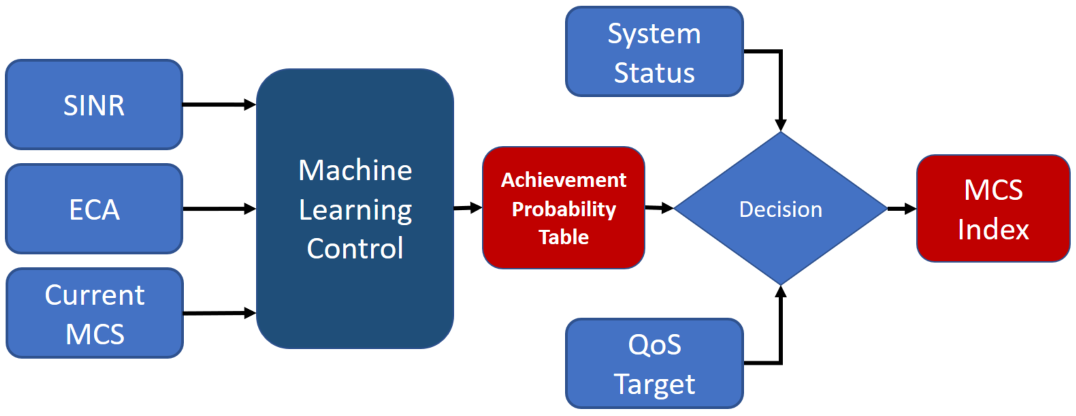


Figure 6.1: *Indirect decision: MCS table with achievement probabilities.*

As mentioned in Chapter 5, the LA control employing DQL is a robust technique to achieve superior performance. At the same time, this scheme can also deliver the probability of achieving goals for each MCS index or a specific range of indexes. In this case, actions can subsequently be taken according to this table and other system QoS indicators.

To succeed in this approach, the DQL algorithm is effective too. Thus, the achievement probability table will be generated by collecting the Q-values of the desired MCS indexes. A normalization function must be included. A Softmax function is an example of how this calculation can be executed. Figure 6.2 better displays the diagram for this implementation.

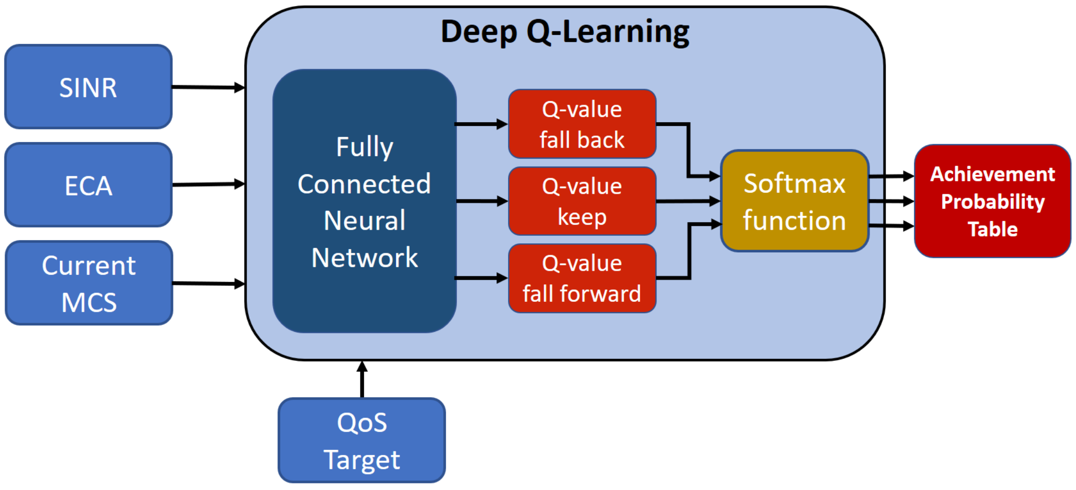


Figure 6.2: *Proposed scheme using Deep Q-Learning and Softmax function.*

Among other implementations that can be performed for future work is a reinforcement learning method to execute a Direct Decision with no unitary MCS index steps. This would also be possible using a DQL technique.

The gap variance verified in ECA performance between some MCS indexes is due to the exclusion of intermediate modulations. All the chosen options of modulation order were based on the current mobile network standards (Fourth Generation of Mobile Networks (4G) and Fifth Generation of Mobile Networks (5G)). The options have constellations of N points where $N = 2^M$ and M is the modulation order. The modulation order, M, is an even value from 2 to 8, generating the modulation types: QPSK, 16-QAM, 64-QAM, and 256-QAM, respectively. This way, the constellation shapes are always squared. These are not mandatory selections. Then better results could be achieved with odd values of M or using multidimensional constellation schemes.

Additionally, the MCS schemes could apply independent options of modulation and coding. This is another advantageous approach to be evaluated since more efficient choices can be found depending on constellation order and shape, plus different forward error correction (FEC) rates.

As the Fifth Generation of Mobile Networks did not fully adopt the polar code for the payload, other forward error correction coding techniques, such as low density parity check (LDPC) and Turbo Code, can also be implemented and tested to provide the ECA indicator using similar approaches executed in Chapters 4 and 5.

References

[1] Y. Shi, L. Lian, Y. Shi, Z. Wang, Y. Zhou, L. Fu, L. Bai, J. Zhang, and W. Zhang, "Machine Learning for Large-Scale Optimization in 6G Wireless Networks," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2023.

[2] M. Kulin, T. Kazaz, E. De Poorter, and I. Moerman, "A survey on machine learning-based performance improvement of wireless networks: Phy, mac and network layer," *Electronics*, vol. 10, no. 3, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/3/318

[3] P. Sethuraman, "Machine learning and signal processing," 2020. [Online]. Available: https://towardsdatascience.com/machine-learning-and-signal-processing-103281d27c4b

[4] D. M. Molla, H. Badis, L. George, and M. Berbineau, "Software defined radio platforms for wireless technologies," *IEEE Access*, vol. 10, pp. 26 203–26 229, 2022.

[5] T. Ulversoy, "Software defined radio: Challenges and opportunities," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 4, pp. 531–550, 2010.

[6] R. Akeela and B. Dezfouli, "Software-defined radios: Architecture, state-of-the-art, and challenges," *Computer Communications*, vol. 128, pp. 106–125, 2018.

[7] T. Erpek, T. J. O'Shea, Y. E. Sagduyu, Y. Shi, and T. C. Clancy, "Deep learning for wireless communications," 2020.

[8] C. V. Gonzalez Zelaya, "Towards explaining the effects of data preprocessing on machine learning," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 2086–2090.

[9] Horizon Europe 2020, "Remote Area Access Network for 5th Generation," 2017-2020. [Online]. Available: https://doi.org/10.3030/777137

[10] G. Fettweis, M. Krondorf, and S. Bittner, "GFDM - Generalized Frequency Division Multiplexing," in *VTC Spring 2009 - IEEE 69th Vehicular Technology Conference*, 2009, pp. 1–4.

[11] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *CoRR*, vol. abs/0807.3917, 2008. [Online]. Available: http://arxiv.org/abs/0807.3917

[12] A. Cyriac and G. Narayanan, “Polar code encoder and decoder implementation,” in *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*, 2018, pp. 294–302.

[13] W. Dias, A. Ferreira, R. Kagami, J. S. Ferreira, D. Silva, and L. Mendes, “5G-RANGE: A transceiver for remote areas based on software-defined radio,” in *2020 European Conference on Networks and Communications (EuCNC)*, 2020, pp. 100–104.

[14] W. Dias, D. Gaspar, L. Mendes, M. Chafii, M. Matthé, P. Neuhaus, and G. Fettweis, “Performance Analysis of a 5G Transceiver Implementation for Remote Areas Scenarios,” in *2018 European Conference on Networks and Communications (EuCNC)*, 2018, pp. 363–367.

[15] S. Gueron and V. Krasnov, “Fast Quicksort Implementation Using AVX Instructions,” *The Computer Journal*, vol. 59, no. 1, pp. 83–90, 2016.

[16] C. S. Anderson, J. Zhang, and M. Cornea, “Enhanced Vector Math Support on the Intel®AVX-512 Architecture,” in *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, 2018, pp. 120–124.

[17] Nvidia, *Geforce RTX 3070 Specifications*, 2023. [Online]. Available: https://www.nvidia.com/en-eu/geforce/graphics-cards/30-series/rtx-3070-3070ti

[18] National Instruments, *USRP-2954 Specifications*, 2023. [Online]. Available: https://www.ni.com/docs/en-US/bundle/usrp-2954-specs/page/specs.html

[19] GNU Radio Website, accessed July 2020. [Online]. Available: http://www.gnuradio.org

[20] S. K. Lam, A. Pitrou, and S. Seibert, “Numba: A llvm-based python jit compiler,” in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6.

[21] J. Demel, M. Dickens, D. Anderson, B. Ashton, P. Balister, D. Behar, S. Behnke, A. Bekhit, A. Bhowmick, E. Blossom, J. Blum, A. M. Bottoms, E. Briggs, J. Cardoso, P. Cercueil, J. Corgan, N. Corgan, L. Cruz, R. Economos, B. P. Enochs, C. Fernandez, M. Fischer, N. Foster, D. Geiger, P. Giard, G. Goavec-Merou, B. Hilburn, A. Holguin, J. Iwamoto, M. Kaesberger, M. Lichtman, K. A. Logue, M. Lundmark, S. Markgraf, C. Mayer, N. McCarthy, N. McCarthy, D. Miralles, S. Munaut, M. Müller, G. Nieboer,

T. O'Shea, J. Olivain, S. Oltmanns, J. Pinkava, M. Piscopo, J. M. H. Quiceno, M. Rene, F. Ritterhoff, D. Robertson, F. L. L. Rocca, A. Rode, A. Rodionov, T. Rondeau, T. Sekine, K. Semich, V. Sergeev, A. Slokva, C. Smith, A. Stigo, A. Thompson, R. Thompson, V. Velichkov, R. Volz, A. Walls, D. Ward, N. West, B. M. Wiedemann, S. Wunsch, V. Zapodovnikov, J. Škarvada, Aang23, AlexandreRouma, Andrew, Zlika, luz.paz, and rear1019, "Vector-optimized library of kernels (volk)," Feb. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6052858

[22] *Intel® Intrinsics Guide*, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

[23] M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[24] E. Arikan, "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[25] A. Cassagne, O. Hartmann, M. Leonardon, K. He, C. Leroux, R. Tajan, O. Aumage, D. Barthou, T. Tonnellier, V. Pignoly *et al.*, "AFF3CT: A Fast Forward Error Correction Toolbox!" *SoftwareX*, vol. 10, p. 100345, 2019.

[26] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, 2014.

[27] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "Unrolled polar decoders, part i: Hardware architectures," *ArXiv*, vol. abs/1505.01459, 2015.

[28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[29] Google, "TensorFlow Community." [Online]. Available: https://www.tensorflow.org/community

[30] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[31] Google, *XLA GitHub repository*. [Online]. Available: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/compiler/xla

[32] A. M. Khan, V. Jeoti, M. Rehman, and M. Jilani, "Noise power estimation for broadcasting ofdm systems," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2017, pp. 1–6.

[33] D. Wu, H. Shao, F. Yang, and L. Cui, "An improved snr estimator for wireless ofdm systems," *Procedia Engineering*, vol. 29, pp. 3124–3131, 2012, 2012 International Workshop on Information and Electronics Engineering. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877705812004626

[34] S. Baumgartner, G. Hirtz, and A. Baumgartner, "A modified maximum likelihood method for snr estimation in ofdm based systems," in *2014 IEEE International Conference on Consumer Electronics (ICCE)*, 2014, pp. 155–158.

[35] S. Malik, S. Portugal, C. Seo, C. Kim, and I. Hwang, "Proposal and performance analysis of a novel preamble-based snr estimation algorithm," in *2011 34th International Conference on Telecommunications and Signal Processing*, 2011, pp. 100–103.

[36] S. Boumard, "Novel Noise Variance and SNR Estimation Algorithm for Wireless MIMO OFDM Systems," in *GLOBECOM '03. IEEE Global Telecommunications Conference (IEEE Cat. No.03CH37489)*, vol. 3, 2003, pp. 1330–1334 vol.3.

[37] G. Ren, Y. Chang, and H. Zhang, "Snr estimation algorithm based on the preamble for wireless ofdm systems," *Science in China Series F: Information Sciences*, vol. 51, no. 7, pp. 965–974, Jul 2008. [Online]. Available: https://doi.org/10.1007/s11432-008-0063-8

[38] G. Ren, H. Zhang, and Y. Chang, "Snr estimation algorithm based on the preamble for ofdm systems in frequency selective channels," *IEEE Transactions on Communications*, vol. 57, no. 8, pp. 2230–2234, 2009.

[39] M. Zivkovic and R. Mathar, "An improved preamble-based snr estimation algorithm for ofdm systems," in *21st Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2010, pp. 172–176.

[40] F.-X. Socheleau, A. Aissa-El-Bey, and S. Houcke, "Non data-aided snr estimation of ofdm signals," *IEEE Communications Letters*, vol. 12, no. 11, pp. 813–815, 2008.

[41] X. Xie, S. Peng, and X. Yang, “Deep learning-based signal-to-noise ratio estimation using constellation diagrams,” *Mobile Information Systems*, vol. 2020, p. 8840340, Nov 2020. [Online]. Available: https://doi.org/10.1155/2020/8840340

[42] T. Ngo, B. Kelley, and P. Rad, “Deep learning based prediction of signal-to-noise ratio (snr) for lte and 5g systems,” in *2020 8th International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2020, pp. 1–6.

[43] A. Landi, P. Piaggi, M. Laurino, and D. Menicucci, “Artificial Neural Networks for Nonlinear Regression and Classification,” 11 2010, pp. 115–120.

[44] Bin Li, R. DiFazio, and A. Zeira, “A low bias algorithm to estimate negative snrs in an awgn channel,” *IEEE Communications Letters*, vol. 6, no. 11, pp. 469–471, Nov 2002.

[45] 3GPP TS 38.211 version 15.2.0 Release 15, *Physical channels and modulation*, July 2018. [Online]. Available: http://www.etsi.org

[46] R. P. McDonald, “A general Approach to Nonlinear Factor Analysis,” *Psychometrika*, vol. 27, no. 4, pp. 397–415, Dec 1962. [Online]. Available: https://doi.org/10.1007/BF02289646

[47] J. G. Nemeth, M. Al–Imari, and W. Ozan, “Soft-ACK Feedback Based Link Adaptation for Latency Critical Applications in 5G/B5G,” in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 01–06.

[48] Y. Abdelkader and E. Jamal, “Optimal Spacing Design for Pilots in OFDM Systems over Multipath Fading Channels,” vol. 2, no. 4, 2010, pp. 221–229.

[49] M. López-Benítez, “Performance Analysis of SNR Threshold-setting Strategies for Adaptive Modulation and Coding Under Fading Channels,” *Physical Communication*, vol. 30, pp. 154–166, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1874490718302532

[50] J. Nilsson, E. Mobile, C. Ab, B. Wittenmark, M. Törngren, and M. Sanfridson, “Timing Problems in Real-Time Control Systems,” 05 2001.

[51] S. K. Pulliyakode, S. Kalyani, and K. Narendran, “Rate prediction and selection in lte systems using modified source encoding techniques,” *IEEE Transactions on Wireless Communications*, vol. 15, no. 1, pp. 416–429, 2016.

[52] J. Park and S. Baek, “Two-Stage Thompson Sampling for Outer-Loop Link Adaptation,” *IEEE Wireless Communications Letters*, vol. 10, no. 9, pp. 2004–2008, 2021.

[53] B.-C. et al., “eOLLA: an enhanced Outer Loop Link Adaptation for Cellular Net-

works," *EURASIP Journal on Wireless Communications and Networking*, 2016.

[54] M. G. Sarret, D. Catania, F. Frederiksen, A. F. Cattoni, G. Berardinelli, and P. Mogensen, "Dynamic Outer Loop Link Adaptation for the 5G Centimeter-Wave Concept," in *Proceedings of European Wireless 2015; 21th European Wireless Conference*, 2015, pp. 1–6.

[55] S. Park, R. C. Daniels, and R. W. Heath, "Optimizing the Target Error Rate for Link Adaptation," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[56] V. Saxena and J. Jaldén, "Bayesian Link Adaptation under a BLER Target," in *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2020, pp. 1–5.

[57] A. Durán, M. Toril, F. Ruiz, and A. Mendo, "Self-Optimization Algorithm for Outer Loop Link Adaptation in LTE," *IEEE Communications Letters*, vol. 19, no. 11, pp. 2005–2008, 2015.

[58] F. J. Martín-Vega, J. C. Ruiz-Sicilia, M. C. Aguayo, and G. Gómez, "Emerging Tools for Link Adaptation on 5G NR and Beyond: Challenges and Opportunities," *IEEE Access*, vol. 9, pp. 126 976–126 987, 2021.

[59] P. S, J. Khan, and L. Jacob, "Reinforcement Learning Based Link Adaptation in 5G URLLC," in *2021 8th International Conference on Smart Computing and Communications (ICSCC)*, 2021, pp. 159–163.

[60] M. P. Mota, D. C. Araujo, F. H. Costa Neto, A. L. F. de Almeida, and F. R. Cavalcanti, "Adaptive Modulation and Coding Based on Reinforcement Learning for 5G Networks," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019, pp. 1–6.

[61] P. S. Pati, S. S. Sahoo, D. Krishnaswamy, and R. Datta, "A Novel Machine Learning Approach for Link Adaptation in 5G Wireless Networks," in *2020 2nd PhD Colloquium on Ethically Driven Innovation and Technology for Society (PhD EDITS)*, 2020, pp. 1–2.

[62] V. Saxena, H. Tullberg, and J. Jaldén, "Reinforcement Learning for Efficient and Tuning-Free Link Adaptation," *IEEE Transactions on Wireless Communications*, pp. 1–1, 2021.

[63] A. Cassagne, O. Hartmann, M. Leonardon, K. He, C. Leroux, R. Tajan, O. Aumage, D. Barthou, T. Tonnellier, V. Pignoly *et al.*, "AFF3CT: A Fast Forward Error Correction Toolbox!" *SoftwareX*, p. 100345, 2019.

[64] R. Kagami and L. Mendes, "A Low-Complexity Deep Neural Network for

Signal-to-Interference-Plus-Noise Ratio Estimation," in *Anais do I Workshop de Redes 6G*. Porto Alegre, RS, Brasil: SBC, 2021, pp. 1–6. [Online]. Available: https://sol.sbc.org.br/index.php/w6g/article/view/17227

[65] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire, "Gambling in a Rigged Casino: The Adversarial Multi-Armed Bandit Problem," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 1995, pp. 322–331.

[66] V. Saxena, J. Jaldén, J. E. Gonzalez, M. Bengtsson, H. Tullberg, and I. Stoica, "Contextual Multi-Armed Bandits for Link Adaptation in Cellular Networks," ser. NetAI'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 44–49. [Online]. Available: https://doi.org/10.1145/3341216.3342212

[67] S. K. Pulliyakode and S. Kalyani, "Reinforcement Learning Techniques for Outer Loop Link Adaptation in 4G/5G Systems," 2017.

[68] F. Bois, "Bayesian inference," *Methods in molecular biology (Clifton, N.J.)*, vol. 930, pp. 597–636, 01 2013.

[69] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Oxford, 1989.

[70] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A Theoretical Analysis of Deep Q-Learning," 2020.

[71] E. Duryea, M. Ganger, and W. Hu, "Exploring Deep Reinforcement Learning with Multi Q-Learning," *Intelligent Control and Automation*, vol. 07, pp. 129–144, 01 2016.

[72] F. Geiser, D. Wessel, M. Hummert, A. Weber, D. Wübben, A. Dekorsy, and A. Viseras, "DRLLA: Deep Reinforcement Learning for Link Adaptation," *Telecom*, vol. 3, no. 4, pp. 692–705, 2022. [Online]. Available: https://www.mdpi.com/2673-4001/3/4/37

[73] C. Hellings, A. Dehmani, S. Wesemann, M. Koller, and W. Utschick, "Evaluation of neural-network-based channel estimators using measurement data," in *WSA 2019; 23rd International ITG Workshop on Smart Antennas*, 2019, pp. 1–5.

[74] C. Wang, G. Wang, H. Wang, A. Chen, and R. Santiago, "Quality of service (qos) contract specification, establishment, and monitoring for service level management," in *2006 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06)*, 2006, pp. 49–49.

[75] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[76] R. Bellman, “Some Applications of the Theory of Dynamic Programming - A Review,” *Oper. Res.*, vol. 2, no. 3, pp. 275–288, 1954. [Online]. Available: https://doi.org/10.1287/opre.2.3.275

[77] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An Introduction to Deep Reinforcement Learning,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018. [Online]. Available: https://doi.org/10.1561%2F2200000071

Appendix I

Codes

I.1 Deep Neural Network for SINR Estimation

I.1.1 Learning Code: QPSK

```
import numpy as np
from sklearn import preprocessing
from matplotlib import pyplot
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.models import Sequential
import keras
from datetime import datetime
from packaging import version

# Preraring dataset
X_l = np.loadtxt("adapted_qpsk.csv")
y_l = np.loadtxt("diff_qpsk.csv")
logdir = "logs/scalars/" + "QPSK"
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)

X = X_l[200:800]
y = y_l[200:800]

# Data Scaling from 0 to 1, X and y originally have very different scales.
X_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
y_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
X_scaled = (X_scaler.fit_transform(X.reshape(-1,1)))
y_scaled = (y_scaler.fit_transform(y.reshape(-1,1)))
```

```
# New sequential network structure.
model = Sequential()

# Input layer with dimension 1 and hidden layer i with 6 neurons.
model.add(Dense(1, kernel_initializer='uniform', activation='linear'))
# Hidden layer j with 4 neurons plus activation layer.
model.add(Dense(4, activation='linear'))
# Hidden layer k with 4 neurons.
model.add(Dense(4, activation='sigmoid'))
# Output Layer.
model.add(Dense(1))

# Model is derived and compiled using mean square error as loss
# function, accuracy as metric and gradient descent optimizer.
model.compile(loss='mse', optimizer='adam', metrics=["accuracy"])

# Training model with train data. Fixed random seed:
#np.random.seed(123)
model.fit(X_scaled, y_scaled, epochs=2000, batch_size=2, callbacks=[tensorboard_c

# Serialize model to JSON
model_json = model.to_json()
with open("model_qpsk_144d.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_qpsk_144d.h5")
print("Saved model to disk")

# Predict the response variable with new data
predicted = model.predict(X_scaled)

# Plot in blue color the predicted adata and in green color the
# actual data to verify visually the accuracy of the model.
pyplot.plot(y_scaler.inverse_transform(predicted), color="red")
pyplot.plot(y_scaler.inverse_transform(y_scaled), color="green")
pyplot.legend(('Predicted', 'Data'), loc='lower right')
pyplot.show()
```

I.1.2 Learning Code: 16-QAM

```
import numpy as np
from sklearn import preprocessing
from matplotlib import pyplot
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.models import Sequential
import keras
from datetime import datetime
from packaging import version

# Preraring dataset
X_l = np.loadtxt("adapted_16qam.csv")
y_l = np.loadtxt("diff_16qam.csv")
logdir = "logs/scalars/" + "16QAM"
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)

X = X_l[200:800]
y = y_l[200:800]

# Data Scaling from 0 to 1, X and y originally have very different scales.
X_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
y_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
X_scaled = (X_scaler.fit_transform(X.reshape(-1,1)))
y_scaled = (y_scaler.fit_transform(y.reshape(-1,1)))

# New sequential network structure.
model = Sequential()

# Input layer with dimension 1 and hidden layer i with 6 neurons.
model.add(Dense(1, kernel_initializer='uniform', activation='linear'))
# Hidden layer j with 4 neurons plus activation layer.
model.add(Dense(4, activation='linear'))
# Hidden layer k with 4 neurons.
model.add(Dense(4, activation='sigmoid'))
# Output Layer.
model.add(Dense(1))

# Model is derived and compiled using mean square error as loss
# function, accuracy as metric and gradient descent optimizer.
model.compile(loss='mse', optimizer='adam', metrics=["accuracy"])

# Training model with train data. Fixed random seed:
#np.random.seed(123)
model.fit(X_scaled, y_scaled, epochs=2000, batch_size=2, callbacks=[tensorboard_c

# Serialize model to JSON
model_json = model.to_json()
with open("model_16qam_144d.json", "w") as json_file:
    json_file.write(model_json)
```

```
# serialize weights to HDF5
model.save_weights("model_16qam_144d.h5")
print("Saved model to disk")

# Predict the response variable with new data
predicted = model.predict(X_scaled)

# Plot in blue color the predicted adata and in green color the
# actual data to verify visually the accuracy of the model.
pyplot.plot(y_scaler.inverse_transform(predicted), color="red")
pyplot.plot(y_scaler.inverse_transform(y_scaled), color="green")
pyplot.legend(('Predicted', 'Data'), loc='upper right')
pyplot.show()
```

I.1.3 Learning Code: 64-QAM

```
import numpy as np
from sklearn import preprocessing
from matplotlib import pyplot
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.models import Sequential
import keras
from datetime import datetime
from packaging import version

# Prerating dataset
X_l = np.loadtxt("adapted_64qam.csv")
y_l = np.loadtxt("diff_64qam.csv")
logdir = "logs/scalars/" + "64QAM"
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)

X = X_l[200:800]
y = y_l[200:800]

# Data Scaling from 0 to 1, X and y originally have very different scales.
X_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
y_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
X_scaled = (X_scaler.fit_transform(X.reshape(-1,1)))
y_scaled = (y_scaler.fit_transform(y.reshape(-1,1)))

# New sequential network structure.
model = Sequential()
```

```
# Input layer with dimension 1 and hidden layer i with 6 neurons.
model.add(Dense(1, kernel_initializer='uniform', activation='linear'))
# Hidden layer j with 4 neurons plus activation layer.
model.add(Dense(4, activation='linear'))
# Hidden layer k with 4 neurons.
model.add(Dense(4, activation='sigmoid'))
# Output Layer.
model.add(Dense(1))

# Model is derived and compiled using mean square error as loss
# function, accuracy as metric and gradient descent optimizer.
model.compile(loss='mse', optimizer='adam', metrics=["accuracy"])

# Training model with train data. Fixed random seed:
#np.random.seed(123)
model.fit(X_scaled, y_scaled, epochs=2000, batch_size=2, callbacks=[tensorboard_c

# Serialize model to JSON
model_json = model.to_json()
with open("model_64qam_144d.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_64qam_144d.h5")
print("Saved model to disk")

# Predict the response variable with new data
predicted = model.predict(X_scaled)

# Plot in blue color the predicted adata and in green color the
# actual data to verify visually the accuracy of the model.
pyplot.plot(y_scaler.inverse_transform(predicted), color="red")
pyplot.plot(y_scaler.inverse_transform(y_scaled), color="green")
pyplot.legend(('Predicted', 'Data'), loc='lower right')
pyplot.show()
```

I.1.4 Learning Code: 256-QAM

```
import numpy as np
from sklearn import preprocessing
from matplotlib import pyplot
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.models import Sequential
import keras
```

```
from datetime import datetime
from packaging import version

# Prerаring dataset
X_l = np.loadtxt("adapted_256qam.csv")
y_l = np.loadtxt("diff_256qam.csv")
logdir = "logs/scalars/" + "256QAM"
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)

X = X_l[200:800]
y = y_l[200:800]

# Data Scaling from 0 to 1, X and y originally have very different scales.
X_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
y_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
X_scaled = (X_scaler.fit_transform(X.reshape(-1,1)))
y_scaled = (y_scaler.fit_transform(y.reshape(-1,1)))

# New sequential network structure.
model = Sequential()

# Input layer with dimension 1 and hidden layer i with 6 neurons.
model.add(Dense(1, kernel_initializer='uniform', activation='linear'))
# Hidden layer j with 4 neurons plus activation layer.
model.add(Dense(4, activation='linear'))
# Hidden layer k with 4 neurons.
model.add(Dense(4, activation='sigmoid'))
# Output Layer.
model.add(Dense(1))

# Model is derived and compiled using mean square error as loss
# function, accuracy as metric and gradient descent optimizer.
model.compile(loss='mse', optimizer='adam', metrics=["accuracy"])

# Training model with train data. Fixed random seed:
#np.random.seed(123)
model.fit(X_scaled, y_scaled, epochs=2000, batch_size=2, callbacks=[tensorboard_c

# Serialize model to JSON
model_json = model.to_json()
with open("model_256qam_144d.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_256qam_144d.h5")
print("Saved model to disk")
```

```
# Predict the response variable with new data
predicted = model.predict(X_scaled)

# Plot in blue color the predicted adata and in green color the
# actual data to verify visually the accuracy of the model.
pyplot.plot(y_scaler.inverse_transform(predicted), color="red")
pyplot.plot(y_scaler.inverse_transform(y_scaled), color="green")
pyplot.legend(('Predicted', 'Data'), loc='lower right')
pyplot.show()
```

I.2 Deep Neural Network for SINR Estimation - Inference Block.

```
import pmt
import numpy as np
import tensorflow as tf
from gnuradio import gr
from tensorflow.python.ops.numpy_ops import np_config
from sklearn import preprocessing
from keras.models import model_from_json
from numba import jit

class blk(gr.sync_block):
  def __init__(self, dnn_dis=0.0):    # DNN disable parameter
    gr.sync_block.__init__(
      self,
      name='SINR Estimation-DNN',
      in_sig=[np.complex64],
      out_sig=[np.float32, np.float32]
    )

    gpus = tf.config.list_physical_devices('GPU')
    if gpus:
      try:
        for gpu in gpus:
          tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus),\
          "Logical GPUs")
      except RuntimeError as e:
        print(e)
    self.mcs = 0
    self.mcs_d = 0
    self.modul = 0
    self.modul_d = 0
    self.message_port_register_in(pmt.intern('msg_in'))
    self.set_msg_handler(pmt.intern('msg_in'), self.handle_msg)

    # if an attribute with the same name as a parameter is found,
    # a callback is registered (properties work, too).
    self.dnn_dis = dnn_dis

    # Preraring dataset

    # QPSK
    X_l = np.loadtxt("adapted_qpsk.csv")
    y_l = np.loadtxt("diff_qpsk.csv")
    X = X_l[200:800]
    y = y_l[200:800]

    self.X_scaler_qpsk = preprocessing.MinMaxScaler(feature_range=(0, 1))
    self.y_scaler_qpsk = preprocessing.MinMaxScaler(feature_range=(0, 1))
    self.X_scaler_qpsk.fit(X.reshape(-1,1))
    self.y_scaler_qpsk.fit(y.reshape(-1,1))

    # 16QAM
    X_l = np.loadtxt("adapted_16qam.csv")
    y_l = np.loadtxt("diff_16qam.csv")
    X = X_l[200:800]
    y = y_l[200:800]

    self.X_scaler_16 = preprocessing.MinMaxScaler(feature_range=(0, 1))
    self.y_scaler_16 = preprocessing.MinMaxScaler(feature_range=(0, 1))
    self.X_scaler_16.fit(X.reshape(-1,1))
    self.y_scaler_16.fit(y.reshape(-1,1))

    # 64QAM
    X_l = np.loadtxt("adapted_64qam.csv")
    y_l = np.loadtxt("diff_64qam.csv")
    X = X_l[200:800]
    y = y_l[200:800]

    self.X_scaler_64 = preprocessing.MinMaxScaler(feature_range=(0, 1))
    self.y_scaler_64 = preprocessing.MinMaxScaler(feature_range=(0, 1))
    self.X_scaler_64.fit(X.reshape(-1,1))
    self.y_scaler_64.fit(y.reshape(-1,1))
```

```
# 256QAM
X_l = np.loadtxt("adapted_256qam.csv")
y_l = np.loadtxt("diff_256qam.csv")
X = X_l[200:800]
y = y_l[200:800]

self.X_scaler_256 = preprocessing.MinMaxScaler(feature_range=(0, 1))
self.y_scaler_256 = preprocessing.MinMaxScaler(feature_range=(0, 1))
self.X_scaler_256.fit(X.reshape(-1,1))
self.y_scaler_256.fit(y.reshape(-1,1))

# load created models - QPSK
self.json_file = open('model_qpsk_144d.json', 'r')
self.model_qpsk_json = self.json_file.read()
self.json_file.close()
self.model_qpsk = model_from_json(self.model_qpsk_json)
# load weights into new model
self.model_qpsk.load_weights("model_qpsk_144d.h5")

# load created models - 16QAM
self.json_file = open('model_16qam_144d.json', 'r')
self.model_16_json = self.json_file.read()
self.json_file.close()
self.model_16 = model_from_json(self.model_16_json)
# load weights into new model
self.model_16.load_weights("model_16qam_144d.h5")

# load created models - 64QAM
self.json_file = open('model_64qam_144d.json', 'r')
self.model_64_json = self.json_file.read()
self.json_file.close()
self.model_64 = model_from_json(self.model_64_json)
# load weights into new model
self.model_64.load_weights("model_64qam_144d.h5")

# load created models - 256QAM
self.json_file = open('model_256qam_144d.json', 'r')
self.model_256_json = self.json_file.read()
self.json_file.close()
self.model_256 = model_from_json(self.model_256_json)
# load weights into new model
self.model_256.load_weights("model_256qam_144d.h5")

# Default configuration
```

```
    self.X_scaler = self.X_scaler_qpsk
    self.y_scaler = self.y_scaler_qpsk
    self.model = self.model_qpsk
    self.max_value = np.float32(2.0)
    self.factor = np.float32(1.414213562)
    self.beta = np.float32(1.0)

  def handle_msg(self, msg_pmt):
    set_tx_mcs = pmt.intern("MCS")
    if (pmt.car(msg_pmt) == set_tx_mcs):
      msg = pmt.cdr(msg_pmt)
      print (msg)
      self.mcs_d = self.mcs
      self.mcs = pmt.to_python(msg)
    if self.mcs == self.mcs_d:
      return
    if self.mcs < 4:
      self.modul_d = self.modul
      self.modul = 0
    elif self.mcs < 8:
      self.modul_d = self.modul
      self.modul = 1
    elif self.mcs < 11:
      self.modul_d = self.modul
      self.modul = 2
    elif self.mcs < 14:
      self.modul_d = self.modul
      self.modul = 3
    if self.modul == self.modul_d:
      return
    if self.modul == 0:
      self.X_scaler = self.X_scaler_qpsk
      self.y_scaler = self.y_scaler_qpsk
      self.model = self.model_qpsk
      self.max_value = np.float32(2.0)
      self.factor = np.float32(1.414213562)
      self.beta = np.float32(1.0)
      print ("***** config DNN: QPSK *****")
    elif self.modul == 1:
      self.X_scaler = self.X_scaler_16
      self.y_scaler = self.y_scaler_16
      self.model = self.model_16
      self.max_value = np.float32(4.0)
      self.factor = np.float32(3.16227766)
      self.beta = np.float32(2.0)
      print ("***** config DNN: 16QAM *****")
    elif self.modul == 2:
      self.X_scaler = self.X_scaler_64
      self.y_scaler = self.y_scaler_64
      self.model = self.model_64
      self.max_value = np.float32(8.0)
      self.factor = np.float32(6.4807407)
      self.beta = np.float32(3.0)
      print ("***** config DNN: 64QAM *****")
    elif self.modul == 3:
      self.X_scaler = self.X_scaler_256
      self.y_scaler = self.y_scaler_256
      self.model = self.model_256
      self.max_value = np.float32(16.0)
      self.factor = np.float32(13.038413)
      self.beta = np.float32(4.0)
      print ("***** config DNN: 256QAM *****")
  @jit
  def mse_calc(self, n, complex_input):
    max_value = self.max_value
    factor = self.factor
    beta = self.beta
    mse = np.float32(0.0)
    for i in range(n):
      d_real = np.abs(np.real(complex_input[i]))*factor
      d_imag = np.abs(np.imag(complex_input[i]))*factor
      dec_real = np.round((d_real + 1.0)/2.0)*2.0 - 1.0
      comp_real = np.float32(0.0)
      if (d_real > max_value):
        dec_real = max_value - 1.0
        comp_real = np.abs(d_real - dec_real)
      dec_imag = np.round((d_imag + 1.0)/2.0)*2.0 - 1.0
      comp_imag = np.float32(0.0)
      if (d_imag > max_value):
        dec_imag = max_value - 1.0
        comp_imag = np.abs(d_imag - dec_imag)
      comp_real = beta*comp_real/factor
      comp_imag = beta*comp_imag/factor
      d_real = (d_real - dec_real)/factor
      d_imag = (d_imag - dec_imag)/factor
      mse = mse + d_real*d_real + d_imag*d_imag + comp_real*comp_real + \
            comp_imag*comp_imag
    return -10.0*np.log10(mse/n)

  @tf.function(jit_compile=True)   # XLA acceleration
```

```
  def predict(self, x):
    return self.model(x)

  def calc_error(self, snr):    # error prediction
    x = np.arange(1)
    x[0] = (snr)
    y = self.X_scaler.transform(x.reshape(-1, 1))
    pred = self.predict(y)
    scaled = self.y_scaler.inverse_transform(pred)
    return scaled

  def work(self, input_items, output_items):
    n = np.size(input_items[0])
    snr_adapt = self.mse_calc(n, input_items[0][:])
    dnn_comp = self.calc_error(snr_adapt)
    output_items[0][:] = snr_adapt
    output_items[1][:] = snr_adapt + dnn_comp
    return len(output_items[0])
```

I.3 Deep Neural Network description

I.3.1 DNN for SINR estimation - QPSK modulation.

```
name: 'dense'
trainable: True
dtype: float32
units: 1
activation: linear
use_bias: True
kernel_initializer: RandomUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[1.744565]], dtype=float32), array([-0.39607584], dtype=float32)]

name: 'dense_1'
trainable: True
dtype: float32
units: 4
activation: linear
```

```
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[-2.7459078, -2.691793 , -2.188306 ,  1.4207704]], dtype=float32), array(

name: 'dense_2'
trainable: True
dtype: float32
units: 4
activation: sigmoid
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[ 1.5428836 , -0.5844092 , -1.066457  ,  0.15086588],
       [ 0.7825846 , -2.0736282 , -1.4029425 , -4.669783  ],
       [-0.0183926 , -1.157625  , -1.4011582 , -2.3091369 ],
       [-0.3506581 ,  0.6435468 ,  1.949831  ,  0.95179504]], dtype=float32),
array([-0.89490986, -0.84202504,  0.63870823, -1.958233  ], dtype=float32)]

name: 'dense_3'
trainable: True
dtype: float32
units: 1
activation: linear
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None
```

```
[array([[ 0.87478286],
       [ 0.5331497 ],
       [-0.363003  ],
       [-0.36531928]], dtype=float32),
array([0.2349322], dtype=float32)]
```

I.3.2 DNN for SINR estimation - 16-QAM modulation.

```
name: 'dense'
trainable: True
dtype: float32
units: 1
activation: linear
use_bias: True
kernel_initializer: RandomUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[-1.5231512]], dtype=float32), array([0.48227042], dtype=float32)]

name: 'dense_1'
trainable: True
dtype: float32
units: 4
activation: linear
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[ 2.5494463,  1.6521053, -0.9304318, -2.0497146]], dtype=float32),
array([ 0.82721186,  0.27608192, -1.1282926 , -0.2633901 ], dtype=float32)]

name: 'dense_2'
trainable: True
dtype: float32
units: 4
```

```
activation: sigmoid
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[-2.62281   , -3.9025857 , -0.16771424,  0.91203034],
    [-1.4043089 , -2.220354  ,  0.5819991 ,  0.94500756],
    [ 0.6884257 ,  2.1576314 ,  0.94431645,  0.8787845 ],
    [ 1.3302621 ,  2.1502903 , -0.5246794 , -1.2474699 ]], dtype=float32),
array([-0.8571584 , -1.4458402 , -0.6312608 , -0.84736884], dtype=float32)]

name: 'dense_3'
trainable: True
dtype: float32
units: 1
activation: linear
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[ 0.4077577 ],
    [-0.40018824],
    [-0.2564823 ],
    [ 1.1334004 ]], dtype=float32),
array([0.02707014], dtype=float32)]
```

I.3.3 DNN for SINR estimation - 64-QAM modulation.

```
name: 'dense'
trainable: True
dtype: float32
units: 1
activation: linear
use_bias: True
kernel_initializer: RandomUniform
```

```
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[-1.5649678]], dtype=float32), array([0.6780824], dtype=float32)]

name: 'dense_1'
trainable: True
dtype: float32
units: 4
activation: linear
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[ 0.6845124 , -3.2464304 , -0.30720258,  1.2212253 ]],
      dtype=float32), array([ 0.33447587, -0.9892976 ,  0.8300899 ,  0.37142828],

name: 'dense_2'
trainable: True
dtype: float32
units: 4
activation: sigmoid
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[-1.2263339 , -0.736026  , -1.7499377 , -0.14865619],
       [ 1.5564194 ,  1.2173266 ,  4.9087057 , -1.4162921 ],
       [ 0.50010645,  1.0052675 , -1.1484196 , -2.531729  ],
       [-1.0023221 , -0.48698905, -5.23182   ,  0.63789666]], dtype=float32),
array([-0.17096059,  0.48393083, -1.5232935 , -1.6049603 ], dtype=float32)]
```

```
name: 'dense_3'
trainable: True
dtype: float32
units: 1
activation: linear
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[ 0.47277203],
       [-0.79940933],
       [-0.18062598],
       [ 0.5554822 ]], dtype=float32),
array([0.51508737], dtype=float32)]
```

I.3.4 DNN for SINR estimation - 256-QAM modulation.

```
name: 'dense'
trainable: True
dtype: float32
units: 1
activation: linear
use_bias: True
kernel_initializer: RandomUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[1.4903759]], dtype=float32), array([-0.6253101], dtype=float32)]

name: 'dense_1'
trainable: True
dtype: float32
units: 4
activation: linear
use_bias: True
```

```
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[-2.592138 , -2.7536142,  2.2150857, -2.2256777]], dtype=float32),
array([ 1.3707513 ,  0.86395025, -1.4262638 , -0.26397267], dtype=float32)]

name: 'dense_2'
trainable: True
dtype: float32
units: 4
activation: sigmoid
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None

[array([[ 2.8943214 , -1.5125563 , -0.0172313 ,  3.170016  ],
       [ 2.1509678 , -1.3592256 ,  0.51694417,  2.4633443 ],
       [-2.3070054 , -0.36173788,  0.1866049 , -2.7771604 ],
       [ 0.634983  , -1.3760759 ,  2.0622723 ,  1.2282077 ]], dtype=float32),
array([ 0.5810818 ,  1.0439905 , -0.88657016,  0.89210737], dtype=float32)]

name: 'dense_3'
trainable: True
dtype: float32
units: 1
activation: linear
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None
```

```
[array([[ 0.4171708 ],
       [-0.20888531],
       [ 0.848935  ],
       [-0.4000026 ]], dtype=float32),
array([0.20988683], dtype=float32)]
```

I.4 Machine Learning for Link Adaptation.

I.4.1 Direct Decision with Q-Learning

```
import numpy as np
import matplotlib.pyplot as plt

# Environment:
number_of_mcs = 22
number_of_snr = 280
eca_max = 5

# Actions:
#codes: 0 = stay (same MCS), 1 = up (fall forward), 2 = down (fall back)
actions = ['stay', 'up', 'down']

# Q-values: Q(s, a)
#the matrix contains 22 MCS indexes, 60 (SNR + ECA) values and 3 actions.
q_values = np.zeros((number_of_mcs, number_of_snr, 3))

# Rewards:
rewards = np.zeros((number_of_mcs, number_of_snr))
snr_limits = [3.8, 4.6, 5.5, 6.2, 7.3, 7.8, 10.3, 11.3, 12.2,\
              13.2, 14.3, 14.9, 16.1, 16.8, 17.9, 18.9, 19.7,\
              20.4, 24.4, 25.3, 26.4, 27.1]
for i in range (number_of_mcs):
    for j in range(number_of_snr):
        if (j > int(10*(snr_limits[i]))):
          rewards[i,j] = i + 20
        elif (j > int(10*(snr_limits[i]-0.5))):
          rewards[i,j] = 0
        else:
          rewards[i,j] = -280 + j
        if (j == int(10*snr_limits[i])):
          rewards[i,j] = i + j

print (rewards[20])
```

```
def get_random_position():
  return np.random.randint(number_of_mcs), np.random.randint(number_of_snr)

def get_next_action(current_mcs_index, current_snr_index, epsilon):
  if np.random.random() < epsilon:
    return np.argmax(q_values[current_mcs_index, current_snr_index])
  else:
    return np.random.randint(3)      #random action

def get_next_position(current_mcs_index, action_index):
  new_mcs_index = current_mcs_index
  if actions[action_index] == 'up'and current_mcs_index <(number_of_mcs - 1):
    new_mcs_index += 1
  elif actions[action_index] == 'down' and current_mcs_index > 0:
    new_mcs_index -= 1
  return new_mcs_index

def get_mcs(start_mcs_index, start_snr_index, eca_index):
  current_snr_index = int(10*(start_snr_index)) + int(0.5 - eca_index)
  current_mcs_index = start_mcs_index
  count_stay = 0
  count_max = 0
  while count_stay < 2:
    action_index = get_next_action(current_mcs_index, current_snr_index, 1.)
    if action_index == 0:
      count_stay += 1
    current_mcs_index = get_next_position(current_mcs_index, action_index)
    count_max += 1
    if (count_max > 20):
      break
  return current_mcs_index

# training parameters
discount_factor = 0.8
learning_rate = 0.9         #learning rate

# training
for episode in range(200000):
  mcs_index, snr_index = get_random_position()
  action_index = 1
  count = 0
  while count < 20:
    if episode < 150000:
      epsilon = 0.1
    else:
      epsilon = 0.9
    action_index = get_next_action(mcs_index, snr_index, epsilon)
    if action_index == 0:
      count += 1
    old_mcs_index, old_snr_index = mcs_index, snr_index
    mcs_index = get_next_position(mcs_index, action_index)
    reward = rewards[mcs_index, snr_index]
    old_q_value = q_values[old_mcs_index, old_snr_index, action_index]
    difference = reward + (discount_factor * np.max(q_values[mcs_index,\
        snr_index])) - old_q_value
    new_q_value = old_q_value + (learning_rate * difference)
    q_values[old_mcs_index, old_snr_index, action_index] = new_q_value
print('Training complete!')

decided_mcs = np.zeros(number_of_snr)
for i in range(number_of_snr):
    decided_mcs[i] = get_mcs(0,i/10.0,1.0)
print (decided_mcs)
np.savetxt('decided.txt', decided_mcs, delimiter=',')

# data read
file = "decided.txt"

y = np.loadtxt(file)
x = np.arange(len(y))
x1 = x[30:280]/10.0
y1 = y[30:280]+1

# style
plt.style.use('seaborn-darkgrid')
plt.figure(figsize=(14,5))
plt.plot(x1, y1, color="black", linewidth=1.4, alpha=1)
plt.xlabel('Signal to Noise Ratio (dB)', fontsize=20)
plt.ylabel('MCS', fontsize=20)
plt.xticks(np.arange(3,29,1), fontsize=15)
plt.yticks(np.arange(22)+1, fontsize=15)
plt.xlim([3.0, 28])
plt.savefig("decided_mcs")
plt.show()
```

I.4.2 Indirect Decision with achievement index output

Learning Code

```
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
import numpy as np
from keras.models import load_model
import os

n_inputs = 2
# Load dataset
dataset = np.loadtxt("train_eca_ber.csv", delimiter=",")
# Split into input (X) and output (Y) variables
X = dataset[:,0:n_inputs]
Y = dataset[:,n_inputs]
# Create model
model = Sequential()
model.add(Dense(4, input_dim=n_inputs, activation='linear'))
model.add(Dense(4, activation='sigmoid'))
model.add(Dense(1))
# Compile model
model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
# Fit model
model.fit(X, Y, epochs=20000, batch_size=64, verbose=2)
# Evaluate model
scores = model.evaluate(X, Y, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

# Save model descriptor
model_json = model.to_json()
with open("model_eca.json", "w") as json_file:
    json_file.write(model_json)
# Save model weights
model.save_weights("model_eca.h5")
```

Inference Code

```
import numpy as np
from matplotlib import pyplot
from keras.models import model_from_json

# Load created model
json_file = open('model_eca.json', 'r')
loaded_model_json = json_file.read()
```

```
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# Load weights
loaded_model.load_weights("model_eca.h5")
# Summarize model
loaded_model.summary()

n_mcs = 22
mcs = np.zeros(n_mcs)
x1 = np.zeros(n_mcs)
x2 = np.zeros(n_mcs)
x3 = np.zeros(n_mcs)
x4 = np.zeros(n_mcs)

for i in range(n_mcs):
    mcs[i] = i + 1
    x = np.array([mcs[i]/22,0.25], ndmin=2)
    x1[i] = 10**(-12*(loaded_model.predict([x])))
    x = np.array([mcs[i]/22,1.0], ndmin=2)
    x2[i] = 10**(-12*(loaded_model.predict([x])))
    x = np.array([mcs[i]/22,2.0], ndmin=2)
    x3[i] = 10**(-12*(loaded_model.predict([x])))
    x = np.array([mcs[i]/22,8.0], ndmin=2)
    x4[i] = 10**(-12*(loaded_model.predict([x])))

pyplot.figure(figsize=(10, 5))
pyplot.semilogy(mcs, x1,'o',markersize=8,color="red",label='ECA=0.25')
pyplot.semilogy(mcs, x2,'^',markersize=8,color="blue",label='ECA=1.0')
pyplot.semilogy(mcs, x3,'s',markersize=8,color="orange",label='ECA=2.0')
pyplot.semilogy(mcs, x4,'>',markersize=8,color="black",label='ECA=8.0')
pyplot.grid()
pyplot.xlim(0, 29)
pyplot.ylim(0.0001, 0.0000001)
pyplot.xlabel('MCS', fontsize=14)
pyplot.ylabel('BER (DNN output)', fontsize=16)
pyplot.xticks(np.arange(1,23,1), fontsize=14)
pyplot.yticks(fontsize=14)
pyplot.legend(fontsize=12, loc='upper right')
pyplot.savefig('eca_ber.pdf')
pyplot.show()
```

I.4.3 Low complexity DNN description

```
name: 'dense'
trainable: True
```

```
batch_input_shape: (None, 2)
dtype: float32
units: 4
activation: linear
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None}

[array([[ 0.41089684, -0.27577838, 0.20587917, -0.08847548],
    [ 1.5422895, -1.3661727, -4.703927, 0.20001416]], dtype=float32),
array([ 1.3518943, 1.1140927, -0.29445174, 0.00925397], dtype=float32)]

name: 'dense_1'
trainable: True
dtype: float32
units: 4
activation: sigmoid
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None}

[array([[ 0.50372106, 1.6832101, 4.371791, 0.517354],
    [ 0.9896978 , -1.958983, 3.5532384, 0.5536084],
    [ 0.01298787, -6.7618065, 0.16763994, -0.10537869],
    [-0.65475005,  0.01550872, -3.1136909, -0.01021753]], dtype=float32),
array([14.819842, -0.50683445, 3.8010275, 0.34193793], dtype=float32)]

name: 'dense_2'
trainable: True
dtype: float32
units: 4
activation: linear
use_bias: True
kernel_initializer: GlorotUniform
bias_initializer: Zeros
```

```
kernel_regularizer: None
bias_regularizer: None
activity_regularizer: None
kernel_constraint: None
bias_constraint: None}

[array([[ 1.5882529],
    [-2.1631515],
    [ 1.2733723],
    [-1.3134537]], dtype=float32),
array([0.99261254], dtype=float32)]
```

I.4.4 Indirect Decision with Achievement Probability

Learning Code

```python
import numpy as np
import random
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
import gym
from gym import Env
from gym.spaces import Discrete, Box, Dict, Tuple
from rl.agents import DQNAgent
from rl.policy import BoltzmannQPolicy
from rl.memory import SequentialMemory

class DQN_Env(Env):
    def __init__(self):
        # Actions: down, stay, up
        self.action_space = Discrete(3)

        # States: MCS, SINR, ECA
        self.mcs_min = 0
        self.mcs_max = 21
        self.sinr_min = 0
        self.sinr_max = 40
        self.eca_min = 0
        self.eca_max = 5
        self.observation_space = Box(low=np.array([self.mcs_min,
            self.sinr_min, self.eca_min]), high=np.array([self.mcs_max,
            self.sinr_max, self.eca_max]), shape=(3,))
        # Set states
        self.state = self.observation_space.sample()
        self.count = 10
        self.sinr_lim = [3.8, 4.6, 5.5, 6.2, 7.3, 7.8, 10.3, 11.3, 12.2,
                         13.2, 14.3, 14.9, 16.1, 16.8, 17.9, 18.9, 19.7,
                         20.4, 24.4, 25.3, 26.4, 27.1]
        self.window_sinr = 0.8
        self.window_eca = 0.2
        self.eca_target = 1.0
        self.lim_sinr_inf = self.sinr_lim[10] - self.window_sinr
        self.lim_sinr_sup = self.sinr_lim[10] + self.window_sinr

    def best_action(self, sinr, eca):
        if eca > (self.eca_target-self.window_eca) and
            eca < (self.eca_target + self.window_eca):
          return 1
        elif eca > (self.eca_target + self.window_eca):
          return 0
        elif sinr < self.lim_sinr_inf:
          return 0
        elif eca < (self.eca_target - self.window_eca) and
            sinr > self.lim_sinr_sup:
          return 2
        else:
          return 1

    def step(self, action):
        # Apply action
        # 0 = down
        # 1 = stay
        # 2 = up
        mcs = self.state[0]
        sinr = self.state[1]
        eca = self.state[2]
        self.lim_sinr_inf = self.sinr_lim[mcs] - self.window_sinr
        self.lim_sinr_sup = self.sinr_lim[mcs] + self.window_sinr

        self.state[0] += action - 1
        self.state[1] += (action - 1) + random.randint(-5,5)
        self.state[2] += (action - 1) + random.randint(-1,1)

        # Test the limits
        if self.state[0] < self.mcs_min:
            self.state[0] = self.mcs_min
        if self.state[0] > self.mcs_max:
```

```
    self.state[0] = self.mcs_max
if self.state[1] < self.sinr_min:
    self.state[1] = self.sinr_min
if self.state[1] > self.sinr_max:
    self.state[1] = self.sinr_max
if self.state[2] < self.eca_min:
    self.state[2] = self.eca_min
if self.state[2] > self.eca_max:
    self.state[2] = self.eca_max

self.count -= 1

# Calculate reward
best_act = self.best_action(sinr, eca)

if (action == 0) and (best_act == 0):
  reward = 1
elif (action == 2) and (best_act == 2):
  reward = 2
elif (action == 1) and (best_act == 1):
  reward = 1
else:
  reward = -2

if eca > (self.eca_target + self.window_eca) and action != 0:
  reward -= 5

if sinr < self.lim_sinr_inf and action != 0:
  reward -= 5

if eca > (self.eca_target - self.window_eca) and
    eca < (self.eca_target + self.window_eca):
  if action == 1:
    reward += 10

# Check if number of steps is done
if self.count <= 0:
    done = True
else:
    done = False

# Placeholder for info
info = {}
# Return step information
return self.state, reward, done, info
```

```
    def reset(self):
        self.state = np.array([10,20,0])
        self.count = 100
        return self.state

env = DQN_Env()
states = env.observation_space.shape
actions = env.action_space.n

def build_model(states, actions):
    model = keras.Sequential()
    model.add(Flatten(input_shape=(1,3)))
    model.add(Dense(12, activation='relu'))
    model.add(Dense(actions, activation='linear'))
    return model

model = build_model(states, actions)
model.summary()

def build_agent(model, actions):
    policy = BoltzmannQPolicy()
    memory = SequentialMemory(limit=100000, window_length=1)
    dqn = DQNAgent(model=model, memory=memory, policy=policy,
                   nb_actions=actions, nb_steps_warmup=1000,
                   target_model_update=1e-2)
    return dqn

dqn = build_agent(model, actions)
dqn.compile(Adam(learning_rate=0.002), metrics=['mae'])
dqn.fit(env, nb_steps=50000, visualize=False, verbose=1)

scores = dqn.test(env, nb_episodes=50, visualize=False)
print(np.mean(scores.history['episode_reward']))

dqn.save_weights('dqn_weights_la_1h_12.h5f', overwrite=True)

def softmax(x):
    ex = np.exp(x - np.max(x))
    return ex/ex.sum()

x = np.zeros((1,1,3))
for i in range(22):
    x[0][0][0] = i*1.0
    x[0][0][1] = env.sinr_lim[i]
```

```
x[0][0][2] = 1.0
y = 100*softmax(model.predict(x))
print('MCS', i+1, model.predict(x), y)
```