

AI models for digital signal processing in future 6G-IoT networks

Guillaume Larue

▶ **To cite this version:**

Guillaume Larue. AI models for digital signal processing in future 6G-IoT networks. Artificial Intelligence [cs.AI]. Institut Polytechnique de Paris, 2023. English. ⟨NNT : 2023IPPAT003⟩. ⟨tel-03990483⟩

HAL Id: tel-03990483

https://theses.hal.science/tel-03990483v1

Submitted on 15 Feb 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAT003

Thèse de doctorat

AI Models for Digital Signal Processing in Future 6G-IoT Networks

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat: Réseaux, Information et Communications

Thèse présentée et soutenue à Grenoble, le 10 Janvier 2023, par

GUILLAUME LARUE

Composition du Jury :

Didier LE RUYET Professeur, CNAM, Paris	Président
Catherine DOUILLARD Professeure, IMT Atlantique, Brest	Rapporteuse
Emmanuel BOUTILLON Professeur, UBS, Lorient	Rapporteur
Laurent SCHMALEN Professeur, KIT, Karlsruhe	Examinateur
Valérian MANNONI Ingénieur de recherche, CEA-Leti, Grenoble	Examinateur
Ghaya REKAYA BEN-OTHMAN Professeure, Télécom Paris, Paris	Directrice de thèse
Quentin LAMPIN Ingénieur de recherche, Orange Labs, Meylan	Co-encadrant
Louis-Adrien DUFRENE Ingénieur de recherche, Orange Labs, Meylan	Co-encadrant, invité
Hadi GHAUCH Maitre de Conférence, Télécom Paris, Paris	Co-encadrant, invité

Remerciements

A l'issue de ces travaux de thèse, je tiens à remercier l'ensemble des personnes qui m'ont accompagné tout au long de cette aventure :

J'aimerais tout d'abord remercier chaleureusement mes encadrants chez Orange, Quentin et Louis-Adrien, les principaux initiateurs de ces travaux et sans qui rien de tout cela n'aurait vu le jour. Le temps qu'ils m'ont consacré, les nombreuses discussions et leurs conseils avisés ont constitué un soutien précieux dans ce travail. Mais surtout, je tiens à leur témoigner de toute mon amitié, six ans après qu'ils m'aient invité à occuper le siège qui était alors vacant dans leur bureau. Je garde de très bons souvenirs des nombreux moments en leur compagnie qui s'en sont suivis.

Je remercie également ma directrice de thèse, Ghaya, et mon co-encadrant académique, Hadi, pour le suivi enthousiaste de ces travaux, hélas principalement à distance en raison des difficultés liées à la pandémie de Covid.

Je suis reconnaissant à l'égard de l'ensemble des membres du jury qui m'ont fait l'honneur d'assister à ma soutenance et d'évaluer mon travail avec attention. En particulier, j'adresse mes remerciements à Catherine Douillard et Emmanuel Boutillon pour avoir accepté d'être rapporteurs de cette thèse et pour le temps qu'ils ont consacré à la relecture de mon manuscrit. Je remercie Didier Le Ruyet, président du jury, Laurent Schmalen et Valérian Mannoni d'avoir accepté les rôles d'examinateurs.

Je tiens à remercier plus largement l'ensemble des collègues et amis rencontrés au fil de ces six années à Orange, pendant ma thèse ou l'alternance qui la précéda. Notamment, au sein de l'équipe CITY : Dominique, Eric, Esteban, Jean, Madhu, Marion, Mohamed T., Mohamed M., Nadège, Philippe, Stéphane... ; et plus largement sur le site de Meylan : Pauline, Paul, Philippe G., Eric P. ... Merci à vous tous, et à ceux que j'aurai eu la maladresse d'oublier de citer, pour ces moments de partage scientifique et de convivialité dont je garderai de très bons souvenirs.

Mes remerciements vont également à l'ensemble des amis qui m'ont accompagné de près ou de loin pendant cette thèse, et plus généralement tout au long de mes études. Les amis de longue date, du lycée, de l'IUT, d'école d'ingénieur ou d'ailleurs : Maëva, Gaëlle, Marion, Zoé, Clémence, Ousmane, Jérôme, Quentin, Victor, Julien, Stéphane, Mathieu... les citer dans leur intégralité serait trop long mais je suis certain que tous se reconnaitront.

Ces remerciements seraient loin d'être complets si j'oubliais d'évoquer ma famille. Parmi tant d'autres choses ils m'ont transmis le gout de la science, la curiosité et l'envie d'apprendre, qui m'ont amené jusqu'à entreprendre cette thèse. Sans leurs encouragements et leur affection je ne serais jamais parvenu à ce résultat. Merci Papa, Maman, Florent et Pierre d'être toujours présents pour moi. Merci à mes grands-parents pour ces moments inoubliables que l'on partage avec vous. Et à tous les autres membres de ma famille, cousins, tantes et oncles, pour tout le plaisir que j'ai à passer du temps en votre compagnie.

Enfin merci Elodie pour ton soutien de chaque instant et tous les beaux moments que l'on partage ensemble depuis bientôt 10 ans.

Résumé

Les technologies sans fil occupent une place importante dans les sociétés d'aujourd'hui. Par conséquent, les futurs réseaux de communication de $6^{\text{ème}}$ génération - au sujet desquels les premiers travaux de recherche ont récemment débuté - sont appelés à relever nombre de défis sociétaux et technologiques. On peut notamment citer la réduction de l'impact environnemental de nos sociétés et des technologies numériques, la souveraineté et la sécurité des infrastructures de télécommunications, ou encore la réduction de la fracture numérique. Plus spécifiquement, en ce qui concerne le premier point, là où les infrastructures de communication ont un impact environnemental croissant qu'il est essentiel de limiter, les technologies du numérique ont également un rôle à jouer dans la réduction de l'impact de tous les secteurs de l'économie. À cette fin, les réseaux du futur devront non seulement permettre un transfert plus efficace de l'information, mais aussi répondre aux besoins croissants en matière de capacité d'échange de données, favorisant ainsi l'optimisation des sociétés.

C'est notamment le rôle des cas d'usage de l'internet des objets, plus particulièrement de la ville ou de l'industrie intelligente, où l'utilisation d'un nombre massif de capteurs permet de superviser en temps réel des systèmes complexes. De toute évidence, ces cas d'usage sont associés à de nombreuses contraintes. Par exemple, un scénario typique de l'internet des objets implique un grand nombre de dispositifs connectés, disposant de ressources énergétiques restreintes mais supposées durer des années, avec un faible coût unitaire, une complexité, et donc des capacités de calcul, limitées, communicant dans un environnement de propagation complexe (faible couverture radio, nombreux interférents, etc.) et avec des exigences de latences raisonnables. Dès lors, une couche physique - c'est-à-dire l'ensemble des fonctionnalités permettant la transmission efficace et robuste d'informations entre deux dispositifs du réseau - à la fois performante et peu complexe est absolument cruciale.

L'utilisation de techniques d'intelligence artificielle est considérée comme une option pertinente pour la réalisation d'un tel objectif. D'une part, on considère que le cadre mathématique des réseaux de neurones, sur lesquelles de nombreuses solutions d'intelligence artificielle sont basées, permet des implémentations matérielles génériques, extrêmement efficaces et peu coûteuses. Ainsi, la description des fonctionnalités classiques de télécommunication et de traitement du signal sous forme d'architectures de réseaux de neurones est un moyen de bénéficier des avantages de ces technologies matérielles. D'autre part, et ceci est en partie permis par l'utilisation des réseaux de neurones, l'application de procédures d'apprentissage est un moyen d'améliorer les performances des fonctions de traitement du signal via la réduction des déficits algorithmiques et/ou de modélisation de certaines solutions conventionnelles.

Dans ces travaux nous nous intéressons à l'utilisation de techniques issues de l'intelligence artificielle, en particulier les réseaux de neurones et l'apprentissage machine, pour la réalisation d'opérations de traitement numérique du signal en couche physique dans le contexte des cas d'usage de l'internet des objets dans les réseaux 6G. Dans un premier temps, nous nous intéressons à la transcription d'algorithmes conventionnels issus de la littérature des communications numériques sous la forme de réseaux de neurones. Cette approche, premier pas vers l'intégration d'intelligence artificielle au sein de la couche physique, permet à elle seule de bénéficier des avantages des supports matériels pour l'exécution de réseaux de neurones. De plus, une telle approche n'exige aucune modification des spécifications techniques des réseaux existants. Cela permet ainsi l'intégration de telles solutions basées sur l'intelligence artificielle au sein de ces derniers. Sont, entre autres, proposées diverses structures d'égalisation, de démodulation et de décodage de codes correcteurs d'erreurs. Dans un second temps, nous nous intéressons à l'application de mécanismes d'apprentissage sur ces structures de réseaux de neurones dans le but d'en améliorer les performances vis-à-vis d'algorithmes conventionnels. Entre autres, un décodeur de codes linéaire en bloc générique est proposé permettant la découverte à l'aveugle d'un schéma de décodage avec des performances au moins équivalentes à celles du décodeur de référence. Pour finir, une structure de réseau de neurones de bout-en-bout de type auto-encodeur est proposée, permettant l'apprentissage conjoint d'un schéma de codage et du décodeur associé. Cette dernière contribution permet la découverte de schéma de codage pertinents pour les tailles réduites des paquets de l'internet des objets. Ce régime de taille est particulièrement contraignant et il n'existe pas à ce jour de schéma de codage/décodage qui soit à la fois optimal

d'un point de vue des performances et à complexité raisonnable. Le processus d'apprentissage et l'architecture proposés permettent la découverte de schémas de codage dont les performances sont comparables aux solutions état de l'art, construites via des approches conventionnelles et sur la base de nombreuses connaissances expertes.

Abstract

Wireless technologies are of paramount importance for today's societies. Therefore, future 6^{th} generation communication networks - whose initial research efforts have recently started - are expected to address a number of societal and technological challenges, including the environmental impact of our societies and of digital technologies, the sovereignty and security of telecommunication infrastructures or the reduction of the digital divide. More specifically, with regard to the first point, where communication infrastructures have a rising environmental impact which it is essential to reduce, digital technologies also have a role to play in reducing the impact of all sectors of the economy. To this end, the networks of the future must not only enable more efficient transfer of information, but also meet the growing need for data exchange capacity to enable informed decision-making for the optimisation of the societies.

This is notably the role of the use cases of the internet of things, more particularly of the smart city and smart industry, where the use of a massive number of sensors allows the real-time monitoring of complex systems. Obviously, these use cases are associated with a large number of constraints. A typical scenario for example involves a large number of connected devices, with limited energy resources but expected to last for years, with low unit cost, limited complexity and thus computational capabilities, in a complex propagation environment (low radio coverage, many interferers, etc.) and with low latency requirements. Therefore, a physical layer - i.e. the set of functionalities that allow the efficient and robust transmission of information between two network devices - that is both efficient and low-complexity is absolutely crucial.

The use of artificial intelligence techniques is considered as a relevant option for the fulfilment of such an objective. On the one hand, it is considered that the mathematical framework of neural networks, on which many so-called artificial intelligence solutions are based, allows extremely efficient, low-power and low-cost generic hardware implementations. Therefore, the description of classical telecommunication and signal processing functionalities as neural network architectures is a way to benefit from the advantages of these hardware technologies, even when the raw algorithmic performance is not expected to be improved. On the other hand, and this partly enabled by the use of neural networks, the application of learning procedures is a way of improving the performance of signal processing functions via the reduction of algorithmic and/or modelling deficits with respect to the conventional solutions.

In this work we are interested in the use of artificial intelligence techniques, in particular neural networks and machine learning, for the realisation of digital signal processing operations at the physical layer in the context of internet of things use cases of 6G networks. First, we are interested in the transcription of conventional algorithms from the digital communication literature into neural networks. This approach, which is a first step towards the integration of artificial intelligence within the physical layer, allows to benefit from the advantages of neural network dedicated hardware. Moreover, such an approach does not require any modification of the technical specifications of existing networks, thus guaranteeing the possible integration of such systems within these networks. Among other things, various structures for equalisation, demodulation and decoding of error-correcting codes are proposed. In a second step, we are interested in the application of learning mechanisms on these neural network structures in order to improve their performance with respect to the reference algorithms from the literature. In particular, a linear block code decoder is proposed which allows the blind discovery of a decoding scheme with performance at least equivalent to that of the reference decoder. Finally, an end-to-end auto-encoder structure is proposed, allowing the joint learning of an encoding scheme and the associated decoder. This last contribution allows the discovery of relevant coding schemes for the small packet sizes of the internet of things. This size regime is particularly challenging and no single code and decoder scheme exists to date that is both performance-optimal and of reasonable complexity. The learning process and the proposed architecture allow the discovery

of coding schemes whose performance is comparable to state-of-the-art schemes, which are built via conventional approaches, and rely on a lot of domain specific expertise.

CONTENTS

Remerciements i

Résumé iii

Abstract iv

List of Tables xi

List of Figures xiii

Acronyms xvii

Notations xxiii

1 Introduction 1
- 1.1 General Context - The Future 6G Networks . . . 1
- 1.2 Internet of Things in Future Networks . . . 2
- 1.3 Methodology and Approach . . . 4
- 1.4 Research Plan . . . 5
- 1.5 Contributions . . . 6
  - 1.5.1 Conference and Journal Publications . . . 6
  - 1.5.2 Patents . . . 6
  - 1.5.3 Open Source . . . 6
  - 1.5.4 Hexa-X Project . . . 6

2 Theoretical Foundations on Artificial Intelligence and Machine Learning 9
- 2.1 A Brief History of Computer Science and Artificial Intelligence . . . 10
- 2.2 Neural Networks and Related Computational Structures . . . 12
  - 2.2.1 A Simple Building Block: The Formal Neuron . . . 12
  - 2.2.2 Adding Width: Standard Neurons Layers . . . 13
  - 2.2.3 Increasing the Depth: Some Neural Networks Structures . . . 19
  - 2.2.4 Probabilistic Reasoning and Related Structures . . . 22
- 2.3 Neural Networks and Hardware Architectures . . . 33
- 2.4 Learning and Optimisation . . . 33
  - 2.4.1 The Main Classes of Learning Problems . . . 36
  - 2.4.2 Optimisation of Neural Networks . . . 37
- 2.5 Conclusion of the Chapter . . . 45

3 Artificial Intelligence at the Physical Layer **49**
3.1 A Brief Introduction to Digital Communication Systems and Physical Layer . . . 50
3.1.1 General Role and Architecture of the Physical Layer in a Digital Communication System . . . 50
3.1.2 Source Coding . . . 51
3.1.3 Channel Coding . . . 53
3.1.4 Line Encoding and Modulation . . . 55
3.1.5 Transmission Medium . . . 57
3.1.6 Detection, Synchronisation and Equalisation . . . 59
3.1.7 Demodulation and Decoding . . . 61
3.1.8 Specific Considerations in the Present Study . . . 63
3.2 Artificial Intelligence at the Physical Layer - A Survey . . . 63
3.2.1 An Historical Perspective on Artificial Intelligence at the Physical Layer . 63
3.2.2 Approaches to the Integration of Artificial Intelligence at the Physical Layer 64
3.2.3 Expected Gain and Opportunities . . . 69
3.2.4 Challenges . . . 71
3.3 Conclusion of the Chapter . . . 72

4 Describing Signal Processing Operations Using Neural Networks **73**
4.1 Equalisation . . . 74
4.1.1 Base-Band Equivalent Tap Channel Model . . . 74
4.1.2 Linear Equalisation . . . 77
4.1.3 Single-Path Equalisation using Neural Networks . . . 79
4.1.4 Extension to Multi-Path Equalisation . . . 83
4.1.5 Perspectives . . . 84
4.2 Designing a Neural Network Based Detection System . . . 84
4.2.1 QAM Demodulator - Naive approach . . . 85
4.2.2 Low Complexity Neural Network Based QAM Demodulator . . . 88
4.3 Experimentation on a Software Defined Radio Test-Bed . . . 97
4.3.1 System Model and Working Assumptions . . . 98
4.3.2 Proposed Neural Network Based Receiver and Experimental Performance 99
4.4 Conclusion of the Chapter . . . 102

5 Learning Process at the Air Interface: Application to Channel Coding **105**
5.1 Motivations . . . 107
5.2 An Introduction to Iterative Methods for the Decoding of Linear Block Codes . . 108
5.2.1 Fundamentals of Linear Block Codes . . . 109
5.2.2 A Step Towards Soft Iterative Decoding . . . 111
5.2.3 Factor Graph Representation of a Linear Block Code . . . 111
5.2.4 Applying Message Passing Algorithm to the Factor Graph of a Code . . . 112
5.2.5 Degree Distribution Notations . . . 117
5.3 Belief Propagation Decoder as an Efficient Recurrent Neural Network for Short Codes . . . 117
5.3.1 Description of the Neural Belief Propagation Algorithm and its Variants . 117
5.3.2 Neural Belief Propagation as an Efficient Gated Recurrent Neural Network 118
5.4 Learning to Decode - Blind Neural Belief Propagation Decoder . . . 126
5.4.1 System model . . . 126
5.4.2 An Auto-Encoder Structure to Learn to Decode . . . 126
5.4.3 Model Training and Results . . . 128
5.5 Learning to Code - Neural Belief Propagation Auto-Encoder for Linear Block Code Design . . . 130
5.5.1 Learning to Code: Main Challenges . . . 130
5.5.2 Linear Block Code Neural Belief Propagation Auto-Encoder . . . 130
5.5.3 Data-sets, Training and Evaluation Procedures . . . 134
5.5.4 Results . . . 135
5.5.5 Discussion and Perspectives . . . 145

5.5.6 Source Code 146
5.6 Conclusion of the Chapter 147

6 Conclusion **149**
6.1 Reminder on the Context of this Work 149
6.2 Results 149
6.3 Perspectives 150

Appendices **151**
A Proofs 151
A.I Sum of Random Binary Variables 151
B Decoding Complexity 152
B.I Standard Belief Propagation Decoders 152
B.II Neural Belief Propagation Decoders 153
B.III Gated Neural Belief Propagation Decoders 153
B.IV Parallel Belief Propagation Decoders 154
C BER and BLER Performance 155

Bibliography **157**

LIST OF TABLES

0.1 Numbers and Arrays xxiii
0.2 Random Numbers and Probabilities xxiii
0.3 Indexing and Linear Algebra Operations[1] xxiv
0.4 Calculus xxiv
0.5 Functions & Signals xxv
0.6 Sets xxv
0.7 Other Conventions & Notations xxv

1.1 Overview of NR-RedCap Reference Use-Cases and Requirements [4] 3
1.2 LTE-M and NB-IoT capabilities with respect to NR-RedCap requirements [4] . . 3

2.1 Common Non-Linear Activation Functions 21

3.1 Correction Table of the $\mathcal{C}(3,1)$ Repetition Code 55

4.1 Minimal Neural Network Model Architecture Proposed for M-QAM Demodulation 91
4.2 Complexity Comparison for Different Demodulation Schemes 91

5.1 Sum-Product Rules for Equality and Zero-Sum Factors of a Tanner Graph 115
5.2 Complexity - Variable Nodes to Check Nodes 120
5.3 Complexity - Check Nodes to Variable Nodes 123
5.4 Complexity - Output Marginalisation Operation 124
5.5 Number of Trainable Parameters of the Proposed Decoder 128
5.6 Auto-Encoder Number of Parameters 134
5.7 "The Curse of Code Dimensionality" 141
5.8 Estimated Number of Operations for Each Code and Decoder 143
5.9 Degree Distribution of the Codes 144

C..1 BER and BLER Performance 155

LIST OF FIGURES

1.1 AI Models for Digital Signal Processing at the Physical Layer in a 6G IoT Context 5

2.1 A Simplified Hierarchy of AI Related Concepts 11
2.2 McCulloch and Pitts Formal Neuron Model 12
2.3 Modern Definition of a Perceptron 13
2.4 Compact Representation of a Neuron 13
2.5 Standard Definition of Neuron Layer 14
2.6 Cross-Correlation of Two Signals x and y 15
2.7 1D Convolutional Layer 16
2.8 Example of a 2D Convolutional Layer 16
2.9 Compact Representation of a Standard Recurrent Layer 17
2.10 Unfolded Structure of a Standard Recurrent Layer 18
2.11 An Example of Gated Cell: the Long Short Term Memory Cell 18
2.12 Various Recurrent Layer Inputs/Outputs Topology 19
2.13 The Multi-Layer Perceptron 20
2.14 Log-Likelihood Functions of a Bernouilli Model for Sequences of Coin Flips . . . 26
2.15 Bayesian Posterior Distribution Computation 27
2.16 An Example of Factor Graph 28
2.17 An Example of Factor Graph with Equality Constraint Node 29
2.18 Marginalization on a Factor Graph 30
2.19 Example of a joint probability distribution represented as a Bayesian directed acyclic graph 32
2.20 Equivalent Factor Graph to the Bayesian newtork of Figure 2.19 33
2.21 One Possible HW Implementation of (2,n) by (n,2) Matrix Multiplication Unit . 34
2.22 One Possible Implementation of a Dense Neural Network Layer as a Systolic Array 35
2.23 The Bias-Variance Trade-off 39
2.24 Cross-Validation Method 39
2.25 Gradient Descent 41
2.26 Some Challenges of the Gradient Descent - Critical Points & Initial Conditions . 42
2.27 Computational Graph of a Two-Layer Neural Network 44
2.28 Chain Rule of Derivatives & Back-Propagation of Gradients 45
2.29 Complete Optimisation Graph of a Two-Layer Neural Network 47

3.1 Simplified Digital Communication System 51
3.2 Entropy of a Binary Source 52
3.3 Huffman Entropic Coding 52
3.4 Illustration of the $\mathcal{C}(3,1)$ Repetition Code 54
3.5 Simplified Line Coding and Modulation Chain in a Digital Communication System 55
3.6 Quadrature Modulation - Example of the 4-QAM Modulation 57

3.7 Characteristics of an Arbitrary Multi-path Channel 58
3.8 Typical Frame Structure for Synchronization and Block Equalization 60
3.9 Hard versus Soft Demodulation - BPSK under AWGN 62
3.10 5G NR PUSCH Chain 65
3.11 A Simple Typology of AI for PHY 68
3.12 Benchmarking Neural Network Models on Different Hardware Platforms 70

4.1 Tap Channel Models 75
4.2 Digital Base-Band Equivalent System Model 75
4.3 Schematic of a Linear Equaliser Represented Using a Transversal Filter 76
4.4 Noise Enhancement Problem in Zero-Forcing Equalisers 78
4.5 Proposed CNN Architecture for Single-Path Equalisation 81
4.6 Single Path Equalisation Using Convolutional Linear Neural Network Layer . . . 82
4.7 Multi-Path Equalisation Using Linear Convolutional Neural Network Layer . . . 83
4.8 Performance of the proposed CNN-based Multi-Path Equaliser over a 3-Tap Channel 84
4.9 Constellation of a 16th Order Gray Mapped Quadrature Amplitude Modulation . 85
4.10 The QAM Demodulation - Neural Network Decision Problem Perspective 86
4.11 Naive Neural Network Based QAM Demodulator 87
4.12 Optimal 4-QAM Decision Boundaries 88
4.13 Learned M-QAM Demodulators 89
4.14 Proposed CNN Model for 16-QAM Demodulation 90
4.15 Hard Demodulation of 16-QAM Using the Proposed CNN Model 92
4.16 Hard Demodulation of 256-QAM Using the Proposed CNN Model 93
4.17 Comparison of the BER Demodulation Performance 94
4.18 Soft Demodulation of a BPSK Under AWGN Channel 95
4.19 Single Channel 16-QAM Soft Demodulation under AWGN 96
4.20 LLR function for a 16-QAM Modulation under AWGN 97
4.21 Baseband Description of the Considered M-QAM Communication Chain 98
4.22 Functional Diagram of the Test Bench Implementation 98
4.23 Ettus USRP B210 99
4.24 Spirent Vertex Channel Emulator 99
4.25 Description of the Proposed Processing Chain 99
4.26 Comparison of the Performance of a Regular Receiver and the Proposed CNN-Based Receiver over an Emulated AWGN Channel 100

5.1 Representation of the Three Parity Check Equations of Hamming (7,4) Code . . 110
5.2 Iterative decoding of three erasures with Hamming (7,4) codes. 111
5.3 Tanner Graph of the Hamming (7,4) Code. 112
5.4 Generic 3×3 Factor Graph 119
5.5 Naive Implementation of Eq. (5.43) using a Matrix Multiplication 120
5.6 Efficient Computational Graph for Eq. (5.43) 121
5.7 Naive Implementation of Eq. (5.45) 122
5.8 Efficient Computational Graph for Eq. (5.45) 123
5.9 Naive Implementation of Eq. (5.44) using a Matrix Multiplication 124
5.10 Efficient Computational Graph for Eq. (5.44) 124
5.11 Gated Neural Belief Propagation Recurrent Neural network Cell 125
5.12 System model of the considered communication scheme. 126
5.13 Proposed Auto-Encoder Architecture 127
5.14 Block Error Rate of the Blind Gated Neural Belief Propagation Decoder 129
5.15 Proposed Linear Block Code Neural Encoder Model 131
5.16 Proposed Auto-Encoder Architecture for Linear Block Code Design 133
5.17 Performance Comparison of the Auto-Encoder with that of a Hamming (8,4) Code 136
5.18 (8,4) PC Matrices and their Associated Tanner Graph Representation 136
5.19 BER Evolution During a Training of the Auto-Encoder Model 137
5.20 Repeatability of the Auto-Encoder Training 138

5.21 Correlogram of the Performance of the 50 Learned Codes Evaluated on GNBP and BP Decoders. 139
5.22 Correlogram of the Performance of the 50 Learned Codes Evaluated on GNBP and MLD Decoders. 139
5.23 Correlogram of the Performance of the 50 Learned Codes Evaluated on BP and MLD Decoders. 139
5.24 Variable Node Degree Distribution Before and After Training 139
5.25 Check Node Degree Distribution Before and After Training 139
5.26 Study of the Joint Encoder/Decoder Training Procedure 140
5.27 Performance Comparison Between (63,36) BCH and AE-Based Codes with Various Decoders. 141
5.28 Performance Comparison Between (63,45) BCH and AE-Based Codes with Various Decoders. 141
5.29 Performance to Complexity Ratios of the Different (63,36) Codes and Decoders . 142
5.30 Performance to Complexity Ratios of the Different (63,45) Codes and Decoders . 142
5.31 Performance Comparison Between Various (64,k) LDPC Codes and The Proposed AE 143
5.32 Comparison of Different (128,64) Codes and their Respective Decoders from [170, 108] 145

B.I.1 Equation (B.I.1) - Variable to Check - Example of Degree 3 Variable Node. . . . 152
B.I.2 Equation (B.I.2) - Check to Variable - Example of Degree 3 Check Node. 152
B.I.3 Equation (B.I.3) - Marginalisation - Example of Degree 3 Variable Node. 152

ACRONYMS

AE	*Auto-Encoder*
ADC	*Analogue to Digital Converter*
AI	*Artificial Intelligence*
ALU	*Algorithmic Logic Unit*
ANN	*Artificial Neural Network*
ARQ	*Automatic Repeat Request*
ASIC	*Application Specific Integrated Circuit*
ASP	*Adaptive Signal Processing*
AWGN	*Additive White Gaussian Noise*
BCE	*Binary Cross-Entropy*
BCH	*Bose, Ray-Chaudhuri et Hocquenghem*
BER	*Bit Error Rate*
BLER	*Block Error Rate*
BPROP	*Back-Propagation*
BPSK	*Binary Phase Shift Keying*
BS	*Base Station*
BSC	*Binary Symmetric Channel*
CAZAC	*Constant Amplitude Zero Auto-Correlation*
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
CP-OFDM	*Cyclic Prefixed Orthogonal Frequency Division Multiplexing*
CRC	*Cyclic Redundancy Check*
CSI	*Channel State Information*
CV	*Computer Vision*
dB	*Decibel*
DFE	*Direct Feedback Equaliser*
DL	*Deep Learning*

DMRS *Demodulation Reference Signal*

DNN *Deep Neural Network*

DSF *Differentiable Step Function*

DSP *Digital Signal Processor.ing*

ECC *Error Correction Code*

eMBB *enhanced Mobile Broad-Band*

ERM *Empirical Risk Minimisation*

E2E *End-to-End*

FEC *Forward Error Correction*

FG *Factor Graph*

FPGA *Field Programmable Gate Array*

FLOPS *Floating Point Operations Per Second*

GA *Genetic Algorithm*

GD *Gradient Descent*

GNBP *Gated Neural Belief Propagation*

GPU *Graphics Processing Unit*

GRU *Gated Recurrent Unit*

HARQ *Hybrid Automatic Repeat Request*

HDPC *High Density Parity-Check*

HW *Hardware*

IC *Integrated Circuit*

i.i.d. *independently and identically distributed*

IoT *Internet of Things*

IQ *In-phase/Quadrature*

ISI *Inter-Symbol Interference*

JSCC *Joint Source-Channel Coding*

k-NN *k-Nearest Neighbours*

LBC *Linear Block Codes*

LDPC *Low Density Parity Check*

LMS *Least-Mean Square*

LO *Local Oscillator*

LoS *Line of Sight*

LPWAN *Low Power Wide Area Network*

LR *Learning Rate*

LS *Least Square*

LSB *Least Significant Bit*

LSTM *Long-Short Term Memory*

LTE *Long-Term Evolution*

LTE-M *Long-Term Evolution for Machines*

LTI *Linear Time Invariant*

LUT *Look-Up Table*

MAB *Multi-Armed Bandit*

MAC *Medium Access Control*

MAPE *Maximum A Posteriori Estimator*

MDS *Maximum Distance Separable*

MEC *Mobile Edge Computing*

MIMO *Multiple-Inputs Multiple-Outputs*

ML *Machine Learning*

MLD *Maximum Likelihood Decoding*

MLE *Maximum Likelihood Estimator*

MLP *Multi-Layer Perceptron*

MLSE *Maximum Likelihood Sequence Estimator*

MMSE *Minimum Mean Square Error*

MMU *Matrix Multiply Unit*

MPA *Message Passing Algorithm*

M-QAM *M-Ary Quadrature Amplitude Modulation*

mRRD *modified Random Redundant Decoder*

MS *Min-Sum*

MSE *Mean Squared Error*

MSB *Most Significant Bit*

M-AC *Multiply-Accumulate*

NBP *Neural Belief Propagation*

NB-IoT *Narrowband Internet of Things*

NLP *Natural Language Processing*

NN *Neural Network*

NOMS *Neural Offset Min-Sum*

NPU *Neural processing Unit*

OFDM *Orthogonal Frequency Division Multiplexing*

OMS *Offset Min-Sum*

OSD *Ordered Statistic Decoding*

PC *Parity-Check*

PCA *Principal Component Analysis*

PD *Positive Definite*

PDF *Probability Density Function*

PE *Processing Element*

PEG *Progressive Edge Growth*

PHY *Physical Layer*

PMF *Probability Mass Function*

PUSCH *Physical Uplink Shared Channel*

QoS *Quality of Service*

QPSK *Quadrature Phase Shift Keying*

QAM *Quadrature Amplitude Modulation*

RAN *Radio Access Network*

RE *Resource Element*

ReLU *Rectified Linear Unit*

RF *Radio Frequencies*

RL *Reinforcement Learning*

RNN *Recurrent Neural Network*

RRC *Radio Resource Control*

SCMA *Sparse Code Multiple Access*

SDR *Software Defined Radio*

SGD *Stochastic Gradient Descent*

SNR *Signal to Noise Ratio*

SOM *Self Organising Map*

SoC *System on Chip*

SOTA *State of the Art*

SPA *Sum-Product Algorithm*

TB-CC *Tail Biting Convolutional Code*

TCO *Total Cost of Ownership*

TG *Tanner Graph*

TPU *Tensor Processing Unit*

TS *Technical Specification*

***t*-SNE**	*t-distributed Stochastic Neighbour Embedding*
UE	*User Equipment*
URLLC	*Ultra-Reliable Low-Latency Communications*
Wi-Fi	*Wireless Fidelity*
WP	*Work Package*
XOR	*eXclusive-OR*
ZC	*Zadoff-Chu*
ZF	*Zero-Forcing*
4G	4^{th} *Generation*
5G	5^{th} *Generation*
5G-NR	*5G New Radio*
6G	6^{th} *Generation*

NOTATIONS

Throughout the present manuscript and unless otherwise specified, the following notations are employed[2]:

Table 0.1: Numbers and Arrays

a	Scalar.
$\boldsymbol{a}$	Vector (column).
$\boldsymbol{A}$	Matrix.
$\mathbf{e}^{(k)}$	k^{th} standard basis vector, *i.e.* a vector with a 1 at index k and 0 otherwise.
I_n	Identity matrix of size $n \times n$. If n is not specified, it is implied by context.
$\mathbf{0}$	All-zero vector (or any n-dimensional tensor) whose size is implied by context. More generally, bold face numbers denote a vector with a constant value across all indices.
j	Imaginary unit.
x^*	Complex conjugate of the complex number x.

Table 0.2: Random Numbers and Probabilities

a	Random scalar variable.
$\mathbf{a}$	Random vector variable (column).
$\mathbf{A}$	Random matrix variable.
$\mathrm{a} \sim D$	Random variable a follows distribution D.
$\mathbb{P}_{\mathrm{a}\sim D}\{\mathrm{a} = a\}$	Probability that the random variable a takes the value a under distribution D.
$p_{\mathrm{a}\sim D}(\mathrm{a} = a) = p_{\mathrm{a}}(a)$	Probability Mass Function of (discrete) random variable a.
$f_{\mathrm{a}\sim D}(\mathrm{a} = a) = f_{\mathrm{a}}(a)$	Probability Density Function of (continuous) random variable a.
$\mathbb{E}_{\mathrm{a}\sim D}\{\mathrm{a}\}$	Expectation of the random variable a following distribution D.
$\mathrm{Var}_{\mathrm{a}\sim D}\{\mathrm{a}\}$	Variance of the random variable a following distribution D.

[2]Most notations are borrowed from [1].

[3]Denominator layout convention is employed for matrix and vector manipulations [2]. The indexing of any array start at 1. When an operator involve two arrays whose shape does not match across one or several dimensions, a broadcasting operation is usually considered (*e.g.* $\boldsymbol{X} + y$ is equivalent to add the scalar y to all elements of $\boldsymbol{X}$).

Table 0.3: Indexing and Linear Algebra Operations[3]

a_i	i-th element of vector $\boldsymbol{a}$.
a_i	i-th element of the random vector $\mathbf{a}$.
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$.
$\mathrm{A}_{i,j}$	Element i, j of the random matrix $\mathbf{A}$.
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$.
$\boldsymbol{A}_{:,j}$	Column j of matrix $\boldsymbol{A}$.
$\boldsymbol{a}^T$	Transpose of vector $\boldsymbol{a}$.
$\boldsymbol{A}^T$	Transpose of matrix $\boldsymbol{A}$.
$\boldsymbol{A}^{-1}$	Inverse of matrix $\boldsymbol{A}$.
$\lvert\boldsymbol{A}\rvert$	Determinant of matrix $\boldsymbol{A}$.
$\boldsymbol{A}^\dagger$	Transpose conjugate of $\boldsymbol{A}$, or *Hermitian* transpose.
$\boldsymbol{A}^*$	Conjugate of $\boldsymbol{A}$.
$\mathrm{diag}(\boldsymbol{v})$	Diagonal matrix whose diagonal elements are the vector $\boldsymbol{v}$.
$\mathrm{D}(\boldsymbol{M})$	Vector (column) constructed from the diagonal elements of matrix $\boldsymbol{M}$.
$\boldsymbol{A} \times \boldsymbol{B}$	Product of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.
$\boldsymbol{A} \odot \boldsymbol{B}$	Point wise (Hadamard) product of matrices (or any n-dimensional tensor) $\boldsymbol{A}$ and $\boldsymbol{B}$.
$\lvert\lvert\boldsymbol{a}\rvert\rvert_1 = \sum \lvert a_i \rvert$	1-norm
$\lvert\lvert\boldsymbol{a}\rvert\rvert_2 = \sqrt{\sum a_i^2}$	2-norm, or *Euclidean* norm
$\lvert\lvert\boldsymbol{a}\rvert\rvert_2^2 = \sum a_i^2$	Squared Euclidean norm
$\boldsymbol{A} \succ \mathbf{0}$, $\boldsymbol{A} \succcurlyeq \mathbf{0}$	Positive Definite and Positive Semi-Definite matrices.
$\boldsymbol{A} \prec \mathbf{0}$, $\boldsymbol{A} \preccurlyeq \mathbf{0}$	Negative Definite and Negative Semi-Definite matrices.

Table 0.4: Calculus

$\frac{df(x)}{dx} \in \mathbb{R}$ or simply $f'(x)$	Derivative of $f(x) : \mathbb{R} \to \mathbb{R}$ with respect to scalar x.
$\frac{\partial f(x)}{\partial x} \in \mathbb{R}$	Partial derivative of scalar field $f(x) : \mathbb{R}^n \to \mathbb{R}$ with respect to scalar x.
$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \in \mathbb{R}^n$	Gradient of scalar field $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ with respect to vector $\boldsymbol{x}$.
$\nabla_{\boldsymbol{X}} f(\boldsymbol{X}) = \frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}} \in \mathbb{R}^{n,m}$	Matrix derivative of scalar field $f(\boldsymbol{X}) : \mathbb{R}^{n,m} \to \mathbb{R}$ with respect to matrix $\boldsymbol{X}$.
$\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x} \in \mathbb{R}^m$	Derivative of vector field $\boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$ with respect to scalar x.
$\boldsymbol{J}_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}) = \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} \in \mathbb{R}^{n,m}$	Jacobian matrix of vector field $\boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$ with respect to vector $\boldsymbol{x}$.
$\frac{\partial \boldsymbol{F}(\boldsymbol{X})}{\partial x} \in \mathbb{R}^{l,p}$	Derivative of matrix function $\boldsymbol{F}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^{l,p}$ with respect to scalar x.
$\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \in \mathbb{R}^{n,n}$	Hessian of scalar field $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ with respect to vector $\boldsymbol{x}$.

Table 0.5: Functions & Signals

$x(t)$	A continuous signal as a whole or indexed at time t.
$x[k]$	A discrete signal as a whole or the k-th sample (which is sometime denoted as x_k using vector indexing notations).
$\delta(x)$	Dirac function or Kronecker Delta function.
$f(x;\theta)$	Parametric function of x wiht θ as a parameter.
$\underset{\boldsymbol{\theta}}{\text{argmin}} f(\boldsymbol{\theta})$	Argument of the minima of the function $f(\boldsymbol{\theta})$ with regard to parameters $\boldsymbol{\theta}$.
$\underset{\boldsymbol{\theta}}{\text{argmax}} f(\boldsymbol{\theta})$	Argument of the maxima of the function $f(\boldsymbol{\theta})$ with regard to parameters $\boldsymbol{\theta}$.
$\underset{x \longrightarrow \infty}{\text{Lim}}\, f(x)$	Limit of the function $f(x)$ as x approaches $+\infty$.
$x * y$	Convolution of signal x by signal y (discrete or continuous).
$x \star y$	Cross-correlation of signal x by signal y (discrete or continuous).
$\boldsymbol{R_{yy}}$	Auto-Correlation Matrix of signal $\boldsymbol{y}$.

Table 0.6: Sets

$\mathbb{S}$	A set.
$\{1, 2, 3\}$	The elements of a set.
$\mathbb{S}^n$	The power set whose element are the n-ary cartesian product of $\mathbb{S}$.
$\mathbb{S}^c$	Complement of a set (with regard to another).
$\mathbb{A} \bigcup \mathbb{B}$	Union of sets $\mathbb{A}$ and $\mathbb{B}$.
$\mathbb{A} \bigcap \mathbb{B}$	Intersection of sets $\mathbb{A}$ and $\mathbb{B}$.
Card($\mathbb{S}$)	The cardinal, *i.e* size, of a set.
$[a, b[\in \mathbb{R}$	An interval over an ordered set between a (included) and b (excluded) endpoints.
$\mathbb{F}_p$	A finite fied, *i.e.* Gallois field, of p element.
$\mathbb{R}$	The set of real numbers.
$\mathbb{C}$	The set of complex numbers.

Table 0.7: Other Conventions & Notations

$x_i^{<t>}$	i-th feature of time step t (RNN).
$y_i^{(n)}$	i-th output of n-th layer (MLP).
$\hat{x}$	The estimated value of x.
$x^\star$	The optimal value of x.
$\tilde{x}$	Approximately x.

CHAPTER 1

INTRODUCTION

1.1 General Context - The Future 6G Networks

After a rapid evolution over the last 50 years, wireless technologies are increasingly present in our societies and their importance is expected to grow in the future, fostering the emergence of new services and ecosystems. While the 5^{th} *Generation* (5G) of mobile network is rolling out, the research on future 6^{th} *Generation* (6G) networks have recently started worldwide. Among the different research projects initiated worldwide, this Ph.D. work takes place in the context of Hexa-X, the European Union H2020 flagship project on 6G networks. Hexa-X project aims to initiate research on the next generation of mobile networks in Europe, and imagine tomorrow's technological tools to interconnect a *"digital world of data, a physical world of process and human world of intelligence and values"* [3]. In the context of this European project, the early reflections identified societal needs and the associated major challenges that are to be addressed by the future generations of 2030s networks, and lay their foundations [3]:

1 - Connecting Intelligence: The next generation of networks should, on the one hand, rely on increased intelligence to allow for a more efficient transfer of information and, on the other hand, offer society as a whole the means to implement intelligent processes on a large scale.

2 - Network of Networks: 6G will need to support heterogeneous communication and computing resources that interact and connect together in a global digital ecosystem. Such an ecosystem will need to integrate multiple scales, communicating seamlessly as a network of intelligent networks. Such a complex entity will need to enable cost and energy efficient mass deployment to sustainably support all segments of society.

3 - Sustainability: With the digital world having an increasingly significant environmental impact, the footprint of future telecommunication infrastructures is at the heart of the definition of 6G. The latter will need to fully improve the end-to-end resource utilisation of the cellular network and significantly improve its energy efficiency. In addition, 6G networks are expected to play an important role in improving the sustainability of all parts of the economy by providing the digital tools to meet this objective.

4 - Global Service Coverage: While 6G networks need to integrate heterogeneous network technologies into a network of networks, they also need to provide coverage on a global scale. The next generation of networks should provide efficient and affordable digital access to connect remote and vulnerable people and address the digital divide, as well as facilitate business and transport activities by providing service coverage over very large geographical areas.

5 - Extreme Experience: 6G is expected to offer, among other things, increased speeds and capacity, reduced latency and more accurate location and sensing services compared to 5G networks.

6 - Trustworthiness: In view of the ever-increasing role of digital ecosystems in our societies and lives, 6G networks should be highly secure and resilient infrastructures, supporting the confidentiality and privacy of data communications. A trusted network is essential for the development of secure services in the democratic and sovereign societies of the EU.

To address all these challenges, and in particular the first one, *Artificial Intelligence* (AI) and *Machine Learning* (ML) technologies are seen as key technologies and are therefore expected to play a major role. While AI was already present in some parts of the 5G network, the extent of its use is expected to reach a whole new dimension in 6G networks, where AI functions will be natively supported in all parts of the system. In particular, while the design of AI-driven communications is expected to improve the performance of the network itself, *e.g.* by increasing the spectral efficiency at the *Physical Layer* (PHY), it is also expected from the 6G network to support and enhance reliable AI/ML technologies at scale to provide the wider society with an access to interconnected, sustainable and trustworthy intelligence. In other words, AI-driven communication is expected to improve the network, and the latter should facilitate the use of AI by the society.

1.2 Internet of Things in Future Networks

Among the many use cases for 6G networks, and extending those of previous network generations, 6G is envisioned to address the expansion of *Internet of Things* (IoT) use-cases (*e.g.* sensor networks, smart cities and industries) whose traffic and *Quality of Service* (QoS) requirements are deemed to grow significantly over the next decades. Broadly speaking, these scenarios aim to collect large amounts of data from a very large number of connected devices, enabling informed decision-making and thus improving the overall efficiency of complex systems, such as cities. IoT use-cases usually come with particularly stringent requirements and constraints, such as energy efficiency (especially in the case of battery-powered devices), reduced *Hardware* (HW) cost and complexity, and the subsequent reduction in computing capacity. These constraint typically imply the use of short packets and low-complexity communication schemes. Moreover, the expected number of devices leads to very high connection density, often with bad coverage conditions (*e.g.* deep indoor scenarios, wide area coverage, etc.). In this context an efficient and low complexity PHY layer, ensuring the efficient and robust transmission of data over challenging physical radio channels, is a key requirement.

An example of these constraints is provided in Table 1.1. This table lists some reference use-cases and requirements of NR-RedCap[1] from [4]. These balanced IoT scenarios exhibit multiple contraints such as reasonable latencies, relatively high data-rates and perharps more importantly, a controled energy consumption and a cost well below standard 5G-NR solutions (and thus a reduced complexity). Furthermore, the Table 1.2 compares these requirements to the capabilities of conventional *Low Power Wide Area Network* (LPWAN) technologies such as NB-IoT and LTE-M and shows a clear emphasis on achieving significantly higher performance within a controlled complexity envelope.

Again, AI and ML seem very promising tools to improve the capabilities of IoT cellular networks and particularly to address these two conflicting goals of increased performance and reduced complexity [5, 6]. These techniques might allow to improve the spectral efficiency, to jointly optimise the *End-to-End* (E2E) communication chain, to reduce the control and signalling overhead or even to correct complex hardware impairments. The main identified benefits of using AI and ML at the PHY layer of IoT systems are related to:

Reduction of model deficits - Performance improvements: When the phenomena encountered become too complex to be adequately modelled or the problems to be solved

[1] NR-RedCap is a lightweight version of *5G New Radio* (5G-NR) proposed for balanced and mixed IoT use cases that are neither supported by 5G-NR, *Long-Term Evolution for Machines* (LTE-M) or *Narrowband Internet of Things* (NB-IoT).

	Industrial IoT	Wearables	Video Surveillance
Communication service availability	99.99%	From 99% to 99.99%	From 99% to 99.9%
Latency	$<$ 100 ms, $<$ 10 ms for augmented reality and safety-related sensors	$<$ 10 ms	$<$ 500 ms
Reference bit rate	$<$ 2 Mb/s	10–50/min. 5 Mb/s in Down-Link/Up-Link; Peak 150/50 Mb/s in Down-Link/Up-Link	In Up-Link 2–4 Mb/s and 7.5–25 Mb/s for high-end video
Device battery life	$\geq$ 5 years	1 to 2 weeks	Not Applicable

Table 1.1: Overview of NR-RedCap Reference Use-Cases and Requirements [4]

	LTE-M cat. M2 (Rel. 15)	NB-IoT (Rel. 15)	NR-RedCap
Bandwidth	5 MHz	180 kHz	Minimum 10 MHz
Peak data rates	Down-Link/Up-Link: 4 Mb/s /$\sim$7 Mb/s (FD-FDD)	Down-Link/Up-Link: $\sim$127 kb/s / $\sim$143 kb/s (HD-FDD)	Down-Link/Up-Link: 150 Mb/s; 50 Mb/s for smart wearables
Latency	$<$ 10 s	$<$ 10 s	$<$ 100 ms; 5–10 ms for safety related sensors
Reliability	$<$ 99%	$<$ 99%	99–99.99%
Device battery life	$>$ 10 years	$>$ 10 years	$>$ 5 years for IIoT sensors; 1–2 weeks for smart wearables

Table 1.2: LTE-M and NB-IoT capabilities with respect to NR-RedCap requirements [4]

are difficult to define in a formal way, the use of *Neural Network* (NN) and data driven approaches can sometimes overcome these limitations and process the signal in a more precise and refined way, therefore leading to performance gain over standard models.

Reduction of algorithmic deficit - Complexity reduction: An exact or satisfactory model is sometimes known but it's algorithmic solution is intractable given a set of operational constraints. The use of NN and ML allows, in certain situations, to perform or assist complex signal processing operations at a reduced complexity compared to standard algorithms, while keeping the same performance level [7, 8]. This reduction of complexity at equal levels of performance is particularly interesting in the case of communication systems constrained in energy and/or computing capacities such as the systems considered in the IoT context.

Hardware efficiency and cost: Beyond algorithmic and modelling improvement enabled by NN and ML, NN are also interesting because of their dedicated efficient hardware implementations. Indeed, as it will be described in Chapter 2, the expressiveness of NN models lies in their layered architectures and not in complex individual mathematical operations. These computational structures of NN allow for very efficient, generic and cost effective hardware to be developed such as *Neural processing Unit* (NPU) already available on the market [9]. The performance per Watts of these devices drastically surpasses that of *Central Processing Unit* (CPU) or *Graphics Processing Unit* (GPU) [10]. Furthermore, as they are used in an increasing number of domains, including *Computer Vision* (CV), such hardware can be purchased off-the-shelf from a number of competing sources, including Intel, Nvidia, Google, etc., which further reduces its cost. Therefore, describing conventional signal processing algorithm using NN architectures is a way to benefits from the advantages of these hardware technologies even in the case where it does not improve the raw algorithmic or modelling performance. The wide development of high-performance hardware architectures dedicated to the execution of NN models makes it possible to consider their use in future IoT networks that are more performing, energy-efficient, software-based and therefore adaptable, less expensive, etc.

1.3 Methodology and Approach

This PhD work focus on different sub-blocks of the PHY layer in an IoT context, in particular those performing demodulation, equalisation and channel coding operations. On all these subjects a particular attention is thus paid to the complexity and the explainability of the proposed models. At first, the proposed approach aims at transcribing classical algorithms of the digital communication domain using the mathematical framework of NN. This first step, without necessarily bringing any performance or complexity gains compared to classical solutions, offers the advantage of fully explainable NN architectures. As a result, and with the same algorithmic performances, this transcription allows to take advantage of the interesting properties of the hardware architectures dedicated to the execution of NN. Then, ML methods are implemented to improve the performance of the proposed models thanks to the adaptation of their parameters with respect to the operating conditions. The proposed methodology thus seeks to take into account the operational constraints of IoT systems as well as to preserve as much as possible the explainability of the proposed models. And, as a matter of fact, the objective of this work is not to seek performance gains at all costs, but rather to improve the performance-to-complexity ratios of signal processing solutions for IoT systems. The chosen approach follow, in a certain way, the road-map towards a 6G AI-native air interface from [11] and is summarised hereafter:

Step 1: Deterministic NN architecture are devised from conventional signal processing algorithms.

Step 2: The introduction of ML enables performance enhancements of said functional processing blocks.

Step 3: The concatenation of NN-based block enables the joint optimisation of parts of the communication chain.

1.4 Research Plan

Following the above methodology, the research work described in this manuscript is structured around the following 4 chapters (see Figure 1.1):

Chapter 2 - Theoretical Foundations on Artificial Intelligence and Machine Learning: As this work aims at applying AI and ML algorithms for signal processing at the physical layer of wireless communication systems, Chapter 2 clarifies the different concepts related to AI that will be manipulated in the rest of the study.

Chapter 3 - Artificial Intelligence at the Physical Layer: The third chapter describes the subject of the study, namely the PHY layer of future cellular IoT network. It provides a brief overview of the main functions of the PHY layer of a digital communication systems, offers a quick survey of the use of AI techniques at the PHY layer and discusses the main opportunities and challenges related to the use of such techniques in this context.

Chapter 4 - Describing Signal Processing Operations Using Neural Networks: The fourth chapter studies, as a first step toward the integration of AI at the PHY layer, how conventional signal processing algorithms can be described as low-complexity NN structures. At first, NN structures for single and multi-path channel equalisation are described. Then, low complexity demodulation structures are introduced. Finally, a simple NN based PHY layer prototype with realistic implementation constraints, including single path equalisation and demodulation, is implemented using *Software Defined Radio* (SDR).

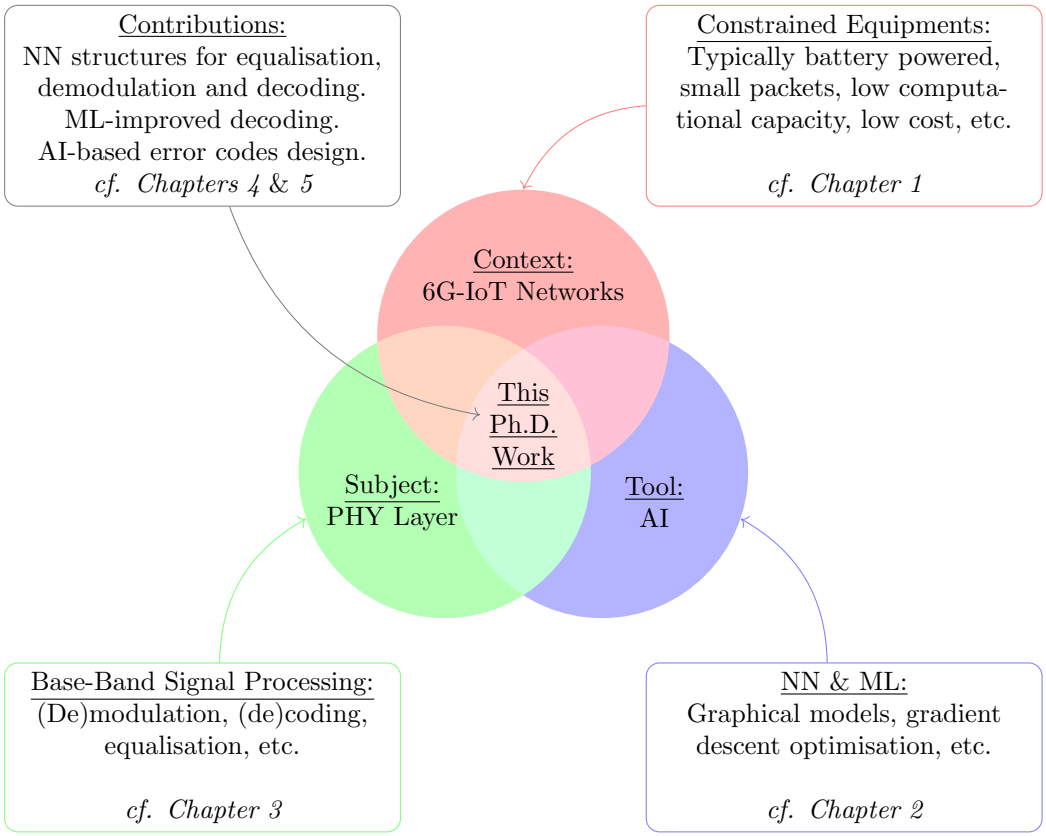


Figure 1.1: AI Models for Digital Signal Processing at the Physical Layer in a 6G IoT Context

Chapter 5 - Learning Process at the Air Interface: Application to Channel Coding: Finally, the fifth chapter discusses the application of ML procedures to NN-based signal processing structures at the PHY layer, and more particularly for channel coding. Following the approach introduced in Chapter 4, this chapter starts by describing how one can express a conventional belief propagation decoding algorithm using a differentiable graphical model. Then, ML techniques are used to improve the decoding performance of proposed decoder in a blind manner. Finally, an end-to-end approach is used for the joint design of performing coding schemes and associated decoders for short to medium packet sizes compatible with the constraints of IoT scenarios. The proposed solution reaches performance comparable to that of state-of-the-art *Low Density Parity Check* (LDPC) codes.

1.5 Contributions

Hereafter, the references to the different contributions made in the course of this thesis work are provided:

1.5.1 Conference and Journal Publications

[**12**] G. Larue, M. Dhiflaoui, L.-A. Dufrene, Q. Lampin, P. Chollet, H. Ghauch, and G. Rekaya, "Low-Complexity Neural Networks for Baseband Signal Processing," in *2020 IEEE Globecom Workshops*, pp. 1–6, 2020

[**13**] G. Larue, L.-A. Dufrene, Q. Lampin, P. Chollet, H. Ghauch, and G. Rekaya, "Blind Neural Belief Propagation Decoder for Linear Block Codes," in *2021 Joint European Conference on Networks and Communications & 6G Summit*, pp. 106–111, 2021

[**6**] C. F. Miltiadis *et al.*, "Pervasive Artificial Intelligence in Next Generation Wireless: The Hexa-X Project Perspective," in *Proceedings of the First International Workshop on Artificial Intelligence in Beyond 5G and 6G Wireless Networks*, vol. 3189, 2022

[**14**] G. Larue, L.-A. Dufrene, Q. Lampin, H. Ghauch, and G. Rekaya, "Neural Belief Propagation Auto-Encoder for Linear Block Code Design," *IEEE Transaction on Communications*, 2022

1.5.2 Patents

[**15**] L.-A. Dufrene, Q. Lampin, and G. Larue, "Procédés et dispositifs pour geler ou adapter les paramètres d'un réseau de neurones utilisé dans un réseau de télécommunications," 2020. WO2022144520A1 - FR3118519A1

[**16**] Q. Lampin, G. Larue, and L.-A. Dufrene, "Procédé et système d'adaptation d'un réseau de neurones utilisé dans un réseau de télécommunications," 2020. WO2021255362A1 - FR3111454A1

1.5.3 Open Source

All the results and source code from the journal publication [14] are available in open-source at: `https://github.com/Orange-OpenSource/GNBP`.

1.5.4 Hexa-X Project

As mentioned above, this work contributed to the Hexa-X project and has led to an active participation in numerous meetings of the "*AI-Driven Communication and Computation Co-Design*" *Work Package* (WP), which aims to improve the performance of the 6G air interface through low-complexity AI and ML mechanisms, and to devise concepts for the 6G network as a distributed learning platform. In particular, multiple discussions and technical presentations were conducted during the meetings of the working group, subdivided into three tasks: "*4.1: Gap Analysis for AI-Driven Communication and Computation Co-Design*", "*4.2: AI-Driven Air Interface Design*" and "*4.3: Methods and Algorithms for Sustainable and Secure Distributed AI*".

The discussed proposals were then documented in three deliverables intended for the European Commission: "*D4.1: Gap Analysis*" [17], "*D4.2: Initial Solutions*" [18] and the upcoming "*D4.3: Solutions*". The publications [13, 14] mentioned above have also been provided as Hexa-X contributions. In particular, the publication [6], is a joint publication initiative of the WP4 Hexa-X members, to which a contribution on the issue of coding has been made.

CHAPTER 2

THEORETICAL FOUNDATIONS ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Contents

2.1 A Brief History of Computer Science and Artificial Intelligence . 10
2.2 Neural Networks and Related Computational Structures 12
2.2.1 A Simple Building Block: The Formal Neuron 12
2.2.2 Adding Width: Standard Neurons Layers 13
Convolutional Layers 14
Recurrent Layers 17
2.2.3 Increasing the Depth: Some Neural Networks Structures 19
Common Activation Functions and their Use 21
2.2.4 Probabilistic Reasoning and Related Structures 22
Probability Background 22
Probabilistic Models and Inference 25
Structured Probabilistic Graphical Models, Factor Graphs and Associated Inference Algorithms 28
2.3 Neural Networks and Hardware Architectures 33
2.4 Learning and Optimisation 33
2.4.1 The Main Classes of Learning Problems 36
2.4.2 Optimisation of Neural Networks 37
Empirical Risk Minimisation Principle 37
Loss Functions 38
Over-fitting, Under-fitting and Regularisation 38
When to Halt the Training 38
Decomposing the Learning Problem 40
Gradient-Based Optimiser 40
Gradient Back-Propagation and Automatic Differentiation 43
2.5 Conclusion of the Chapter 45

Foreword

This manuscript focuses on how AI and ML algorithms could benefit the physical layer of next generation wireless communication systems. The goal of this chapter is therefore to clarify the principal AI and ML concepts, thus providing the reader with the necessary understanding of the tools that will be manipulated throughout this manuscript. First, a brief history of AI and a general description of the associated concepts will be provided. Then, the standard NN structures and computational models will be explained. A short description of the hardware architectures supporting the execution of NN will be described. Finally, some of the related ML optimisation algorithms will be introduced.

2.1 A Brief History of Computer Science and Artificial Intelligence

The ancient Greek mechanism of *Antikythera* [19], the earliest known example of an analogue computer (2^{nd} Century BC); *Euclid's algorithm* [20], a step-by-step method for calculating the greatest common divisor of two integers (3^{rd} century BC); and many other examples around the world, testify to mankind's age-old dream of machines and algorithms capable of automatically solving complex tasks [21]. While the early examples of computing mechanisms were single purpose machines, the pioneering works of, among many others, Charles Babbage (1791 - 1871) [22], Ada Lovelace (1815 - 1852) [23] or Alan Turing (1912 - 1954) [24] have led to the advent of modern computer science and general purpose computers [25]. Babbage, while developing a mechanical machine for calculating mathematical tables, had the idea of incorporating the programming cards from the *Jacquard Machine* [26], the reading of which would provide calculation instructions to his device. In a way, this versatile and programmable *Analytical Engine* is often considered the ancestor of modern computers [27]. Lovelace had a great interest in Babbage device for which she described what is considered the first program, following modern computer science concepts such as machine language or conditional statements and loops [23]. One of the conceptual breakthrough made by Lovelace was to acknowledge that such device could do much more than manipulating numbers to numerically solve analytical mathematical problems [25]. These numbers could also be used as an abstract representation of virtually any concepts, *e.g.* text, and manipulated to perform operations on them [25]. Nearly a century later, Alan Turing was involved in the formal definition of algorithms and the general concepts of computing machine and theory of computation [24]. He conceptualised the *Turing Machine* [28], a mathematical model describing an abstract machine capable of implementing any computer algorithm and was able to prove the existence of fundamental limits on computing machines and algorithms. Turing foresaw the potential of computers as intelligent machines, provided that it is possible to define that notion, and devised one of the first test of artificial intelligence, the *Turing test* [29]. This test is designed to assess a machine's ability to demonstrate intelligent behaviour. An evaluator is asked to have a written conversation with both another human and a machine designed to write human-like responses. If the evaluator is unable to distinguish the human from the machine consistently, the machine is deemed to have successfully completed the test. Turing estimated that computers would be able one day to successfully pass the test and predicted in a pioneering vision that learning processes would play an important role in the advent of performing computers and algorithms. At about the same time, the first digital electronic computer, the ENIAC, was created (1945) [30] and two years later, J. Bardeen, W. Shockley and W. Brattain invented the transistor (1947) [31]. These inventions would spark the digital electronics revolution, as well as the steady increase in computing capacity and speed, and the miniaturisation of devices, without which modern AI techniques would never have been possible.

AI is nowadays a field in great expansion, with a lot of promising applications and research challenges, that tries to answer, in many different ways, the broad and somewhat vague question: can a machine be intelligent? The term AI itself being semantically saturated, it is difficult to define it in a generic way. Nonetheless, one could broadly define an AI as:

> "Any device that perceives its environment and takes actions that maximise its chance

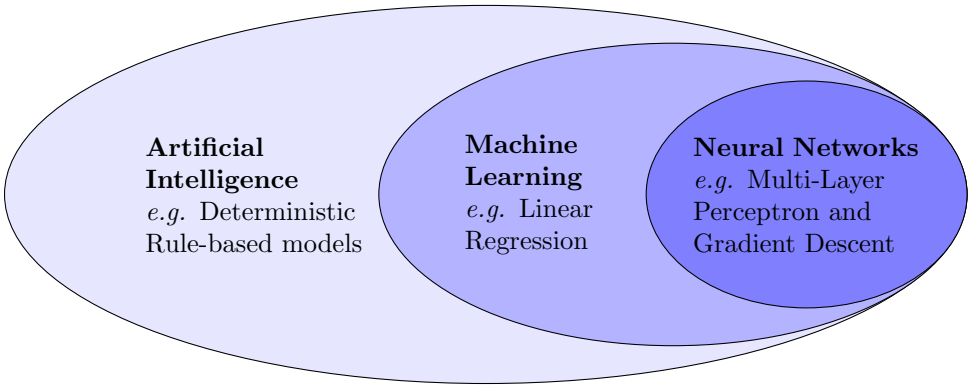


Figure 2.1: A Simplified Hierarchy of AI Related Concepts
To disambiguate this schematic representation, we should emphasise that NN are purely functional models, as will be described in further sections, and are thus not necessarily associated with ML algorithms (although it is often the case).

of successfully achieving its goals."[32]

Solving computationally intensive problems using expert knowledge were among the first successful applications of AI. These problems, particularly difficult for human beings, were relatively simple for a computer thanks to well defined handcrafted algorithmic rules. Ironically, the real challenge turned out to be solving rather simple, intuitive, problems for human beings but difficult to formalise as a list of rules, *e.g.* recognising objects in images. For such problems, and as imagined by A. Turing in 1950, a paradigm shift in algorithmic development took place. Where the expertise of developers was focused on the development of the algorithms solving the actual problems, it gradually shifted to the development of learning algorithms capable of automatically finding solutions to these unsolved problems through data-driven approaches. These algorithms are commonly referred to as ML algorithms. The latter usually try to maximise performance metrics of a parametric model on given data-sets in order to learn to solve certain complex problems. It should thus be emphasised that the ML algorithm is fundamentally different from the functional model for which it tries to find an effective parameterization. Among AI models, ANN, or more simply NN, are a particularly popular mathematical framework loosely inspired by biological neurons. These graphical models are based on the composition of a large number of local elementary operations, called neurons, representing much more complex global functions. These models, sometimes referred to as *connectionist* models, offers efficient hardware implementations and are particularly suitable to the application of ML procedures, hence their successful application to a wide range of domains and the increasing attention they receive from various research fields, among which their potential applications at the PHY layer of next generations of cellular networks. Such a hierarchy of concepts is described in Figure 2.1.

Computer have long been much better than human at solving well-defined complex numerical problems but only recently have they begun to match or even surpass their abilities at *e.g.* recognising images. As an illustration, the (top-5) accuracy of a human annotator on the image classification *ImageNet* challenge is measured at 5.1% or errors (optimistic estimation gives an error rate of 2.4%) [33] where the best model to date shows a (top-5) accuracy of 99.02% [34]. The success of modern AI, such as DeepMind Alpha-Go[35] or Open-AI GPT-3[36], finds its roots in a few key technical advances, namely the democratisation of GPU accelerated computation and subsequent increase in compute capability, the development of *Deep Neural Network* (DNN) models, theoretical advances in ML techniques and the availability of massive data-sets. Even though, current *State of the Art* (SOTA) models are examples of relatively *narrow AI*, in the sense that they are rather specialised on certain tasks, they pave the way to the grail of *general AI*. Only seventy years after asking the question "Can machine think?" in his seminal paper [37], one could realistically imagine that the test imagined by the father of modern computing, A. Turing, could be successfully passed by an AI in a relatively near future, testifying to the impressive progress made in this field.

We acknowledge that this brief summary is just a glimpse into the history of AI, ML and

computer science, and we refer the reader to [38] for a more detailed view. However, we hope that it allows the reader to grasp the historical trends that have led to the computer technologies we know today, and in particular the impressive pace of technical progress in recent years, which augurs for future developments, particularly in the field of AI.

2.2 Neural Networks and Related Computational Structures

As described in previous Section, NN and related ML algorithms are among the main enablers of modern AI techniques. The elementary building block of a NN is a neuron. This parametric model computes a simple operation, usually the weighted sum of its inputs, the addition of a bias and the application of a non-linear activation function. Individual neurons are then combined into layers, which in turn are combined to form a NN. The hierarchical composition of a high number of these simple parametric functions allows the NN to express complex operations. Some of the theoretical properties of NN make them particularly interesting as they could potentially be applied to any kind of problem. Indeed, simple structures such as *Multi-Layer Perceptron* (MLP), described in 2.2.3 were proven to be universal function approximators [39], and *Recurrent Neural Network* (RNN) to be Turing complete [40]. Furthermore, their interesting graphical structures make them particularly efficient with some ML algorithms, *e.g. Back-Propagation* (BPROP), are prone to efficient hardware implementations. These advantages led to successful applications in *e.g.* CV or *Natural Language Processing* (NLP), and NN are now widely adopted in several field and of high interest for many other research domains, among which next generations communication systems.

This section will go through the standard computational structures and probabilistic graphical models from the neural networks literature to provide the reader with the material necessary to the understanding of this manuscript.

2.2.1 A Simple Building Block: The Formal Neuron

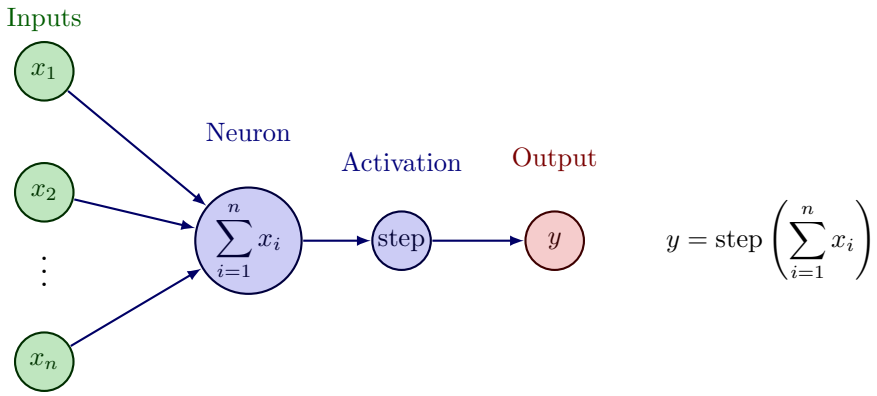


Figure 2.2: McCulloch and Pitts Formal Neuron Model
step(x) denote the step function, equal to 1 if $x > 0$ and 0 otherwise.

Early work on *Artificial Neural Network* (ANN) started as the crossroad of neurosciences, mathematics and computer science. While studying the behaviour of biological neurons, *McCulloch and Pitts* proposed in 1943 a first mathematical model of a neuron, defined as a threshold logical unit [41] as shown in Figure 2.2. In this model, the neuron fires whenever the sum of its inputs is positive. This on-off behaviour is implemented using the step function as a non-linear *activation* function.

This model was later refined and called *Perceptron* by *F. Rosenblatt* in 1958 [42]. In the modern sense, the Perceptron is defined as the binary classifier function $n(\boldsymbol{x}) = \text{step}\left(\sum w_i x_i + b\right)$, where $\boldsymbol{w} = \{w_i\}_{i=1}^{n} \in \mathbb{R}^n$ are trainable weights and $b \in \mathbb{R}$ is the bias of the neuron, and

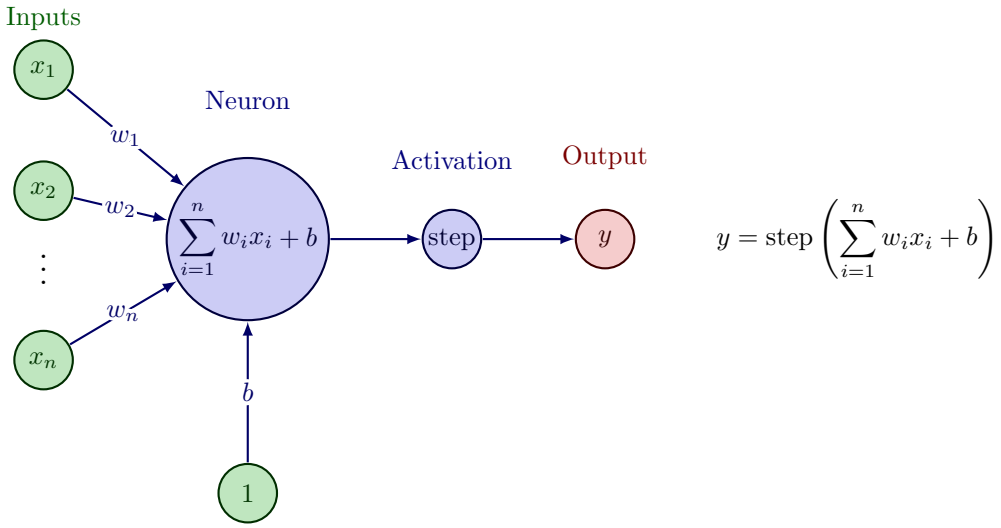


Figure 2.3: Modern Definition of a Perceptron

$\boldsymbol{x} = \{x_i\}_{i=1}^{n} \in \mathbb{R}^n$ are its inputs, as represented in Figure 2.3. These trainable parameters allow to weight the relative contributions of the inputs to the output and define the threshold above which the neuron should be activated.

The name Perceptron is often misused to describe any artificial neuron, the individual brick of neural networks, used in problems other than binary classification, albeit with different activation functions. These neurons are usually represented in a more compact form where the activation and bias (and possibly the weights) are implicitly represented as part of the neuron, as shown in Figure 2.4. This is one of the most commonly accepted definition of an artificial neuron. Note that while the term Perceptron refers to a single neuron, the *Perceptron algorithm* usually refers to a (single) layer of multiple Perceptron neurons. Hence, the term Perceptron is also often used to refer to a single layer of neurons.

Equivalently, a neuron can be seen as a parametric projection of its input space onto an axis, defined by its weights. The activation function then introduces a non-linearity on this projection.

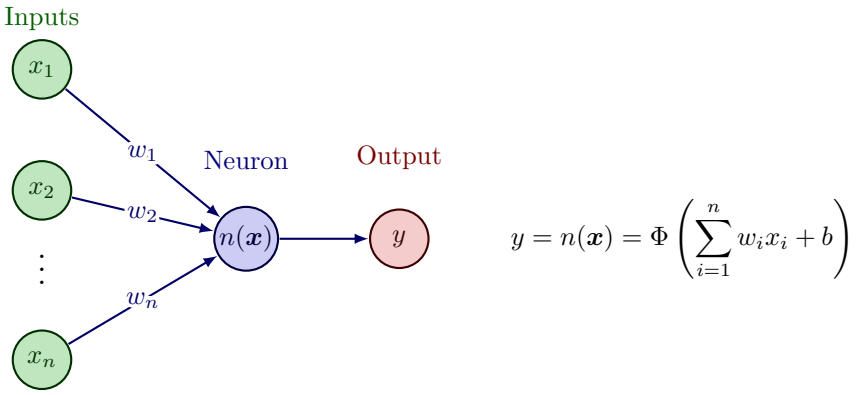


Figure 2.4: Compact Representation of a Neuron
$\Phi(x)$ denotes the arbitrary neuron's activation function.

2.2.2 Adding Width: Standard Neurons Layers

Combining neurons, in a so called layer, allow to compute in parallel the features associated to each of them. All the neurons from a layer usually share the same activation function. From the

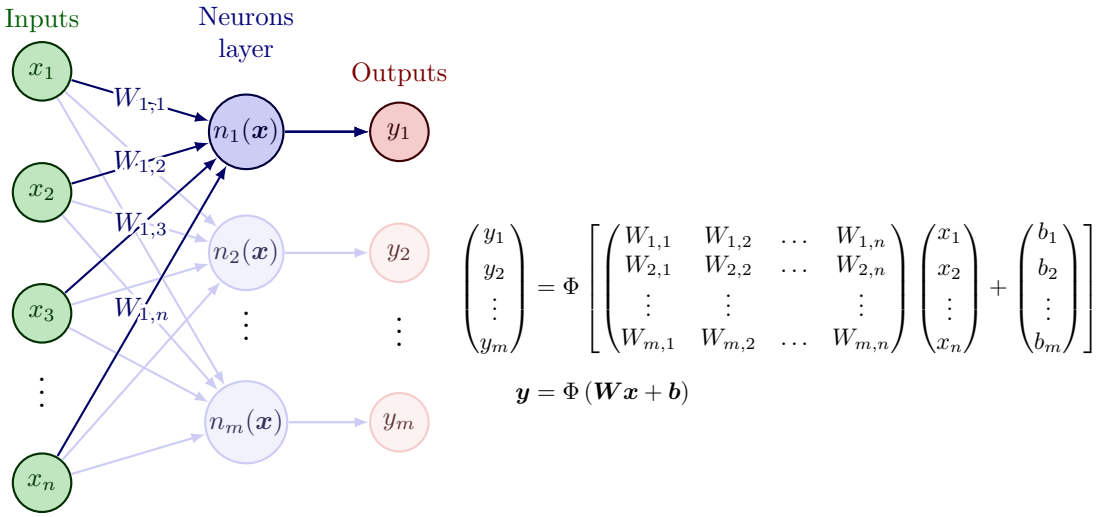


Figure 2.5: Standard Definition of Neuron Layer

definition of a neuron provided in previous section, one can define the computation associated with a layer of neurons as the matrix multiplication of the weight matrix $\boldsymbol{W} \in \mathbb{R}^{m,n}$ by the input vector $\boldsymbol{x} = \{x_i\}_{i=1}^{n} \in \mathbb{R}^n$, the addition of the bias vector $\boldsymbol{b} = \{b_j\}_{j=1}^{m} \in \mathbb{R}^m$ and the element-wise application of the same activation function $\Phi(x) : \mathbb{R}^m \to \mathbb{R}^m$. This simple structure is often referred to as a a *dense* or *fully-connected* layer. Note that here we represent the calculation of the layer output with respect to a single input vector $\boldsymbol{x}$. One can easily express the output of the layer with respect to several input vectors, or what is commonly called a *batch*, using matrix notation:

$$\begin{aligned} \boldsymbol{Y} &= \Phi\left(\boldsymbol{W}\boldsymbol{X} + \boldsymbol{b} \cdot \boldsymbol{1}^T\right) \\ \begin{pmatrix} \boldsymbol{Y}_{:,1} & \dots & \boldsymbol{Y}_{:,k} \end{pmatrix} &= \Phi\left(\boldsymbol{W}\begin{pmatrix} \boldsymbol{X}_{:,1} & \dots & \boldsymbol{X}_{:,k} \end{pmatrix} + \begin{pmatrix} \boldsymbol{b} & \dots & \boldsymbol{b} \end{pmatrix}\right) \\ \begin{pmatrix} Y_{1,1} & \dots & Y_{1,k} \\ \vdots & & \vdots \\ Y_{m,1} & \dots & Y_{m,k} \end{pmatrix} &= \Phi\left[\begin{pmatrix} W_{1,1} & \dots & W_{1,n} \\ \vdots & & \vdots \\ W_{m,1} & \dots & W_{m,n} \end{pmatrix}\begin{pmatrix} X_{1,1} & \dots & X_{1,k} \\ \vdots & & \vdots \\ X_{n,1} & \dots & X_{n,k} \end{pmatrix} + \begin{pmatrix} b_1 & \dots & b_1 \\ \vdots & & \vdots \\ b_m & \dots & b_m \end{pmatrix}\right] \end{aligned} \tag{2.1}$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are matrices representing a batch of k inputs/outputs (column) vectors. To make the addition of the bias vector compatible with the shape of the matrix resulting from the multiplication of the input batch by the weight matrix, one need to perform a so-called broadcasting operation. The latter operation, simply represented by $\boldsymbol{b} \cdot \boldsymbol{1}^T$, consists in duplicating and concatenating the bias vector k times. One should be aware that, although often encountered in NN structures, broadcasting operations are usually not explicitly written, *e.g.* $\boldsymbol{A} + \boldsymbol{b} \cdot \boldsymbol{1}^T$ is simply noted as $\boldsymbol{A} + \boldsymbol{b}$ and the reader needs to infer the correct shape that $\boldsymbol{b}$ should take.

The computational flow of a NN layer, centred on a few simple operations, notably matrix multiplications, lends itself to efficient parallel hardware implementations, as will be detailed in Section 2.3.

Certain types of problems and data have inherent structures that can be advantageously incorporated into computational models. The following sections deal with the most classical structures usually used to process data with local patterns, such as images, or data with temporal relationships, such as languages, namely the *convolutional* and *recurrent* layers.

Convolutional Layers

Some grid-spaced data exhibits local patterns, *e.g.* images can be represented as a 2D grid of pixels in which the pixels values are generally related to that of their grid neighbours due to the underlying structure of the image. When one wants to be able to detect these patterns,

which may be anywhere in the input data, *e.g.* determine whether an image contains a certain object, the processing structure must be position-invariant (also referred to as shift invariant) with respect to the data and integrate this notion of locality into the calculation. A mathematical operator that can represent such a behaviour is the *convolution*, which allows to apply the same small transform to all the elements of the entire input.

Given two signals x and h, the convolution of x by h is the integral of the product of the two signals after one is reversed and shifted, evaluated for all shift values:

$$y(\tau) = (x * h)(\tau) = \int_{t=-\infty}^{+\infty} x(t)h(\tau - t)dt \tag{2.2}$$

Similarly for discrete signals:

$$y[n] = (x * h)[n] = \sum_{m=-\infty}^{+\infty} x[m]h[n - m] \tag{2.3}$$

Convolution has applications in many fields, notably in the context of electrical engineering and communication where it is used to compute the response of various *Linear Time Invariant* systems, *e.g.* a filter or a channel, to an input x given their impulse response h. A closely related operator, used to assess the degree of similarity of two signals x and y, is the *cross-correlation*. It is defined as the integral of the product of one signal by the shifted complex conjugate of the other, evaluated for all shift values[1]:

$$R_{xy}(\tau) = (x \star y)(\tau) = \int_{t=-\infty}^{+\infty} x(t)y^*(t - \tau)dt \tag{2.4}$$

Similarly with discrete signals:

$$R_{xy}[n] = (x \star y)[n] = \sum_{m=-\infty}^{+\infty} x[m]y[m - n] \tag{2.5}$$

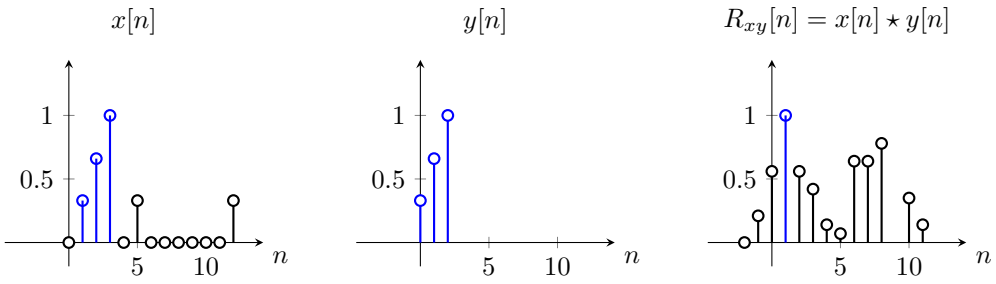


Figure 2.6: Cross-Correlation of Two Signals x and y
The cross-correlation result exhibit a maximum at $n = 1$ which correspond to the position of the signal y that we want to detect in the signal x.

Figure 2.6 shows an example where the cross-correlation operator is used to extract the position of a characteristic waveform within a signal in a similar way to that of *matched filters*.

In the field of NN, the convolutional layers (and corresponding networks) use the same principle. In such context the signal x is usually referred to as the *input* and the reference signal y as the *kernel*. The outputs are called the *(extracted) features* or the *feature map*. Although, they are called *convolutionnal*, they usually rely on the cross-correlation operator. One of the great interest in using convolutional layers (aside from the previously described shift invariant property) is their so called *sparse interaction* and *parameter sharing* properties which lead to very efficient computation and frugal memory usage when compared to standard fully-connected layers.

[1]The cross-correlation of x and y is thus equivalent to the convolution of x by y^*_-, where y^*_- is the reversed complex conjugate of y.

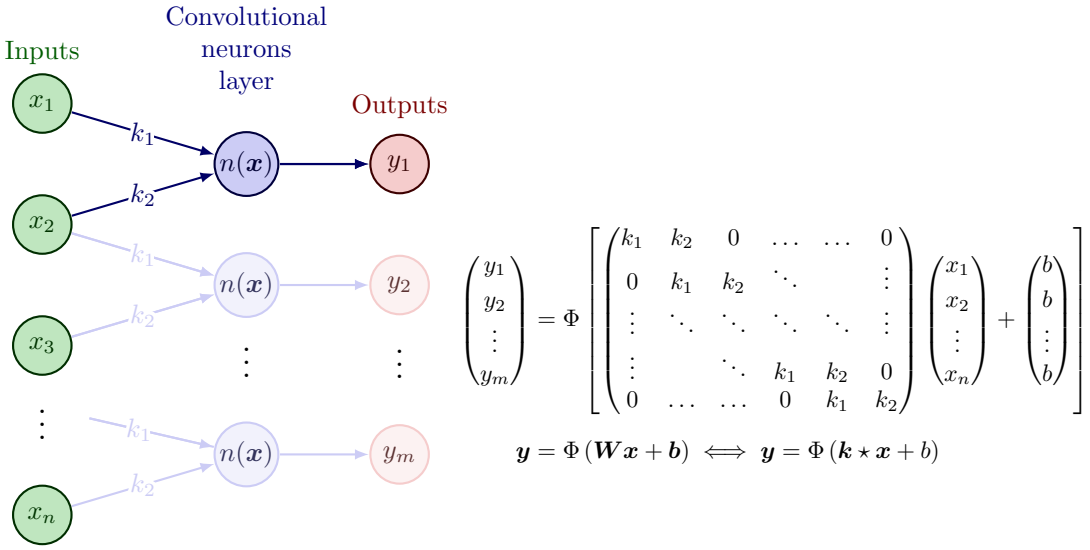


Figure 2.7: 1D Convolutional Layer
A kernel of size 2 is applied to a 1D input of size n. Without padding and with a stride of 1, this lead to an output of size $m = n - 1$. The multiplication of a *Toeplitz* weight matrix by the input vector and the addition of the same bias to all outputs describe the equivalent computation with a conventional dense layer.

The Figure 2.7 shows an example of a 1D convolutional layer applying a kernel of size 2 to an input of size n to compute $n - 1$ outputs. Similarly to a conventional dense layer, a bias is added before applying the activation function Φ, although it is constant for all outputs, *i.e.* it is part of the kernel. As shown on Figure 2.7 an equivalent computation using a dense layer could be defined although much more complex. In this example, and supposing that this structure is fit to the problem, the convolutional layer requires 2 weights and one bias compared to the $n(n-1)$ weights and $n - 1$ bias required by the equivalent dense structure. Furthermore it requires only $2(n-1)$ multiplications and $2(n-1)$ additions (including bias) compared to the $n(n-1)$ multiplications and $n(n-1)$ additions of the dense layer.

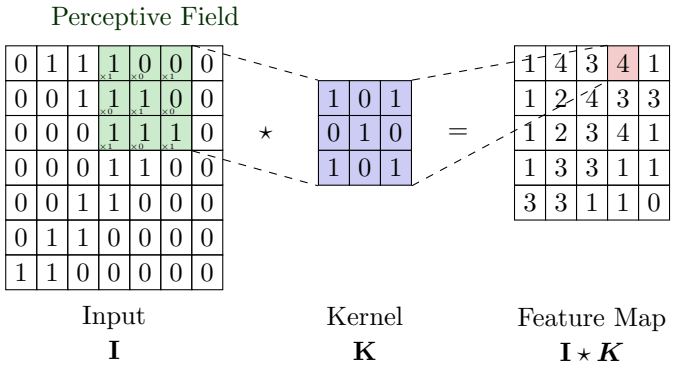


Figure 2.8: Example of a 2D Convolutional Layer
A kernel of size 3×3 is applied to a 2D matrix of size 7×7. Without padding and with a stride of 1, this leads to an output of size 5×5. No bias and activation are considered here.

Without loss of generality, this kind of structure can be applied to 2D data such as images, or volumetric data such as medical scan or 3D scenes. Figure 2.8 shows the example of a 2D convolution, similarly to what can be applied in the field of CV.

For more details on convolutional layers and associated concepts as well as their efficient implementations, we refer the interested reader to [1].

Recurrent Layers

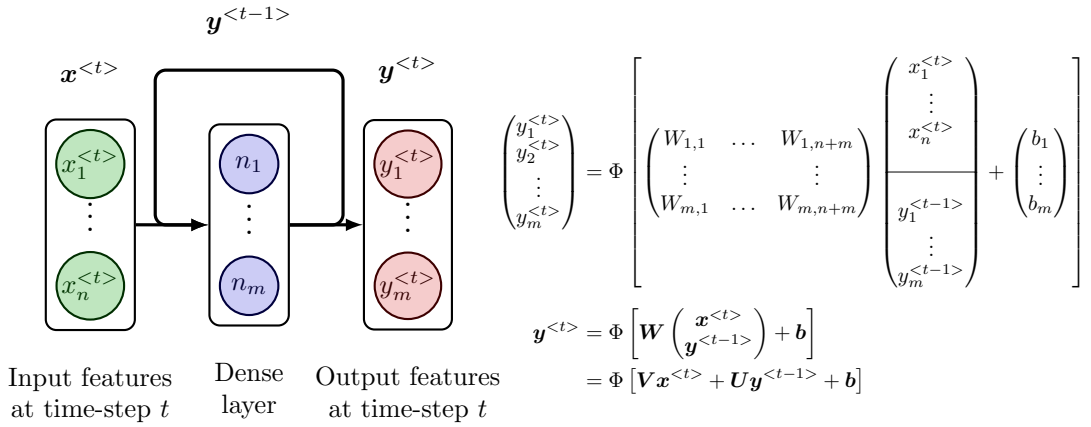


Figure 2.9: Compact Representation of a Standard Recurrent Layer
Arrows are a compact notation for a dense connection topology between the input vector and the recurrent layer nodes, but other topology could be considered, *e.g.* convolutional layers. The converging arrow denotes the concatenation of the input vector at time-step t and the state vector computed at previous time-step $t-1$. The diverging arrow represents a duplication. Activation and bias are not represented but could also be applied inside the recurrent layer.

Some data are ordered and/or present temporal structures, *e.g.* an english text is read from left to right and the knowledge of previous (or future) words in a sentence can be used to infer the current word. Recurrent layers and corresponding RNN models are specific NN structures that can take into account such data properties by performing a sequential operation (*i.e.* with memory) along one of the axes of the inputs. The said axis can be time, as in time series, a position in a sequence as in NLP, or any other dimension. Recurrent neurons and layers are usually represented using the compact form of Figure 2.9. In such standard recurrent layer, a feature vector $\boldsymbol{x}^{<t>} \in \mathbb{R}^n$ is provided at each of the τ time-step[2]. A dense layer of m neurons computes the matrix multiplication of its weight matrix $\boldsymbol{W} \in \mathbb{R}^{m,m+n}$ by the concatenation of the input feature vector at time-step t and the *state* vector from previous time-step, $\boldsymbol{y}^{<t-1>} \in \mathbb{R}^m$. Equivalently, this computation can be decomposed as the sum of the product of inputs $\boldsymbol{x}^{<t>}$ by the weight matrix $\boldsymbol{V} \in \mathbb{R}^{m,n}$ and the product of previous states $\boldsymbol{y}^{<t-1>}$ by the weight matrix $\boldsymbol{U} \in \mathbb{R}^{m,m}$. A bias and an activation function can be applied, as in conventional dense layers. At first time-step, the state vector is arbitrarily initialised, usually to an all-zero vector.

The representation of Figure 2.9 can be confusing as to the different concepts involved, such as the number of input and output features, the number of time-steps, the internal states, the layer's outputs, etc. To clarify the latter, one can unfold the computational graph of a recurrent layer as shown on Figure 2.10[3]. Contrarily to previously described convolutive or dense layers, the recurrent layer is not a *feed-foward* layer although an efficient deterministic computational graph can be defined if the number of time-step τ is pre-defined. It thus becomes similar to having a stack of τ feed-forward layers with additional inputs and outputs all-along the computational graph.

All calculations performed at a given time-step, given the state and input vectors, are con-

[2]The term "time-step", derived from time series processing, is widely used even in cases where the RNN is not applied on inputs with such a temporal ordering. It should be kept in mind as it can sometimes lead to confusion.

[3]Concept disambiguation: A common misunderstanding is to confuse the number of features and the number of time-steps. In a simple time-series analysis setting, the input features can represent various quantities *e.g.* temperature T and pressure P, while the time-steps represent the evolution of these quantities over time, *e.g.* $\{\boldsymbol{x}^{<1>} = \{T[1], P[1]\}, \boldsymbol{x}^{<2>} = \{T[2], P[2]\}, \dots, \boldsymbol{x}^{<\tau>} = \{T[\tau], P[\tau]\}\}$. In such a setting, two features are considered and τ timesteps are provided. One should note, that although a temporal ordering might be present in the data, it does necessarily have to be reflected by the different time-steps, *e.g.* it could be perfectly fine to consider the following inputs sequence: $\{\boldsymbol{x}^{<1>} = \{T[1], P[1]\}, \boldsymbol{x}^{<2>} = \{T[1], P[1]\}, \dots, \boldsymbol{x}^{<\tau>} = \{T[1], P[1]\}\}$. A confusion can arise when considering features generated from a sliding window over the considered time-series *e.g.* $\{\boldsymbol{x}^{<1>} = \{T[1], T[2], T[3]\}, \boldsymbol{x}^{<2>} = \{T[2], T[3], T[4]\}, \dots, \boldsymbol{x}^{<\tau>} = \{T[\tau], T[\tau+1], T[\tau+2]\}\}$. In such a scenario, both the τ time-steps and the 3 features provided at each time-step reflect the temporal ordering of the data.

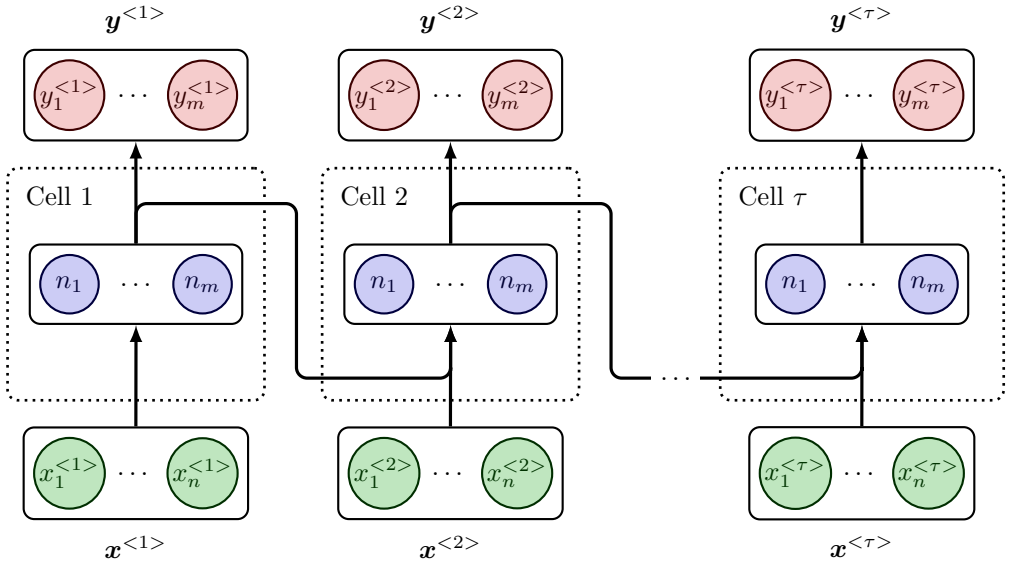


Figure 2.10: Unfolded Structure of a Standard Recurrent Layer
At each of the τ time-steps a feature vector of size n is provided to the so-called RNN cell. Based on the concatenation of that vector and the result from previous cell, usually referred to as the *state*, a new state of size m is computed. The new state is provided as the output of current time-step and passed onto next cell.

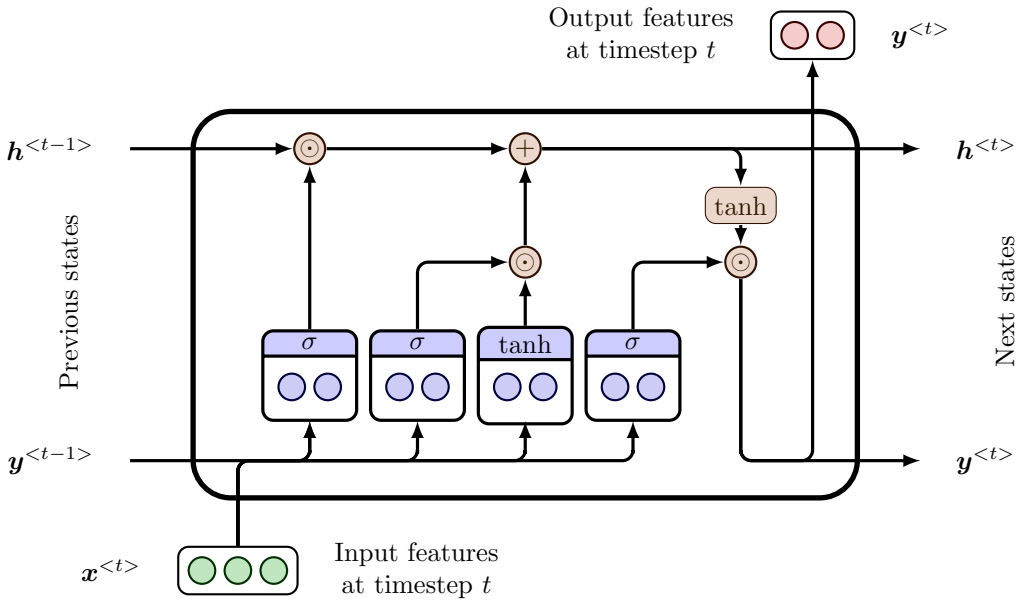
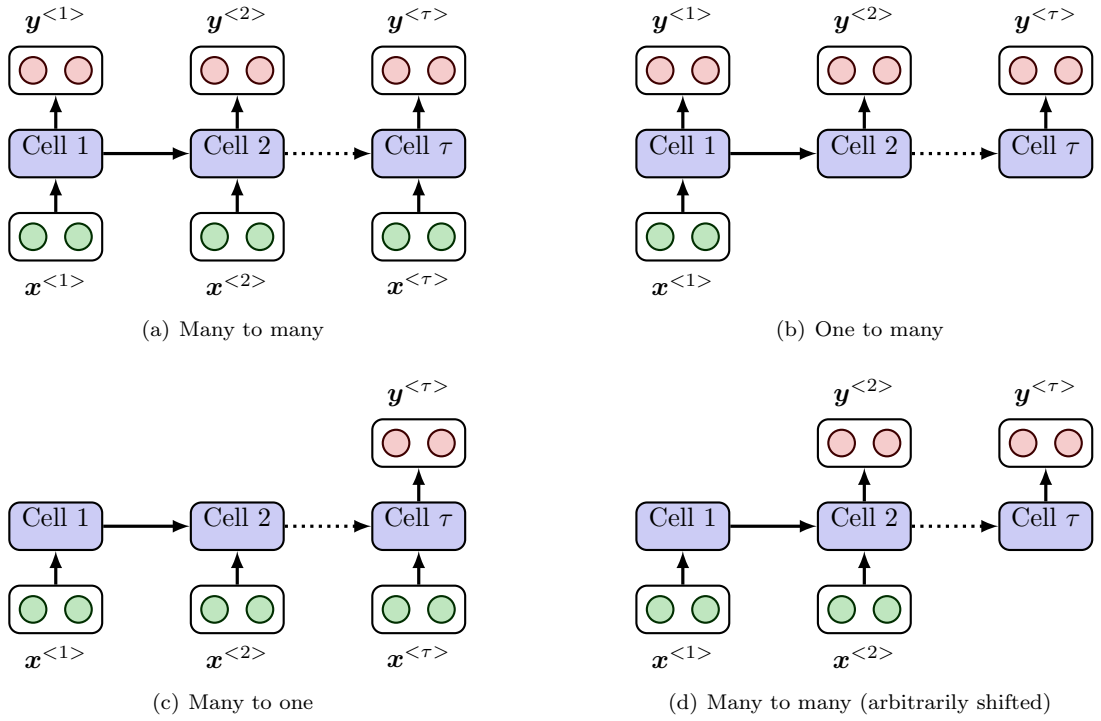


Figure 2.11: An Example of Gated Cell: the Long Short Term Memory Cell
As shown in this figure, the internal mechanism of an RNN cell can be more elaborate than the standard dense-structured RNN unit. The LSTM unit shown here maintains two internal state vectors, $\boldsymbol{h}$ and $\boldsymbol{y}$, which are passed from one iteration to the next. The $\boldsymbol{y}$ state vector is also used as the output of the cell. Gating mechanisms based on multiple dense layers with various activation functions are used to control the information flow in these internal states. This approach aims to preserve long-term dependencies while reducing the problems of gradient vanishing during BPROP. The divided blue rectangles represent dense layers and their respective sigmoid (σ) or tanh activations, while all other operators (in orange) represent point-wise operations ($\odot$ denotes multiplication, $+$ denotes addition and tanh denotes hyperbolic tangent activation). The converging arrows represent vector concatenation while the diverging arrows represent vector copy.

ceptually integrated in what is usually termed the computational *cell*. Each of the cell share the same parameters and computational graph although applied on different inputs and states. In the standard case, the cell simply applies the computation of a standard dense layer but more complex cell topology can be considered. Furthermore, the states and the outputs does not necessarily need to be the same, contrarily to the example of Figures 2.9 and 2.10. For example, the family of *gated* cells, such as *Long-Short Term Memory* (LSTM) [43] or *Gated Recurrent Unit* (GRU) [44], considers more complex cell topology, where a so-called *gate* controls the flow of information through various internal state vectors, usually different from the output vectors, in order to keep track of the long-term dependencies in the data while avoiding gradient vanishing during BPROP [43]. Figure 2.11 provides an example of such a gated cell in the form of a classic LSTM unit.



(a) Many to many

(b) One to many

(c) Many to one

(d) Many to many (arbitrarily shifted)

Figure 2.12: Various Recurrent Layer Inputs/Outputs Topology
Depending on the use-cases and data structures, the inputs/outputs topology of the recurrent layer can vary.

The configuration of input and output time-steps, so-called RNN topology, is adapted to each use-case, as shown in Figure 2.12. For example, a many-to-one topology could be used for text sentiment analysis where many input tokens are provided (letters or words in the text) to be processed sequentially and a single output is expected, namely that the text expresses positive or negative opinions. A many to many topology could be used for text translation, where multiple tokens, forming a sentence in a given language, are provided as input, and the translation of said sentence into another language is expected as output. Finally, a one to many topology could typically be used for image captioning where a single token, the image, is provided while a sequence of words describing the image is expected as an output.

For more details on recurrent layers, networks and sequence modelling we refer the interested reader to the excellent *Deep Learning* book, chapter 10 [1].

2.2.3 Increasing the Depth: Some Neural Networks Structures

Now that some of the principal layer types have been introduced, a proper definition of NN can be stated as the compositions of said layers.

One of the most classic NN structure is the MLP. MLP is a feed forward structure made out of a combination of standard dense layers as shown on Figure 2.13. Obviously, more complex

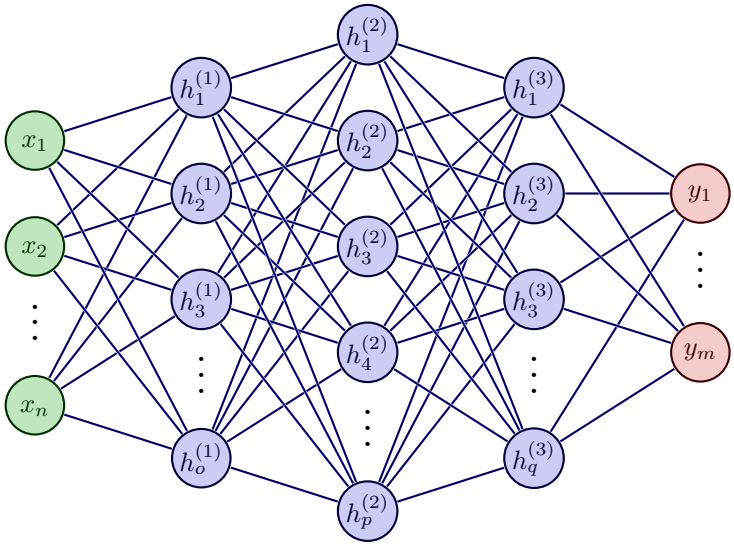


Figure 2.13: The Multi-Layer Perceptron
A standard MLP is defined as the composition of l dense layers whose activation functions and number of neurons can vary from one layer to another.

combinations of heterogeneous layers - *e.g.* RNN, *Convolutional Neural Network* (CNN), etc. - can be considered, but they are outside the scope of this introductory section. The global function described by the MLP is simply the composition of the functions of the l individual layers of the networks:

$$\begin{aligned} \boldsymbol{h}^{(1)} &= \Phi^{(1)}\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \\ \boldsymbol{h}^{(j)} &= \Phi^{(j)}\left(\boldsymbol{W}^{(j)}\boldsymbol{h}^{(j-1)} + \boldsymbol{b}^{(j)}\right) \forall j \in \{2, \ldots, l-1\} \\ \boldsymbol{y} &= \Phi^{(l)}\left(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right) \end{aligned} \tag{2.6}$$

where $\boldsymbol{h}^{(j)}$, $\Phi^{(j)}$, $\boldsymbol{W}^{(j)}$ and $\boldsymbol{b}^{(j)}$ are respectively the output, activation function, weights matrix and bias vector of the j-th layer of the MLP.

The above equation stresses the importance of the use of non-linear activation functions. Indeed, executing a MLP made out only of linear dense layers is equivalent to executing a single linear dense layer, (*i.e* a composition of linear map is a linear map) as shown recursively with the example of the identity activation function:

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)} \\ &= \boldsymbol{W}^{(l)}\left(\boldsymbol{W}^{(l-1)}\boldsymbol{h}^{(l-2)} + \boldsymbol{b}^{(l-1)}\right) + \boldsymbol{b}^{(l)} \\ &= \boldsymbol{W}^{(l,l-1)}\boldsymbol{h}^{(l-2)} + b^{(l,l-1)} \\ &\vdots \\ \boldsymbol{y} &= \boldsymbol{W}^{(l,l-1,\ldots,1)}\boldsymbol{x} + b^{(l,l-1,\ldots,1)} \end{aligned} \tag{2.7}$$

where $\boldsymbol{W}^{(l,l-1,\ldots,1)}$ and $b^{(l,l-1,\ldots,1)}$ are linear combinations of the weights matrices and bias vector of layers 1 through l.

Thus, and as stated before, one way to express more complex functions is to apply non-linear activation functions between the different layers of the network. The following section provides an overview of some of the most common activation functions and their implementation contexts.

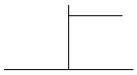
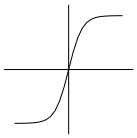
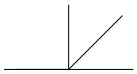
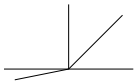
Name	Symbol	Definition	Derivative	Figure
Step	$H(x)$	$\begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$	$\delta(x)$	
Sigmoid	$\sigma(x)$	$\dfrac{1}{1+e^{-x}}$	$\sigma(x)(1-\sigma(x))^2$	
Hyperbolic Tangent	$\tanh(x)$	$\dfrac{e^{x}-e^{-x}}{e^{z}+e^{-z}}$	$1-\tanh(x)^2$	
ReLU	$\mathrm{R}(x)$	$\max(0,x)$	$\begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$	
Leaky ReLU	$\acute{\mathrm{R}}(x)$	$\max(\alpha x, x)$ $\forall \alpha \in [0,1]$	$\begin{cases} \alpha \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$	
Softmax	$\sigma_i(\boldsymbol{x})$	$\dfrac{e^{x_i}}{\sum_j e^{x_j}}$	$\dfrac{\partial \sigma_i(x)}{\partial x_k} = \begin{cases} \dfrac{e^{x_i}\sum_{j\neq i} e^{x_j}}{\left(\sum_j e^{x_j}\right)^2} \text{ if } k = i \\ -\dfrac{e^{x_i}e^{x_k}}{\left(\sum_j e^{x_j}\right)^2} \text{ if } k \neq i \end{cases}$	

Table 2.1: Common Non-Linear Activation Functions

Common Activation Functions and their Use

Table 2.1 lists the most commonly used non-linear activation, as-well as their derivative and plots. The choice of an activation function is mainly related to the class of problems considered as well as the activation numerical conditioning characteristics when running or learning a multi-layer model. On the one hand, the nature of the problem generally influences the choice of the activation function and size of the last layer of the network in order to format the outputs in a suitable way. On the other hand, numerical conditioning issues generally dictate the choice of the activation functions for the intermediate layers in order to ensure proper propagation of data (execution) and gradients (learning) in the network.

As for the last layer activation choice, and when considering *supervised learning* problems (which is primarily the case in this manuscript), it is common to distinguish *regression problems* from *classification* problems. In essence, classification aims to predict a class label while regression aims to predict a quantity. Classifications problems can themselves be subdivided in three categories:

- The *binary* or *single-class* classification problem where the model has a single output that must take the value 0 or 1, for example when one wishes to label emails as spam or not.
- The *multi-class single-label* classification problem where the model has several outputs, each relating to mutually exclusive classes, of which one must take the value 1 and the others 0 (this is commonly referred to as one-hot encoding). This situation is encountered, for example, when classifying entities among several mutually exclusive categories *e.g.* the four colours of playing cards {heart, diamond, spade, clover}.
- The *multi-class multi-label* classification where the model has several outputs, each referring

to a different class, of which several may take the value 1 while the others take the value 0. This may be encountered when one wishes to describe the presence or absence of multiple objects in a scene, *e.g.* there is a tree and a bike but no pedestrian on the picture.

While linear activation functions in the last layer satisfy the need for continuous dynamics of output variables in regression problems, non-linear activations, usually with binary behaviour, are well suited to classification problems [45]. The step function is the prototypical function for binary classification problems, although it is usually not used in practice due to its non-differentiability and subsequent incompatibility with gradient-based optimisation methods. The sigmoid function can be considered as a relaxed approximation of a step function, ideal to solve binary classification or multi-class multi-label problems using gradient descent. When dealing with multi-class single-label problems, the softmax function, a generalisation of the sigmoid that normalises the sum of the outputs of the network to 1, is particularly suited.

For the intermediate layers of the network, the hyperbolic tangent is usually preferred over the sigmoid because it is zero centred which is considered a desirable behaviour [46]. Still, both sigmoid and hyperbolic tangent functions suffer from the so-called vanishing gradient[4] problem [43, 47] that can be described by the fact that the value of the gradient gets close to 0 when $x >> 0$ or $x << 0$. This behaviour causes a slowdown in gradient based optimisation of DNN, eventually up to the point of completely blocking the back-propagation of the gradients to early layers of the networks and any subsequent update of the weights. From a mathematical standpoint, and as will be described in more detail in Section 2.4, the Jacobian matrices of the different layers contain derivatives with small numerical values so that the product of these matrices tends to 0 and leads to small updates of the parameters. To circumvent this vanishing gradient problem, the *Rectified Linear Unit* (ReLU) function has been proposed because of its constant gradient when $x \geq 0$ [48]. Still, when $x < 0$ the gradient of the ReLU is equal to 0 which can also block the weights update. This is the so-called dying ReLU phenomena [49]. Leaky ReLU and variants are widely adopted variants of the ReLU that reduce this problem with a small, but non-zero, gradient value when $x < 0$.

For a finer understanding of the reasons behind the above-mentioned good practices in the choice of activation functions, the reader is referred to Section 2.4. The latter provides additional information on learning processes in NN and related concepts such as differentiability, *Gradient Descent* (GD), BPROP, chain rule of derivatives, etc.

2.2.4 Probabilistic Reasoning and Related Structures

A complex, yet fundamental, aspect of the modelling of any phenomenon is uncertainty. The causes of this uncertainty are various and include our limited ability to observe all the relevant causes and the fact that even what we are able to observe may be observed with errors due to noise or imperfections in measurement processes. Because of this pervasive uncertainty, it is important to reason probabilistically. Probability theory is a mathematical framework that allows to make uncertain statements and reason in the face of uncertainty. It is a fundamental tool in many, if not all, fields of science, including AI where many probabilistic models have been developed. This section provides an outline of probabilistic models and associated inference methods relevant to this study. Readers willing to delve into a more formal and complete description of these concepts are invited to read [50, 51, 52].

Probability Background

Probabilistic reasoning involves working with variables that can take different values at random. Such variables are called random variables. A random variable is not of much use in itself as it simply describes a list of possible outcomes. However, if it is associated with a probability distribution it then becomes possible to describe the probability of occurrence of these outcomes in order to list not only what is possible, but also, and more importantly, what is likely. With some simplification, a *random variable* can be defined as a variable that can take on different, discrete or continuous, values at random in a sample space Ω. In this manuscript, x denotes a

[4] See Section 4.18 for a detailed description of the concept of gradient and related optimisation algorithms.

scalar random variable, while random vectors (random matrices respectively) are denoted by $\mathbf{x}$ ($\mathbf{X}$ respectively). The *probability distribution* of a random variable describes the likelihood that a random variable takes one of its possible outcomes. In this document, we adopt the notation $\mathrm{x} \sim D$ to indicate that the random variable x follows the probability distribution D.

When considering a discrete variable, *e.g.* when describing the outcome of a coin toss, the distribution of the random variable is usually described using a *Probability Mass Function* (PMF) $p_{\mathrm{x}\sim D}(\mathrm{x} = x)$, or simply $p_{\mathrm{x}}(x)$. This function expresses the probability that random variable x takes on value x under distribution D. A probability of 1 means that an outcome is certain and a probability of 0 means that it is impossible. A PMF must verify the following properties:

- $0 \leq p_{\mathrm{x}}(x) \leq 1 \forall x \in \Omega$, the probability of any possible outcomes is always comprised between 0 and 1.
- $\sum_{x\in\Omega} p_{\mathrm{x}}(x) = 1$, the probabilities of all possible outcomes must sum to 1.

For example, the probability of event "x equals a" can be noted as $\mathbb{P}_{\mathrm{x}\sim D}(\{\mathrm{x} = a\}) = p_{\mathrm{x}\sim D}(\mathrm{x} = a)$ where the brace notation $\{\mathrm{x} = a\}$ denote the event "x is equal to a". This can be written more concisely as $\mathbb{P}_{\mathrm{x}}\{\mathrm{x} = a\} = p_{\mathrm{x}}(a)$.

When considering continuous random variables, the sample space corresponds to one or multiple continuous intervals over the infinite set of real numbers $\mathbb{R}$. The probability distribution is then described using the continuous *Probability Density Function* (PDF), denoted as $f_{\mathrm{x}\sim D}(\mathrm{x} = x)$, or simply $f_{\mathrm{x}}(x)$. As the function is continuous, the PDF $f_{\mathrm{x}}(x)$ does not directly provide the absolute likelihood of the random variable x to be equal to x, the latter tending toward 0 (there is infinitely many different possible outcomes). Instead, a PDF can be interpreted as the relative likelihood or the probability of random variable x to takes on a value in the infinitesimal neighbourhood dx of x as $f_{\mathrm{x}}(x)dx$. Similarly to the PMF, the PDF must satisfy certain properties:

- $\int_{x\in\Omega} f_{\mathrm{x}}(x)dx = 1$, similarly to the PMF but replacing the discrete summation by an integral, the probabilities of all possible outcomes must sum to 1.
- $0 \leq f_{\mathrm{x}}(x) \forall x \in \Omega$, the PDF is always positive. Note that, contrarily to PMF, the PDF $f_{\mathrm{x}}(x)$ is not required to be smaller than one because of the infinitesimal nature of this description. Indeed this would not be the case *e.g.* when considering a PDF of the form $f_{\mathrm{x}}(x) = \delta(x)$ where $\delta(x)$ is the Dirac function, even though the area under the curve still equals 1.

With such continuous distribution we are generally interested in finding the probability that the random variable lies on a certain interval which can be computed by means of integration over that interval. For example assume we want to compute the probability that the random variable x is comprised between two numbers a and b. This can be expressed as $\mathbb{P}_{\mathrm{x}\sim D}(\{a \leq \mathrm{x} \leq b\}) = \int_a^b f_{\mathrm{x}\sim D}(\mathrm{x} = x)dx$ where the brace notation $\{a \leq \mathrm{x} \leq b\}$ denote the event "x is comprised between a and b". This can be written more concisely as $\mathbb{P}_{\mathrm{x}}\{a \leq \mathrm{x} \leq b\} = \int_a^b f_{\mathrm{x}}(x)dx$.

One can obviously consider to work with several random variable simultaneously. In such situation, a probability distribution over the different variables is known as a *joint probability distribution*. Similarly to the single variable case one can describe these joint probability distribution with multi-variate PDF or PMF. For example, a multivariate PMF would be noted as $p_{\mathrm{x}\sim D_1,\mathrm{y}\sim D_2}(\mathrm{x} = x, \mathrm{y} = y)$ or more concisely $p_{\mathrm{x,y}}(x, y)$. For continuous variables, a multivariate PDF is denoted as $f_{\mathrm{x}\sim D_1,\mathrm{y}\sim D_2}(\mathrm{x} = x, \mathrm{y} = y)$ or more concisely $f_{\mathrm{x,y}}(x, y)$.

One can sometime be provided with a joint distribution over many variables while being interested only by the distribution of a subset of these variables. Such distribution is known as a *marginal probability distribution* and can be found using what is know as the *sum rule*. The latter consists in the summation or integration of the joint distribution (depending of the continuous or discrete nature of the considered variables) over the sample spaces of the variables that are not of interest. For example, if considering a set of n discrete random variables $\{\mathrm{x}_1, \ldots, \mathrm{x}_n\}$, the marginal distribution over the sub-set made of the first $k < n$ random variables would be defined as:

$$p_{\mathrm{x}_1,\ldots,\mathrm{x}_k}(x_1,\ldots,x_k) = \sum_{x_{k+1}\in\Omega_{k+1}} \cdots \sum_{x_n\in\Omega_n} p_{\mathrm{x}_1,\ldots,\mathrm{x}_n}(x_1,\ldots,x_n) \tag{2.8}$$

which will be noted more concisely in this manuscript as:

$$p_{\mathrm{x}_1,\ldots,\mathrm{x}_k}(x_1,\ldots,x_k) = \sum_{\backsim x_1,\ldots,x_k} p_{\mathrm{x}_1,\ldots,\mathrm{x}_n}(x_1,\ldots,x_n) \tag{2.9}$$

where the $\backsim$ symbol denote the nested summation over all the variable except the ones listed.

Similarly, when considering continuous variables:

$$f_{\mathrm{x}_1,\ldots,\mathrm{x}_k}(x_1,\ldots,x_k) = \int_{x_{k+1}\in\Omega_{k+1}} \cdots \int_{x_n\in\Omega_n} f_{\mathrm{x}_1,\ldots,\mathrm{x}_n}(x_1,\ldots,x_n)\, dx_n \ldots dx_{k+1} \tag{2.10}$$

which will be noted more concisely in this manuscript as:

$$f_{\mathrm{x}_1,\ldots,\mathrm{x}_k}(x_1,\ldots,x_k) = \int_{\backsim x_1,\ldots,x_k} f_{\mathrm{x}_1,\ldots,\mathrm{x}_n}(x_1,\ldots,x_n) \tag{2.11}$$

One can define the *conditional probability* of an event A conditioned by an event B as the probability that the event A is realised under the assumption that the event B is realised. The conditional probability of event $\{A\}$ knowing that event $\{B\}$ has happened is denoted as:

$$\mathbb{P}\{A|B\} = \frac{\mathbb{P}\{A,B\}}{\mathbb{P}\{B\}} \quad \forall \quad \mathbb{P}\{B\} > 0 \tag{2.12}$$

From the above definitions one can derive the famous *Bayes rule* that allows to compute the probability of some hypothesis $\{H\}$ to be true given the observation of evidence $\{E\}$:

$$\mathbb{P}\{H|E\} = \mathbb{P}\{E|H\}\,\frac{\mathbb{P}\{H\}}{\mathbb{P}\{E\}} \tag{2.13}$$

where $\mathbb{P}\{H\}$ is the probability that hypothesis $\{H\}$ is true before any evidence is observed, also known as the *prior*. $\mathbb{P}\{E\}$ is the probability of seeing the evidence $\{E\}$, also known as the *marginal likelihood*. $\mathbb{P}\{E|H\}$ is the probability to observe the evidence $\{E\}$ given the hypothesis $\{H\}$ is true, also known as the *likelihood*. Finally, $\mathbb{P}\{H|E\}$ is the probability that hypothesis $\{H\}$ is true given the evidence $\{E\}$, which can be seen as an update over the prior given the evidences, also known as the *posterior*.

From the definition of conditional probability, one can derive the product rule of probability stating that a multivariate joint probability distribution can be factored into the chained product of their conditional probability:

$$\mathbb{P}\{X_1,\ldots,X_n\} = \mathbb{P}\{X_1\}\prod_{i=2}^{n}\mathbb{P}\{X_i|X_1,\ldots,X_{i-1}\} \tag{2.14}$$

Two events are said independent when $\mathbb{P}\{A|B\} = \mathbb{P}\{A\}$ (and *vice et versa*) and thus $\mathbb{P}\{A,B\} = \mathbb{P}\{A\}\,\mathbb{P}\{B\}$.

Assuming an infinitely large number of samplings, the expected value of a random variable (also called expectation or first moment) is the average of these samplings. Assuming that the distribution of said random variable is known, the expected value can be calculated as the weighted average of the possible outcomes according to their probability of occurrence, as follows:

$$\mathbb{E}_{\mathrm{x}\sim D}\{\mathrm{x}\} = \sum_i x_i p_{\mathrm{x}\sim D}(\mathrm{x} = x_i) \tag{2.15}$$

Which can similarly be written for a continuous variable:

$$\mathbb{E}_{\mathrm{x}\sim D}\{\mathrm{x}\} = \int x f_{\mathrm{x}\sim D}(\mathrm{x} = x)dx \tag{2.16}$$

The empirical mean is an unbiased estimator of the expected value in the sense that it converges to the latter when the number of sample tends to infinity:

$$\lim_{n\to+\infty}\bar{\mathrm{x}} = \lim_{n\to+\infty}\sum_{i=1}^{n}\mathrm{x}_n = \mathbb{E}_{\mathrm{x}\sim D}\{\mathrm{x}\}, \quad \mathrm{x_n}\sim D \tag{2.17}$$

Probabilistic Models and Inference

Lets suppose that a sequence of samples $\boldsymbol{d}$ is generated following an unknown random or *stochastic* process. Let's further assume that we want to build a probabilistic model, with parameters $\boldsymbol{\theta}$, whose outputs fit the observed sequence $\boldsymbol{d}$. Since the model is probabilistic, its outputs can be represented as a random variable $\mathbf{d}$.

In that context, the *likelihood function*[5] $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{d})$ denotes the probability that the model, with parameters $\boldsymbol{\theta}$, outputs a sequence equal to that of the observed sequence $\boldsymbol{d}$:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{d}) = \mathbb{P}_{\mathbf{d}\sim D}\{\mathbf{d} = \boldsymbol{d}; \boldsymbol{\theta}\} \tag{2.18}$$

where $\mathbb{P}_{\mathbf{d}\sim D|\boldsymbol{\theta}}\{\mathbf{d} = \boldsymbol{d}; \boldsymbol{\theta}\}$ denotes the probability that a random sequence $\mathbf{d}$ sampled from the distribution D of the chosen model under parameterization $\boldsymbol{\theta}$, takes the same values $\boldsymbol{d}$ as the observed realizations of the (unknown) data generating distribution.

Assuming independence between samples, the likelihood function can be simplified to a product of probabilities. In this case, it may be convenient to introduce the *log-likelihood function*, *i.e.* the logarithm of the likelihood function:

$$\log \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{d}) = \log \prod_{i=1}^{n} \mathbb{P}_{\mathrm{d}_i\sim D}\{\mathrm{d}_i = d_i; \boldsymbol{\theta}\} = \sum_{i=1}^{n} \log \mathbb{P}_{\mathrm{d}_i\sim D}\{\mathrm{d}_i = d_i; \boldsymbol{\theta}\} \tag{2.19}$$

Although this is only a matter of practical convenience, sums are generally easier to handle in many numerical and/or analytical aspects, such as differentiation or integration, which justifies such a notation. Furthermore, since the logarithm is a strictly monotonic function, it has no impact on maximisation or minimisation processes.

For example, suppose we observe a sequence of n coin flips and do not know whether the coin is fair or not. We choose to model this sequence of observations using a Bernoulli model in which the single parameter θ ($1-\theta$ respectively) is the probability of the coin landing head up (tail up respectively). Given a sequence of observations $\boldsymbol{d}$ the log-likelihood function is thus defined as:

$$\log \mathcal{L}(\theta, \boldsymbol{d}) = \sum_{i=1}^{n} \mathbb{1}(d_i = \mathrm{H})\log\theta + \mathbb{1}(d_i = \mathrm{T})\log(1-\theta) = n_{\mathrm{H}}\log\theta + n_{\mathrm{T}}\log(1-\theta) \tag{2.20}$$

where $\mathbb{1}(d_i = \mathrm{H})$ ($\mathbb{1}(d_i = \mathrm{T})$ respectively) is an indicator function equal to '1' when the i-th coin toss lands head up (tail up respectively) and '0' otherwise.

Figure 2.14 describes the log-likelihood function in such a scenario for different sequences of coin flips. It is clear in this model that if we observe a majority of coins that land head up (tail up respectively), the most probable value for the parameter θ of the chosen Bernoulli model is shifted to 1 (0 respectively). When the coins always land heads up (tails respectively) the log-likelihood function shows that the most probable value for the Bernoulli process is 1 (0 respectively). This estimator, which consists in computing the parameters that maximise the (log-)likelihood function, is commonly known as the *Maximum Likelihood Estimator* (MLE):

$$\boxed{\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} \triangleq \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{d})} \tag{2.21}$$

As shown earlier, the MLE provides a choice of parameters for the chosen probabilistic model most likely to yield the observed data. This choice is made solely on the basis of the available data. Therefore, it is particularly sensitive to small data sets for which it is prone to over-fitting. Sometimes knowledge of the phenomenon to be modelled provides prior information that can be advantageously used for parameter selection, especially when there are few data available. This idea is at the heart of *Maximum A Posteriori Estimator* (MAPE). This method attempts to answer the same question as MLE, but allows us to inject an *a priori* distribution on the value that the parameters could take to allow a better estimation of them. The joint probability of an

[5]The likelihood function should not be confused with a probability distribution.

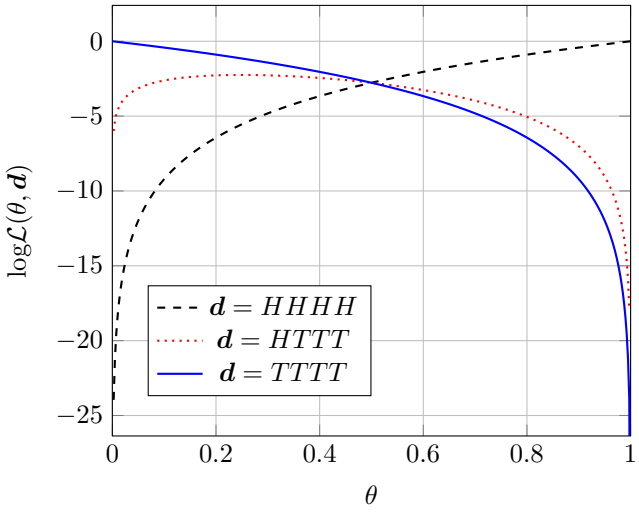


Figure 2.14: Log-Likelihood Functions of a Bernouilli Model for Sequences of Coin Flips
The likelihood functions of the chosen model are computed for three different sequences of coin flips. H stands for "heads up" and T for "tails up".

a priori distribution on the parameters of the model and its output can be defined and factored as follows:

$$\mathbb{P}_{\mathbf{d},\boldsymbol{\theta}}\{\mathbf{d} = \boldsymbol{d}, \boldsymbol{\theta} = \boldsymbol{\theta}\} = \mathbb{P}_{\mathbf{d}|\boldsymbol{\theta}}\{\mathbf{d} = \boldsymbol{d}|\boldsymbol{\theta} = \boldsymbol{\theta}\}\mathbb{P}_{\boldsymbol{\theta}}\{\boldsymbol{\theta} = \boldsymbol{\theta}\} \tag{2.22}$$

Note that since we are now considering an *a priori* distribution on $\boldsymbol{\theta}$ we treat it as a random variable (or vector). As a result, we use the conditioning bar instead of the $\mathbb{P}_{\mathbf{d}\sim D}\{\mathbf{d} = \boldsymbol{d}; \boldsymbol{\theta}\}$ notation used previously for the MLE. The above expression can be used to compute the *a posteriori* distribution of the model parameters given a set of observations $\boldsymbol{d}$, through Bayes rule:

$$\mathbb{P}_{\boldsymbol{\theta}|\mathbf{d}}\{\boldsymbol{\theta} = \boldsymbol{\theta}|\mathbf{d} = \boldsymbol{d}\} = \frac{\mathbb{P}_{\mathbf{d}|\boldsymbol{\theta}}\{\mathbf{d} = \boldsymbol{d}|\boldsymbol{\theta} = \boldsymbol{\theta}\}\mathbb{P}_{\boldsymbol{\theta}}\{\boldsymbol{\theta} = \boldsymbol{\theta}\}}{\mathbb{P}_{\mathbf{d}}\{\mathbf{d} = \boldsymbol{d}\}} \tag{2.23}$$

The MAPE, similarly to the MLE, consists in computing the argument of the maxima with respect to the parameters $\boldsymbol{\theta}$ of the (log-)posterior distribution and is defined as:

$$\hat{\boldsymbol{\theta}}_{\text{MAPE}} \triangleq \underset{\boldsymbol{\theta}}{\text{argmax}} \log \mathbb{P}_{\boldsymbol{\theta}|\mathbf{d}}\{\boldsymbol{\theta} = \boldsymbol{\theta}|\mathbf{d} = \boldsymbol{d}\} = \underset{\boldsymbol{\theta}}{\text{argmax}} \log \frac{\mathbb{P}_{\mathbf{d}|\boldsymbol{\theta}}\{\mathbf{d} = \boldsymbol{d}|\boldsymbol{\theta} = \boldsymbol{\theta}\}\mathbb{P}_{\boldsymbol{\theta}}\{\boldsymbol{\theta} = \boldsymbol{\theta}\}}{\mathbb{P}_{\mathbf{d}}\{\mathbf{d} = \boldsymbol{d}\}} \tag{2.24}$$

As we compute the argument of the maxima with regard to the parameters $\boldsymbol{\theta}$, $\mathbb{P}_{\mathbf{d}}\{\mathbf{d} = \boldsymbol{d}\}$ is a constant. Hence, this expression can be further simplified, making MAPE appear as a regularised version of MLE:

$$\hat{\boldsymbol{\theta}}_{\text{MAPE}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \log \mathbb{P}_{\mathbf{d}|\boldsymbol{\theta}}\{\mathbf{d} = \boldsymbol{d}|\boldsymbol{\theta} = \boldsymbol{\theta}\}\mathbb{P}_{\boldsymbol{\theta}}\{\boldsymbol{\theta} = \boldsymbol{\theta}\} = \underset{\boldsymbol{\theta}}{\text{argmax}} \left(\log \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{d}) + \log \mathbb{P}_{\boldsymbol{\theta}}\{\boldsymbol{\theta} = \boldsymbol{\theta}\}\right) \tag{2.25}$$

The above expression shows MAPE to be equivalent to MLE when the considered prior is uniform. Assuming *independently and identically distributed* (i.i.d.) samples the MAPE can be further decomposed as the following sum:

$$\boxed{\hat{\boldsymbol{\theta}}_{\text{MAPE}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \left(\log \mathbb{P}_{\boldsymbol{\theta}}\{\boldsymbol{\theta} = \boldsymbol{\theta}\} + \sum_{i=1}^{n} \log \mathbb{P}_{\mathrm{d_i}|\boldsymbol{\theta}}\{\mathrm{d_i} = d_i|\boldsymbol{\theta} = \boldsymbol{\theta}\} \right)} \tag{2.26}$$

Supposing we are able to compute or approximate the argmax function with analytical or sampling methods, both MLE and MAPE can be relatively computationally efficient. On the other

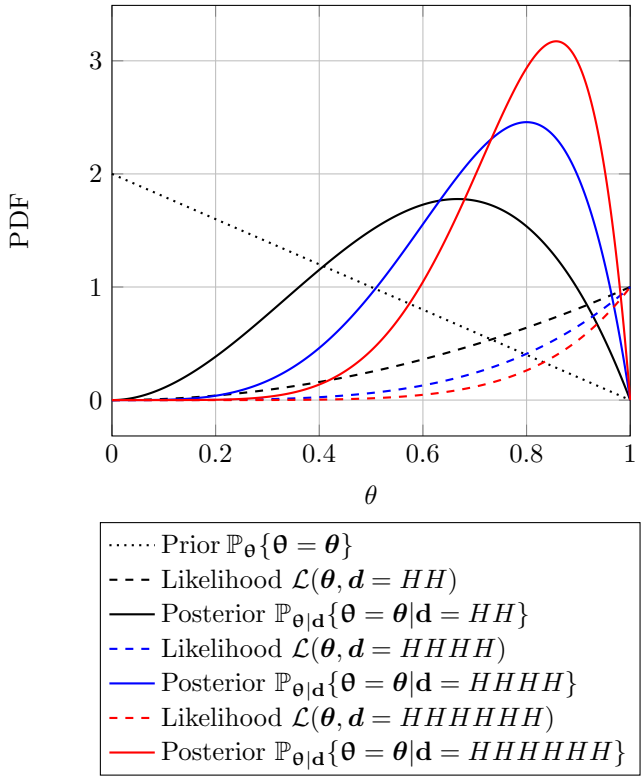


Figure 2.15: Bayesian Posterior Distribution Computation
In this example, a posteriori distributions are computed based on an arbitrary a priori distribution of the Bernoulli model parameter and different observations of sequences of coin flips. The chosen prior distribution indicates that it is more likely that the parameter θ is equal to 0 than to 1. However, the observations seem to contradict this prior distribution since we only observe heads. The prior distribution is therefore updated to 1 when the posterior distribution is calculated. We can see that the more observations we have, the less influence the a priori distribution has on the a posteriori distribution.

hand, they are said to be *point estimators* in the sense that they provide a single set of value for the parameters $\boldsymbol{\theta}$ and not a distribution. This leaves out some information such has an estimate of the uncertainty over the computed parameters. Computing the complete posterior distribution and not only its argmax can be costly as it involves computing it's denominator through an integration (or a sum for discrete variables), potentially of high dimension depending of the dimension of $\boldsymbol{\theta}$. Indeed, the denominator can be formulated as:

$$\mathbb{P}_{\mathbf{d}}\{\mathbf{d} = \boldsymbol{d}\} = \int_{\boldsymbol{\theta}} \mathbb{P}_{\mathbf{d}|\boldsymbol{\theta}}\{\mathbf{d} = \boldsymbol{d}|\boldsymbol{\theta} = \boldsymbol{\theta}\}\mathbb{P}_{\boldsymbol{\theta}}\{\boldsymbol{\theta} = \boldsymbol{\theta}\}d\boldsymbol{\theta} \tag{2.27}$$

Figure 2.15 follows up on the coin flip example of Figure 2.14, but adds the knowledge of an arbitrary prior distribution to compute the *a posteriori* distributions. Here, the prior distribution is a linear function with smaller values of θ being more likely than higher values of θ. The MAPE consists in taking the argument of the maxima of these *a posteriori* distribution. As can be seen by comparing Figures 2.15 and 2.14, for the same observed sequence $\boldsymbol{d} = HHHH$, where the MLE would have selected a parameter $\hat{\boldsymbol{\theta}}_{\text{MLE}} = 1$, the MAPE selects a parameter $\hat{\boldsymbol{\theta}}_{\text{MAPE}} = 0.8$ as a result of the added prior knowledge. It can also be noted that the posterior distribution becomes tighter as the number of observed data increases, thus providing a measure of the uncertainty on the chosen parameter. Furthermore, it can be seen that the impact of the prior distribution on the posterior distribution decreases as the number of observations increases, such that the MAPE converges to the MLE. The *a posteriori* distribution is then supposed to be a better description of the actual data generation process than the *a priori* distribution, as it contains more information. It should also be noted that an *a posteriori* distribution can later become the prior for the calculation of a new *a posteriori* distribution provided that new data are observed.

Structured Probabilistic Graphical Models, Factor Graphs and Associated Inference Algorithms

When many random variables are involved in the representation of a stochastic process, it can become very complex to exhaustively describe the complete joint distribution, *e.g.* in a tabular approach. Indeed, n random variables taking k possible values lead to k^n possible outcomes, which grows exponentially in n. In reality, the relationships between these n variables are usually sparse, in the sense that one variable usually influences only a few others (which, in turn, may influence others) and these joint distributions of considerable dimensions can be represented in a more compact way. The objective of structured probabilistic graphical models is to model these large joint probability distributions as a sparse graph of local relationships describing how random variables interact directly with (hopefully a small number of) other variables.

Among the different types of probabilistic graphical models, this section introduces the notion of undirected *Factor Graph* (FG) and the associated inference algorithms that are of interest for the present manuscript. The idea behind the FG is, as the name implies, to express a global function of many variables, and therefore complex to compute, as a factorisation of many local functions, much easier to compute. This idea is obviously not specific to the calculation of large joint probability distributions but can be used advantageously for such tasks.

Let $g(x_1, \ldots, x_n)$ be a global function of n variables that can be factored as a product of k local functions over subsets of the n variables. A (Forney-style) FG is defined by the following rules:

- Every factor is represented by a unique node.
- Every variable is represented by a unique edge (or half edge) and an edge can connect at most 2 factors.
- A node g_i is connected to an edge x_j if and only if g_i is a function of x_j.

For example, Figure 2.16 provides the FG corresponding to the following function:

$$g(x_1, \ldots, x_5) = g_1(x_1, x_2, x_3) g_2(x_2) g_3(x_3, x_4, x_5) \tag{2.28}$$

where g is a global function factored as a product of local functions g_i working on subsets of all the variables $x_1, \ldots, x_5$.

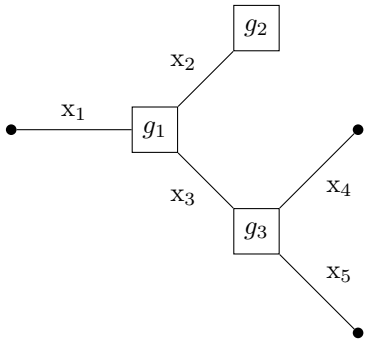


Figure 2.16: An Example of Factor Graph

In some case we might be interested in representing FG where a variable participates in more than two factors. This is in contradiction with the construction rules of a FG described before. To solve that issue one can simply define a new variable and a new factor node whose local function $g_=$ act as a "cloning" function from the original variable to be duplicated to the newly defined variable. For example, lets imagine the following function:

$$g(x_1, \ldots, x_5) = g_1(x_1, x_2, x_3) g_2(x_2, x_3) g_3(x_3, x_4, x_5) \tag{2.29}$$

As one can see, variable x_3 participate in more than two factors thus violating the construction principles of FG enumerated before. One could define an equivalent function, whose FG is depcited in Figure 2.17, as:

$$g'(x_1, \ldots, x_5) = g_1(x_1, x_2, x_3^{(1)}) g_2(x_2, x_3^{(2)}) g_3(x_3^{(3)}, x_4, x_5) g_=(x_3^{(1)}, x_3^{(2)}, x_3^{(3)}) \tag{2.30}$$

The relation described by the equality constraint $g_=$ states that the variables $x_3^{(1)}$ and its two copies $x_3^{(2)}$ and $x_3^{(3)}$ must be equal. Such constraint is formally defined for n variables as:

$$g_=(x^{(1)}, \ldots, x^{(n)}) = \prod_{i=2}^{n} \delta(x^{(1)} - x^{(i)}) \tag{2.31}$$

where $\delta(x)$ express the Kronecker delta function equal to 1 when $x = 0$ and 0 otherwise. The above constraint is valid if and only if all the copied variables $x^{(1)}$ through $x^{(n)}$ are equal.

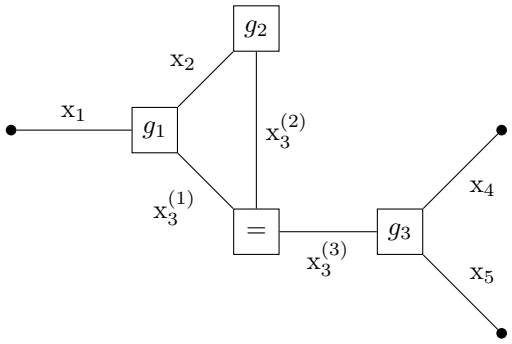


Figure 2.17: An Example of Factor Graph with Equality Constraint Node

Note that the graph from Figure 2.17, contrarily to the one from Figure 2.16, contains a cycle. As will be detailed later on, while inference is exact on *trees*, *i.e.* graph without cycles, it is usually only approximate when such cycles are present.

We are usually interested in answering two types of problems:

- The marginalisation problems where we attempt to compute a function of the form $\bar{g}(x_i) \triangleq \int_{\backsim x_i} g(x_1, \ldots, x_i, \ldots, x_n)$ (or similarly with a sum for discrete variables).

- The maximisation problem where we want to compute a function of the form $\hat{g}(x_i) \triangleq \max_{\backsim x_i} g(x_1, \ldots, x_i, \ldots, x_n)$.

Both problems are in general intractable for large n if not considering specific structure of the global function $g(x_1, \ldots, x_n)$. Yet, the factorisation of the global function $g(x_1, \ldots, x_n)$ in a product of local function can help to reduce that complexity. For example, let's consider the following function and associated FG represented in Figure 2.18:

$$g(x_1, \ldots, x_{11}) = g_3(x_5, x_6, x_7) g_2(x_3, x_4, x_6) g_1(x_1, x_2, x_3) g_4(x_7, x_8, x_9, x_{10}) g_6(x_{10}) g_5(x_9, x_{11}) \tag{2.32}$$

Let's consider the case where we are interested in computing the marginal $\bar{g}(x_7)$. Assuming all x_i are continuous variables, the latter is defined as:

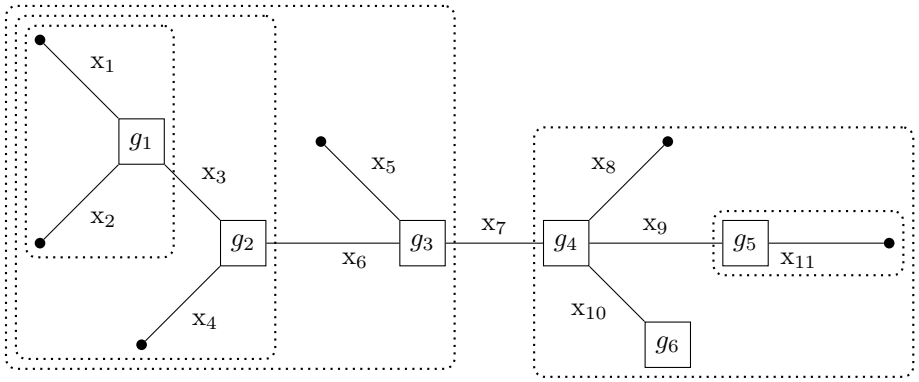


Figure 2.18: Marginalization on a Factor Graph

$$
\begin{aligned}
\bar{g}(x_7) &= \int_{\backsim x_7} g(x_1, \ldots, x_{11}) \\
&= \overbrace{\left[\int_{x_5, x_6} g_3(x_5, x_6, x_7) \overbrace{\left[\int_{x_3, x_4} g_2(x_3, x_4, x_6) \overbrace{\left[\int_{x_1, x_2} g_1(x_1, x_2, x_3) dx_1 dx_2 \right]}^{\mu_{g_1 \to x_3}(x_3)} dx_3 dx_4 \right]}^{\mu_{g_2 \to x_6}(x_6)} dx_5, dx_6 \right]}^{\mu_{g_3 \to x_7}(x_7)} \\
&\quad \overbrace{\left[\int_{x_8, x_9, x_{10}} g_4(x_7, x_8, x_9, x_{10}) g_6(x_{10}) \overbrace{\left[\int_{x_{11}} g_5(x_9, x_{11}) dx_{11} \right]}^{\mu_{g_5 \to x_9}(x_9)} dx_8 dx_9 dx_{10} \right]}^{\mu_{g_4 \to x_7}(x_7)}
\end{aligned}
\tag{2.33}
$$

As evidenced in this example, the advantages of using a factor graph is that it allows to express a global function as a product of local functions. As a consequence, this approach allows to locally eliminate certain variable and simplify a computation such as the marginalisation of the global function. In the above example, one can see that instead of computing a marginal on the global system in $\mathbb{R}^{11}$, the computation has advantageously been recursively decomposed into a nested sums (or integrals) of local marginals of much smaller dimensions. The global integral is simplified to a chained product of simpler integrals by the so called *local elimination property* of FG. In this hierarchical view, each subsystem calculates a summary and passes it on to its parent, which in turn summarises all the data received from all its children and passes it on to its own parent, and so on and so forth. It is in this hierarchical structure that the interest of this type of approach lies. These local summaries, denoted as $\mu_{g_i \to x_j}(x_j)$ in Eq (2.33), can be seen as "messages" that are sent along the edges of the graph to propagate some knowledge. Hence, the common name of *Message Passing Algorithm* (MPA) to refer to many inference algorithms used on FG.

This idea is notably at the heart of the well-known *Sum-Product Algorithm* (SPA), whose name comes explicitly from these alternating sums of products. In this algorithm each node representing a local function $g(x_1, \ldots, x_k)$ receive messages from neighbouring variable edges,

denoted as $\mu_{\mathrm{x_j}\to g}(x_j)$. Then, the node compute the outgoing messages to neighbouring variables $\mathrm{x_i}$ based on the received messages (except the message previously received from the recipient variable) and the local node function. As evidenced in the previous marginalisation example of Eq. (2.33) one can simply define the general Sum-Product rule, that is locally computed at each node of the graph, as:

$$\boxed{\mu_{g\to \mathrm{x}_i}(x_i) = \int\limits_{\backsim x_i} g(x_1,\ldots,x_i,\ldots,x_k)\prod_{\substack{j=1\\ j\neq i}}^{k}\mu_{\mathrm{x_j}\to g}(x_j)} \tag{2.34}$$

Or, in the case of discrete variables:

$$\boxed{\mu_{g\to \mathrm{x}_i}(x_i) = \sum_{\backsim x_i} g(x_1,\ldots,x_i,\ldots,x_k)\prod_{\substack{j=1\\ j\neq i}}^{k}\mu_{\mathrm{x_j}\to g}(x_j)} \tag{2.35}$$

where $\mu_{g\to \mathrm{x}_i}(x_i)$ is the outgoing message from node g toward variable $\mathrm{x_i}$, $g(x_1,\ldots,x_i,\ldots,x_k)$ is the function of k variables of said node and $\mu_{\mathrm{x_j}\to g}(x_j)$ are the incoming message from variables $\mathrm{x_j}$ $(\forall j \in \{1,\ldots,k\}, j\neq i)$ toward the node g.

Note that if the node g is a function of a single variable $\mathrm{x_i}$, then the Sum-Product rule at this specific node simply become:

$$\mu_{g\to \mathrm{x}_i}(x_i) = g(x_i) \tag{2.36}$$

When there is no prior information about a given variable, *e.g.* at the graph initialisation, the associated messages toward neighbouring nodes are by default initialised to a constant function equal to 1 (the neutral element of the product). This, from a probabilistic standpoint, is equivalent to having a uniform prior.

We are able to compute the marginal distribution of a given variable $\mathrm{x_i}$ when the two opposite messages on the corresponding edge are available, *i.e.* they have been computed. The marginal distribution is then defined as follows:

$$\boxed{\bar{g}(x_i) = \mu_{g_a\to \mathrm{x}_i}(x_i).\mu_{g_b\to \mathrm{x}_i}(x_i)} \tag{2.37}$$

where g_a and g_b are the two neighbouring nodes of variable $\mathrm{x_i}$

A very interesting properties of FG and SPA is that the messages exchanged in the graph to compute a given marginal can be stored and reused without additional computations to compute all the other marginals.

If the graph has no cycles the algorithm is simply executed as follow:

- All messages are initialised, based on some observations. When no information is available a uniform prior is usually applied.
- The information is propagated from neighbour to neighbour by applying the sum-product rule to each node where that is possible. As new messages are computed, we can progress in the graph and compute successively new messages. In graphs without cycle, only one pass is necessary and each message is computed only once.
- Finally, for each edge where the backward and forward messages are available, the marginal function can be computed as a product of the two messages as shown in Eq. (2.37).

For cycle-free graphs, the SPA converges toward the correct solution in a finite number of steps depending of the size of the graph. In the case of graphs with cycles, it is necessary to iteratively transmit messages over the cycles of the graphs a certain number of time. This is notably the case of many *Forward Error Correction* (FEC) graph as will be introduced in Chapter 5 of this manuscript. The algorithm is then not guaranteed to converge, notably because of undesired feedback effects, even if it works well in many practical cases. Algorithms similar to the SPA can be used in the more general paradigm of summary propagation algorithms applied on FG.

In this case, other summary rules than the one proposed here are used such as min or max (e.g. Max-Product or Min-Sum algorithms).

FG are a very generic and powerful tool that somehow unify related historical concepts from many separate scientific fields. As an example one can cite the MRF used to describe Ising models in statistical physic [53], Kalman filters [54] or HMM [55] in signal processing, and obviously in *Error Correction Code* (ECC), notably with the work on LDPC codes [56] and associated concepts, as will be the subject of an important part of the present manuscript (see Chapter 5). Obviously, such structure can be used to perform probabilistic inference on joint probability distributions and more precisely joint PDF or joint PMF. Recall that a joint probability distribution can be factored into a product of conditional probability distributions, as presented earlier. For example, let's consider the following joint distribution factored into an arbitrary product of conditional distributions assumed to model a given process:

$$\begin{aligned}\mathbb{P}_{\mathrm{x}_1,\mathrm{x}_2,\mathrm{x}_3,\mathrm{x}_4,\mathrm{x}_5}\{x_1,x_2,x_3,x_4,x_5\} =& \mathbb{P}_{\mathrm{x}_1}\{x_1\}\mathbb{P}_{\mathrm{x}_2}\{x_2\}\mathbb{P}_{\mathrm{x}_3|\mathrm{x}_1,\mathrm{x}_2}\{x_3|x_1,x_2\} \\ & \times \mathbb{P}_{\mathrm{x}_4|\mathrm{x}_3}\{x_4|x_3\}\mathbb{P}_{\mathrm{x}_5|\mathrm{x}_3}\{x_5|x_3\}\end{aligned} \tag{2.38}$$

Such product of conditional distributions can be represented using the formalism of Bayesian networks as shown in Figure 2.19. Bayesian networks allow us to represent the notion of conditional relationships by representing the presence of a conditioning relationship between two variables, symbolized by nodes, with arrows. The direction of the arrows indicates the direction of the conditioning relationship.

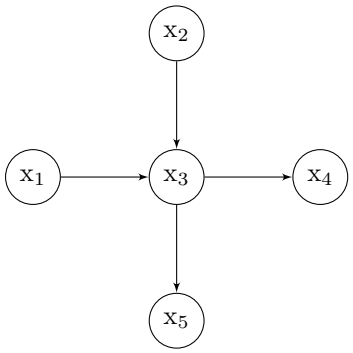


Figure 2.19: Example of a joint probability distribution represented as a Bayesian directed acyclic graph
In a Bayesian graph, random variables are represented by nodes and conditional dependencies are represented by arrows pointing in the direction of the conditioning relationship.

Although conceptually interesting and remarkably simple to read, the practical implementation of inference algorithms on such graphs can be tricky, especially due to their directed nature. Nevertheless, if such a conditioning relationship between two variables exists in one direction, then, regardless of the causality in the interaction between said variables, a correlation "in the other direction" also exists, as shown by Bayes' rule introduced earlier. Therefore, any Bayesian graph can also be represented in the formalism of FG by representing conditional interactions in the form of factors, thus avoiding the problem of directed connections.

Using Bayes rule, Eq. 2.38 can be noted as:

$$\begin{aligned}\mathbb{P}_{\mathrm{x}_1,\mathrm{x}_2,\mathrm{x}_3,\mathrm{x}_4,\mathrm{x}_5}\{x_1,x_2,x_3,x_4,x_5\} =& \mathbb{P}_{\mathrm{x}_1}\{x_1\}\mathbb{P}_{\mathrm{x}_2}\{x_2\}\frac{\mathbb{P}_{\mathrm{x}_3,\mathrm{x}_1,\mathrm{x}_2}\{x_3,x_1,x_2\}}{\mathbb{P}_{\mathrm{x}_1,\mathrm{x}_2}\{x_1,x_2\}} \\ & \times \frac{\mathbb{P}_{\mathrm{x}_4,\mathrm{x}_3}\{x_4,x_3\}}{\mathbb{P}_{\mathrm{x}_3}\{x_3\}}\frac{\mathbb{P}_{\mathrm{x}_5,\mathrm{x}_3}\{x_5,x_3\}}{\mathbb{P}_{\mathrm{x}_3}\{x_3\}}\end{aligned} \tag{2.39}$$

Which is simply a product of factors, represented by the FG of of Figure 2.20, of the form:

$$\mathbb{P}_{\mathrm{x}_1,\mathrm{x}_2,\mathrm{x}_3,\mathrm{x}_4,\mathrm{x}_5}\{x_1,x_2,x_3,x_4,x_5\} = \frac{1}{Z}g_1(x_1)g_2(x_2)g_3(x_1,x_2,x_3)g_4(x_3,x_4)g_5(x_3,x_5) \tag{2.40}$$

where Z is a normalisation constant that ensure that the integral of the probability distribution sums to one.

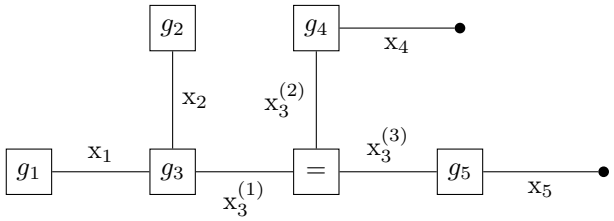


Figure 2.20: Equivalent Factor Graph to the Bayesian newtork of Figure 2.19
Converting conditional probabilities into factors through Bayes' rule allow to express an equivalent factor graph.

The question of the definition of said factors and their efficient computation is of course highly dependant on the specific use-cases considered and thus, out of the scope of this theoretical introduction on probabilistic models. The specificity of the FG models used for ECC and more specifically BP decoders will be introduced more in-depth in the corresponding Chapter 5.

2.3 Neural Networks and Hardware Architectures

The previous sections have described various computational structures, including classical NN models. In this section, we focus on one of the advantages of using these NN structures, namely the associated efficient hardware accelerators.

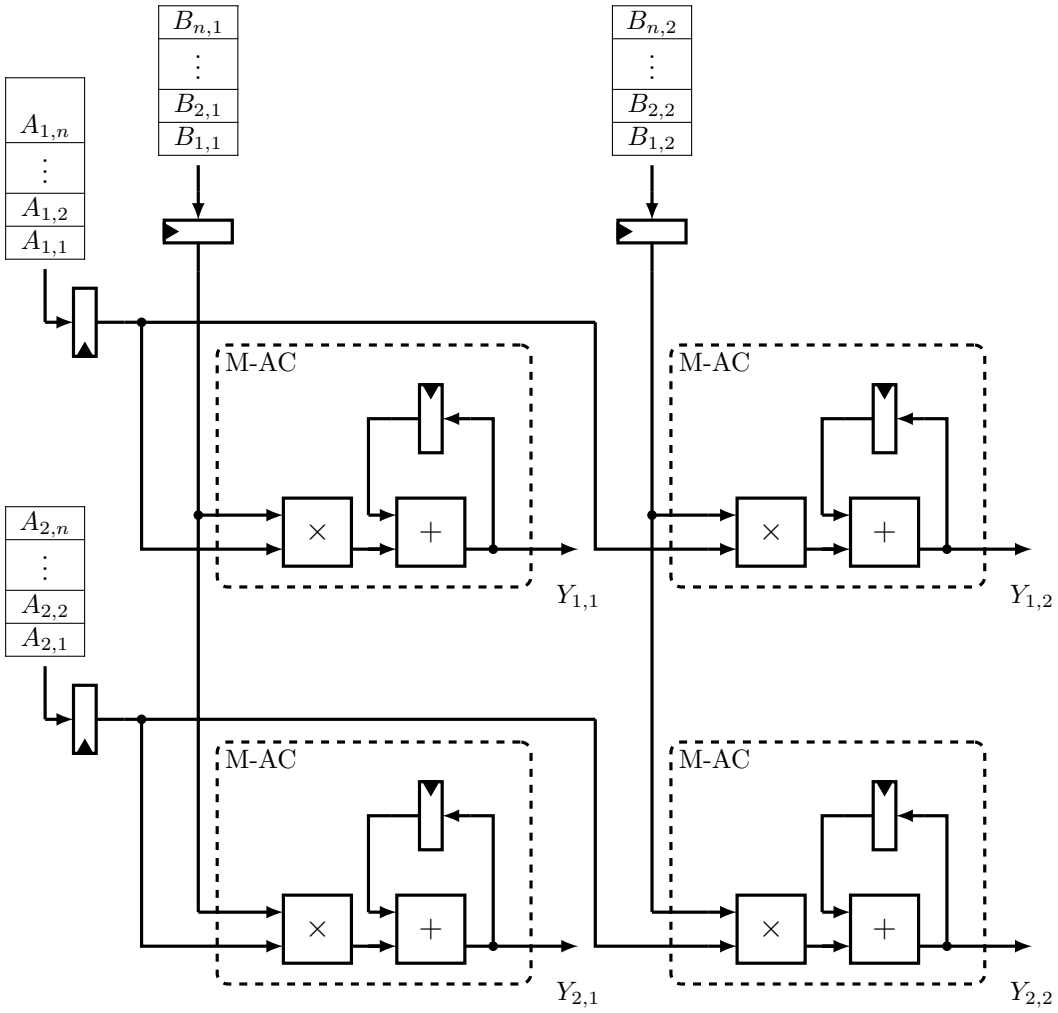
As described in Section 2.2, and despite being universal approximators, NN are simple in essence. Indeed, the expressiveness of NN models lies in their layered architectures and not in complex individual mathematical operations and even the most complex deep-learning instances usually rely on simple operations such as matrix multiplications, additions and a few simple activation functions: maxima, sigmoid, etc. Those simple mathematical operations allow for very efficient and generic hardware implementations [9]. Those implementations are built around a *Matrix Multiply Unit* (MMU) composed of several *Multiply-Accumulate* (M-AC) units, accumulators and *Look-Up Table* (LUT) activation units, matching the mathematical operations carried out by NN. Those co-processors receive both data (tensors) and NN weights from its host and execute different models depending on the task at hand. This genericity allows to efficiently perform different processing tasks on the very same hardware, be it computer visions, speech recognition, radio base-band processing.

As shown in Figure 2.21, a matrix multiplication can be efficiently implemented with specific physical layouts of M-AC units. Similarly, one can implement a dense NN layer in hardware using a so-called stationary weights systolic array architecture, as depicted in Figures 2.22. A systolic architecture is defined by a set of interconnected cells each able to locally perform some simple operation. Information flow directly between cells in a pipe-lined way on short and very fast hard wired interconnections. Contrary to the general-purpose CPU architecture, where operands and instructions are fetched from memory, pushed to an *Algorithmic Logic Unit* (ALU) and the result is then put back in memory, a systolic array architecture chains *Processing Element* (PE) such that one receives the result, and or input data, from its predecessor without using the memory or cache as a buffer.

Such dedicated hardware is now developed by several sources, including Google [57], Intel [58], Nvidia [59], Qualcom [60] and integrated in smartphone by their manufacturers, *e.g.* Apple [61], Huawei [62], etc. While these digital architecture rely on conventional, transistor based circuitry, further research aims to develop physical neuromorphic electronic components to enable a giant leap in terms of energy consumption, latency, number of neurons and inter-connections, speed of execution and distributed memory [63, 64, 65].

2.4 Learning and Optimisation

In the previous sections, various computational models and structures, both deterministic and probabilistic, have been described. The issue of optimising their parameters was only briefly addressed in the case of simple probabilistic models, through the MLE and MAPE techniques.



$$\boldsymbol{Y} = \boldsymbol{AB} \Longleftrightarrow \begin{pmatrix} Y_{1,1} & Y_{1,2} \\ Y_{2,1} & Y_{2,2} \end{pmatrix} = \begin{pmatrix} A_{1,1} & \dots & A_{1,n} \\ A_{2,1} & \dots & A_{2,n} \end{pmatrix} \begin{pmatrix} B_{1,1} & B_{1,2} \\ \vdots & \vdots \\ B_{n,1} & B_{n,2} \end{pmatrix}$$

Figure 2.21: One Possible HW Implementation of (2,n) by (n,2) Matrix Multiplication Unit
An array of 2 by 2 M-AC is used to perform the multiplication of two matrix A and B of size $(2, n)$ and $(n, 2)$ respectively. The column of A are injected on one side of the array while the line of B are injected on the other side. The M-AC are hard wired so as to perform the multiplication of the related terms of matrix A and B and accumulate the results. The inputs are timed using synchronous (clocked) D-Latches, represented by a rectangle with a black triangle clock input.

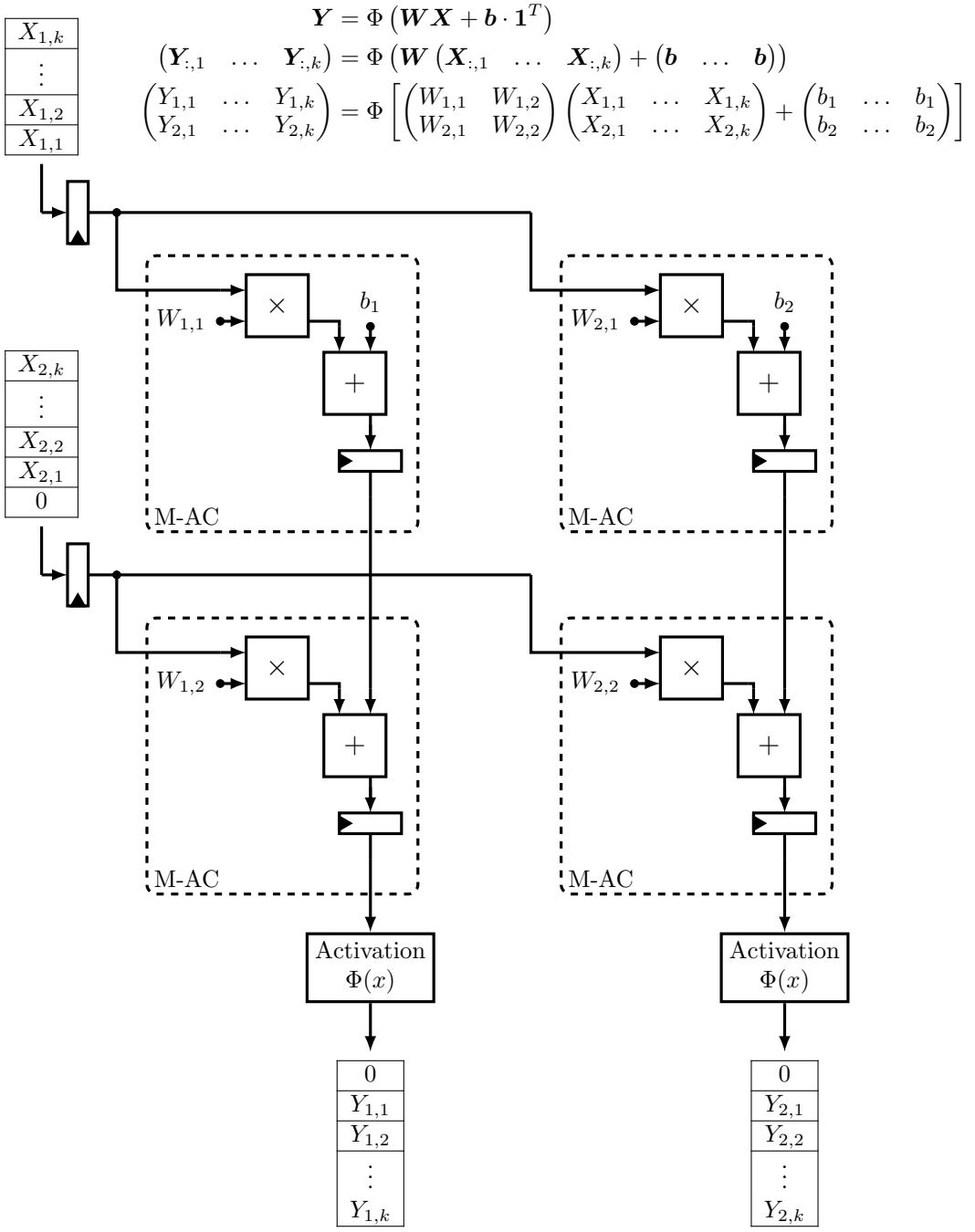


Figure 2.22: One Possible Implementation of a Dense Neural Network Layer as a Systolic Array
A stationary weights systolic array is used to compute the outputs of one layer of NN with two neurons on an input batch of size k (as introduced in Section 2.2.2). Each batch element (*i.e.* column of the input matrix) is provided to the systolic array. Each M-AC compute the multiplication of the corresponding input with its local stationary weight and pass the results onto the next M-AC for results accumulation. The first M-AC in line use the accumulation entry to add the bias of the neuron. The outputs of the systolic array are finaly passed into the chosen activation function, usually implemented as a LUT. The computation flow is timed using synchronous D-Latches.

This section presents in more detail various optimisation techniques and learning algorithms, including the most classical gradient-based methods used to train NN models.

2.4.1 The Main Classes of Learning Problems

The optimisation of a parametric model, also called training or learning, aims at identifying the values of its parameters which allow it to answer a practical problem in the best possible way. Based on a given dataset, this parameter choice is performed regarding an *objective function* often termed *loss function*, which aims at rating the quality of the model work.

ML problems are usually divided into the following three categories:

Supervised Learning: The most classic ML configuration, and probably the easiest to grasp, requires labelled data, used as ground truth, to train the model. For example, images of cats and dogs are labelled individually according to the animal they represent. The model is then used to classify these images. The predicted classes are finally compared to the actual labels of the images to evaluate the quality of the model's predictions and update the model's parameters accordingly. More formally, the goal of such technique is to learn, based on a training set $\mathbb{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{s}$, a mapping $\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta})$ from inputs x_i to known outputs, or labels, $\boldsymbol{y}_i$. This model is trained based on an error metric, or loss function, usually computed as a distance between the true labels $\boldsymbol{y}_i$ and the predicted labels $\hat{\boldsymbol{y}}_i = \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta})$. The ML problem is thus the minimisation problem of the expected loss function, or risk, given the set of model parameters $\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \ell(\boldsymbol{\theta})$. One of the major challenge is to ensure that the mapping learned on the training set is still valid during exploitation of the model on unseen data. We speak of generalisation capabilities of the model.

Unsupervised Learning: Unlike supervised learning, unsupervised methods do not require labelled data. Instead, it relies on intrinsic data structures and is evaluated on its ability to exploit and/or organise the input data in some quantifiable way. These techniques are typically used for clustering methods where one does not know in advance how the data should be labelled, but instead wants to project data with similar properties into a lower sized manifold where they would be close to each other. For example, grouping people based on their buying habits in order to target them with more effective advertising (as questionable as such an approach might be). Unsupervised Learning methods typically include algorithms and models such as *Self Organising Map* (SOM) or Kohonen Map, *k-Nearest Neighbours* (k-NN), *Principal Component Analysis* (PCA), *t-distributed Stochastic Neighbour Embedding* (t-SNE), etc [66, 67].

Reinforcement Learning: Finally, *Reinforcement Learning* (RL) methods are a class of learning methods loosely inspired by the way animals learn, as emphasised for example by *Pavlov*'s famous experiments on animal psychology, and in particular the notion of conditioned reflexes [68]. In the general framework of RL, the model is referred to as an *agent* that interacts with an *environment*. A positive or negative reward is given to the agent to evaluate the impact of its actions (and not the actions themselves as could be done in a supervised framework). The agent's goal is then to find the sequence of actions that give him the highest possible reward when faced with a certain situation in a certain environment [69]. Those techniques include Dynamic Programming methods, *Multi-Armed Bandit* (MAB) [70], Q-learning [71], etc. For example, consider a simple MAB scenario, where the agent has the choice between two dices, one fair dice and one loaded, rolling only sixes. The agent's objective is to maximise the sum of rolls, i.e. rewards, without prior knowledge on the dices, by learning the dice behaviour (exploration phase) and exploiting that knowledge (exploitation), i.e. rolling the dice thought to roll the highest. Thanks to its generality, RL can be used in a variety of domains. Impressive results have recently been obtained using such techniques, for example by defeating the world champion of the very complex game of Go [35].

As far as this manuscript is concerned, supervised and unsupervised frameworks will be mainly targeted, while RL will be left out of the scope of the study. Again, these learning mechanisms go

beyond the sole concept of NN. The latter are purely parametric models on which ML algorithms can be advantageously applied in order to find satisfactory configurations.

2.4.2 Optimisation of Neural Networks

The operation performed by a NN is dictated by its structure and parameters. The Section 2.2 presented different structures of NN. These models were simply described as parametric functions without further explanation as to how their parameters can be set. Although it is sometimes possible to manually set them to perform a given task - as will be explained in Chapter 4 - this approach quickly becomes impractical for large and complex networks. For this reason, ML techniques have been proposed. Indeed, one of the main features of NN models that makes them widely used is that they allow the implementation of very efficient learning algorithms, the most classical of which are presented in this section.

Empirical Risk Minimisation Principle

ML algorithms, generally used to optimise NN models, are different from classical optimisation algorithms in that they attempt to improve the performance of a model on a test set that is not available at the time of training (and may therefore be infeasible). In general, a ML learning algorithm attempts to minimise a surrogate loss function evaluated on a limited training set. The latter is supposed to be representative of the distribution of data in the test set. Therefore, this approach is an indirect form of optimisation. In contrast, in classical optimisation algorithms, the objective is directly to find a solution to a function to be optimised, without using any form of proxy metric.

Typically, in a standard supervised learning setting the ML problem for every sample pair of the training set $\mathbb{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{s}$ of size s, is the optimisation problem of the parameters $\boldsymbol{\theta}$ so as to best fit the output of the parametric model $\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta})$ with the expected true labels $\boldsymbol{y}_i$. The adequacy of the model's prediction with respect to the expected label is evaluated by the loss function, or cost function, which has the form of a scalar distance metric. The individual loss terms are defined for each training sample pair as:

$$l_i = l(\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta}), \boldsymbol{y}_i) \tag{2.41}$$

Ideally, the goal of the optimisation problem is to find the optimal set of model parameters $\boldsymbol{\theta}^\star$ that minimises the expected loss, or risk, $\ell(\boldsymbol{\theta})$, over all instance of the data generating distribution D:

$$\boldsymbol{\theta}^\star = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \ell(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathbb{E}_{(\mathrm{x},\mathrm{y})\sim D}\left\{l_i\right\} \tag{2.42}$$

Unfortunately, the complete data generating distribution is not accessible and the true risk is approximated by the empirical risk $\hat{\ell}(\boldsymbol{\theta})$, defined as the average of the loss terms on the considered training dataset $\mathbb{S}$:

$$\hat{\ell}(\boldsymbol{\theta}) = \frac{1}{s}\sum_{i=1}^{s} l_i \tag{2.43}$$

The training dataset $\mathbb{S}$ is a random sub-sample of size s assumed to be drawn from the same data generating distribution D as any of all the possible test samples. The empirical risk thus is an unbiased estimator of the true risk:

$$\mathbb{E}_{(\mathrm{x},\mathrm{y})\in\mathbb{S}\sim D}\left\{\hat{\ell}(\boldsymbol{\theta})\right\} = \mathbb{E}_{(\mathrm{x},\mathrm{y})\in\mathbb{S}\sim D}\left\{\frac{1}{s}\sum_{i=1}^{s} l_i\right\} = \mathbb{E}_{(\mathrm{x},\mathrm{y})\sim D}\left\{l_i\right\} = \ell(\boldsymbol{\theta}) \tag{2.44}$$

The training of a model based on the minimisation of this average training error is known as the *Empirical Risk Minimisation* (ERM). By the ERM principle, the optimisation algorithm should find the set of parameters $\hat{\boldsymbol{\theta}}$ such that:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \hat{\ell}(\boldsymbol{\theta}) \tag{2.45}$$

$\hat{\boldsymbol{\theta}}$ is usually not equal to $\boldsymbol{\theta}^\star$ although, without considering any other optimisation issues (*e.g.* local minima of the loss), its value is expected to converge to the latter as the dataset size grows.

Loss Functions

As one can imagine, the choice of the loss function plays a very important role in the final performance of the model. Obviously, this choice depends strongly on the use case considered and the nature of the data. Among the most common loss functions are the two prototypical loss functions for regression and classification tasks[6]:

- *Mean Squared Error* (MSE): Typically used for regression problems, the MSE loss computes the squared distance between the model's predictions and targeted labels as:

$$l_{2_i} = (f(\boldsymbol{x}_i; \boldsymbol{\theta}) - y_i)^2 \tag{2.46}$$

- *Binary Cross-Entropy* (BCE): Typically used for classification tasks, the BCE loss increases when the predicted probability of belonging to a certain class diverges from the actual label. It is defined as:

$$l_{\mathrm{BCE}_i} = (1 - y_i)\log(1 - f(\boldsymbol{x}_i; \boldsymbol{\theta})) - y_i \log(f(\boldsymbol{x}_i; \boldsymbol{\theta})) \tag{2.47}$$

Over-fitting, Under-fitting and Regularisation

A performing ML algorithm should be able to both:

- Reduce the training error *i.e.* minimise the empirical risk.
- Ensure that the test error, *i.e.* the error of the model on data not included in the training dataset, is as close as possible to the training error.

These two points are closely related to the concept of over and under fitting and the bias-variance trade-off as illustrated in Figure 2.23. The bias-variance trade-off states that a high-variance model easily reduces the training error, possibly at the cost of a higher test error due to over-fitting. On the contrary, a low variance model reduces the gap between the training error and the test error, at the cost of a higher training error due to under-fitting. Therefore, one should try to find the balance between a high variance/low bias model and a low variance/high bias model.

A regularisation term, in the form of a function of the weights $r(\boldsymbol{\theta})$, is often added to the empirical risk to reduce the model variance, and thus its tendency to over-fit the training data set, leading to the regularised empirical risk:

$$\hat{\ell}_r(\boldsymbol{\theta}) = \frac{1}{s} \sum_{i=1}^{s} l_i + r(\boldsymbol{\theta}) \tag{2.48}$$

By influencing the set of solutions that the model can choose from, the addition of a regulariser is one of many ways to modify the so-called hypothesis space of the model and thus structure the learning process.

When to Halt the Training

One specificity of ML algorithms is that they do not halt upon reaching a local minimum as can be the case *e.g.* with a convex optimisation algorithms. Instead ML algorithm minimises the cost function and halt when reaching some convergence criterion such an early stopping criterion. The latter can be very useful to halt the training just when the model start to overfit the data. This phenomenon can be detected through a method named *cross-validation*, as demonstrated on Figure 2.24.

During a training, the cross-validation consists in monitoring both the loss function evaluated on the training dataset and the loss function evaluated on some validation data, that are not part of the training dataset. While the first metric allows to measure the effective reduction of the training error, the second one allows to measure the gap between the training and test error. When the later increases while the training error continues to decrease, the model is probably starting to overfit the data.

[6] To simplify the description, the loss terms are described for a model with a scalar output. The calculation of the loss function can be extended to vector or matrix valued outputs (*e.g.* when considering a batch), by summing or averaging the individual loss terms over all the outputs.

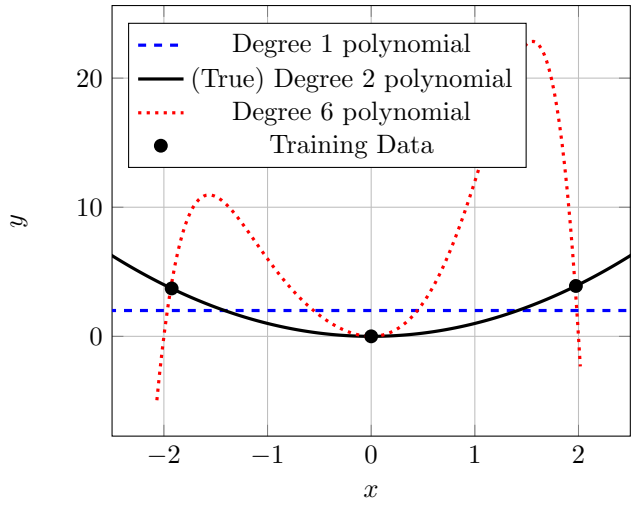


Figure 2.23: The Bias-Variance Trade-off
There are an infinite number of different models that can possibly fit certain training data, but with different biases and variances. In this example, the observed data (●) is sampled from an unknown process, which happens to be quadratic. Three models are used to fit the data. The first model (- - -) is a linear model, which fails to capture the curvature present in the data and is therefore particularly biased. This model is said to under-fit the data. The second model (——) is a quadratic model that is able to well capture the underlying phenomena from the observation of the training data. The model is therefore able to generalize well outside of the training data set. Finally, a third model (·····) based on a polynomial of degree 6, is able to fit the training data perfectly, although its high variance does not capture the underlying phenomena well. The model is said to be over-fitting the data. This example highlights what is called the Bias-Variance trade-off.

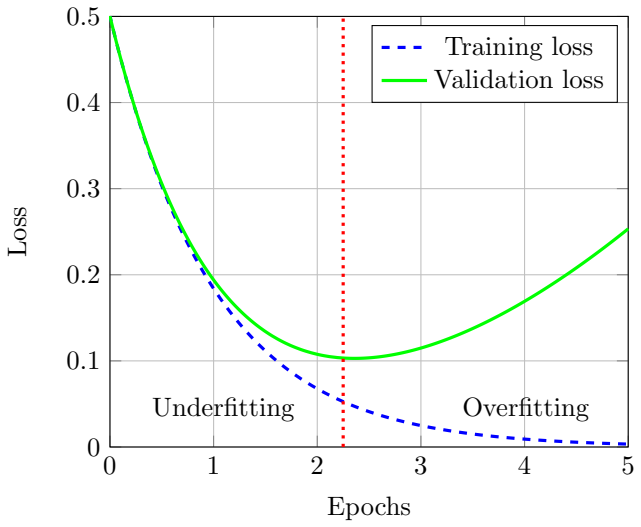


Figure 2.24: Cross-Validation Method
Tracking performance during model learning, both on the training set and on a validation set, is a good way to know the appropriate time to stop the learning procedure. The dotted red line indicates the appropriate time to stop the learning procedure or what is sometimes known as the optimal capacity of the model.

Decomposing the Learning Problem

Depending on the model complexity and the dataset size, it is not always possible to compute the objective function associated to all sample pairs as a single calculation. As the empirical risk usually decomposes as a sum over the training samples, the training dataset can, under certain assumptions, be decomposed into smaller sets. The optimisation algorithm can then update the model based on the estimation of the expected loss on subsets of the complete cost function.

Algorithm working on the complete dataset in a single pass are said to be *deterministic* or *batch* algorithms[7]. When the optimiser works in a sample by sample fashion, the algorithm is said to be *stochastic*. Finally, optimiser working on subsets (of more than one elements) of the entire dataset are referred to as *minibatch* methods. The latter approach is one of the most widely used in ML.

Gradient-Based Optimiser

We have already discussed the "what" - *i.e.* the minimisation of an objective function with respect to a set of training data - but not the "how" - *i.e.* the way to act on the parameters of the model to efficiently reduce this loss function in practice. The optimisation of NN is often performed by GD, or related methods, because of the very interesting properties allowed by the graphical and hierarchical structures of NN, as will be detailed in the next section.

GD is a method that attempts to minimise a function based on the evaluation of its partial derivatives at various points. We first give an illustration of this method in the case of a single-variable scalar function. Let $f(\theta) : \mathbb{R} \to \mathbb{R}$. The partial derivative, or gradient, of the function f with respect to the variable θ, denoted as $\frac{\partial f(\theta)}{\partial \theta}$, represents the slope of the function evaluated at point θ. This slope information is very valuable because it can guide us to the point $\theta^\star$ that minimises the function f. Indeed, imagine that you are blindfolded at the top of a hill, knowing the slope of the hill under your feet is a good way to go back down to the valley (even if one has to be mindful of possible cliffs and trees on the way...).

From a starting point θ_0, the GD algorithm, in its simplest form, iteratively pushes the parameter θ in the opposite direction to that of the gradient to get step by step closer to the minimum of the function. The magnitude of these updates depends not only on the modulus of the gradient, but also on a strictly positive parameter called the *learning rate*, noted here α, which weights the latter (see Algorithm 2.1 and Figure 2.25).

Algorithm 2.1 Gradient Descent (Simplified)

procedure GD($f(x), n_{\text{Steps}}, \alpha, \theta_0$)
  $\theta \leftarrow \theta_0$
  for $i \in \{1, \dots, n_{\text{Steps}}\}$ **do**
    $\theta \leftarrow \theta - \alpha \frac{\partial f(\theta)}{\partial \theta}$
  end for
  Return θ
end procedure

Obviously, this algorithm can easily be extended to the multi-variate case so as to update several parameters simultaneously. Let $f(\boldsymbol{\theta}) : \mathbb{R}^m \to \mathbb{R}$. The gradient is formally defined as the vector of partial derivatives of f with respect to all components θ_i of the parameter vector $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \triangleq \frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_m} \end{pmatrix} \tag{2.49}$$

It should also be kept in mind that, for NN or more generally for parametric functions, the inputs (i.e. from the dataset) of the functions are considered as constants when calculating

[7]The word *batch* can be confusing as the term *batch size* is often used to describe the size of the *minibatch* used for *minibatch* optimisation algorithms.

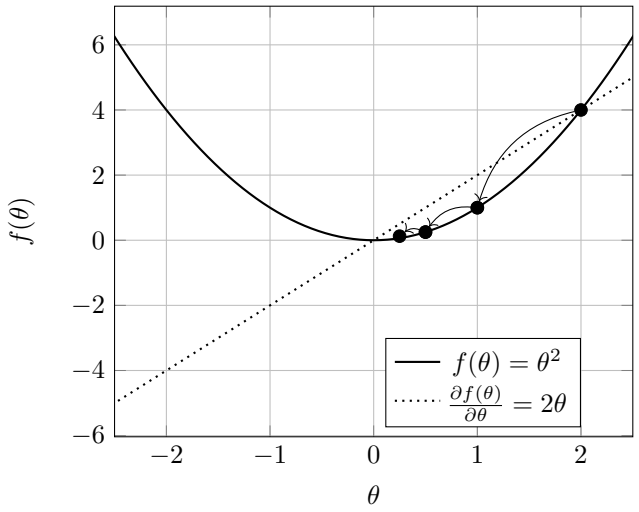


Figure 2.25: Gradient Descent
Starting from the point $\theta_0 = 2$, the GD algorithm, with a LR of 0.25, iteratively update the parameter θ so as to minimize the function f as described in Algorithm 2.1.

Algorithm 2.2 Minibatch Gradient Descent
The model's parametric function is $f(\boldsymbol{x}_i; \boldsymbol{\theta})$ and the loss l. The training set of size s is denoted as $\mathbb{S}$. Minibatch $\mathbb{B}$ of size b are randomly sampled from $\mathbb{S}$ at each of the n_{Steps} of the training. The LR is denoted as α and the inital parameter values as $\boldsymbol{\theta}_0$.

procedure MINIBATCH GD($\mathbb{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{s}, f(x; \boldsymbol{\theta}), n_{\text{Steps}}, b, \alpha, \boldsymbol{\theta}_0$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
  for $i \in \{1, \ldots, n_{\text{Steps}}\}$ **do**
    $\mathbb{S} \leftarrow \text{Shuffle}(\mathbb{S})$
    $\mathbb{B} \leftarrow \mathbb{S}[0 : b]$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \left(\frac{1}{b} \sum_{(\boldsymbol{x}_i, y_i) \in \mathbb{B}} l(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i) \right)$
  end for
  Return $\boldsymbol{\theta}$
end procedure

the partial derivatives with respect to the parameters. Also, the calculation of the empirical risk, including the cost function, the parametric function itself, and this for a specific subset of the training dataset (deterministic, stochastic, mini-batch, etc...), are part of the GD algorithm itself. A more thorough description of the GD algorithm in the multivariate case and including the previous remarks is detailed in Algorithm 2.2.

The attentive reader will have noticed that this method works well in the example of Figure 2.25 thanks to the very favourable conditioning of the considered function. Indeed, the latter is a *strongly convex* function for which GD methods are guaranteed to converge (assuming an appropriate LR is chosen). On the contrary, when considering more complex functions, that present several *critical points*, *i.e* points where the derivative of the function is null, the GD algorithm can get stuck before reaching the global minima. Such a function is exemplified in Figure 2.26. In the proposed example, three situations are described with different initialisation of the GD algorithm leading to drastically different results. As shown on the figure, the initial conditions and the existence of these multiple critical points can lead the algorithm to a solution that performs poorly compared to the optimal one. Many other factors, such as the choice of an excessively large *Learning Rate* (LR), can also have a severe impact on the performance of the model, for example by preventing the stabilisation of the optimisation algorithm.

To overcome these major problems, several variants of the GD algorithm have been developed, *e.g.* [72, 73], including *momentum* techniques. Simply put, momentum techniques are based on the calculation of a moving average of previous gradient values, so as to keep updating the model weights when a critical point, *i.e.* where the gradient is equal to zero, is reached. As indicated by its name, this technique uses the momentum gained on a downward slope to eventually exit a local minima. Such an approach is described in Algorithm 2.3.

Still, it should be noted that, if the GD algorithm should ideally find the global minimum of the function, it is often sufficient to reach a local minimum with performance close to that of the global one. This allows to practically reduce the complexity on very large function with many local minima. This is the idea of *approximate minimisation*.

Many variants of GD have been proposed and an in-depth description of all of them would fall beyond the scope of this introductory section. We refer the interested reader to [1] for a more detailed explanation of these concepts.

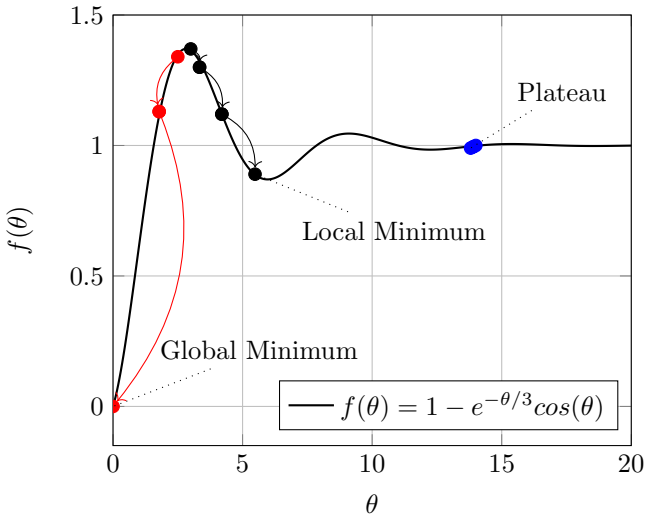


Figure 2.26: Some Challenges of the Gradient Descent - Critical Points & Initial Conditions
Starting from three different points θ_0, with the same LR $\alpha = 5$, the GD algorithm leads to entirely different results. When starting from $\theta_0 = 2.5$ (——), the GD rapidly reach the global minimum of the function $f(\theta)$ defined on $[0, 20]$. When starting from a slightly different point $\theta_0 = 3$ (——), the GD unfortunately fall on the wrong slope and get stuck in a local minimum that performs poorly when compared to the global minimum. Finally, when starting from $\theta_0 = 14$ (——), the slope of the *plateau* being almost null, the GD fails to update the parameter toward the global minimum.

Algorithm 2.3 Minibatch Gradient Descent with Momentum

The model's parametric function is $f(\boldsymbol{x}_i; \boldsymbol{\theta})$ and the loss l. The training set of size s is denoted as $\mathbb{S}$. Minibatch $\mathbb{B}$ of size b are randomly sampled from $\mathbb{S}$ at each of the n_{Steps} of the training. The LR is denoted as α and the inital parameter values as $\boldsymbol{\theta}_0$. At each step, the momentum term is computed as the weighted sum of previous momentum and current gradient values. The β parameter control the balance between the gradient at current step and the momentum information from previous steps. If $\beta = 0$ this algorithm is equivalent to Algorithm 2.2.

procedure MINIBATCH GD WITH MOMENTUM($\mathbb{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{s}, f(x; \boldsymbol{\theta}), n_{\text{Steps}}, b, \alpha, \boldsymbol{\theta}_0$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
  $\boldsymbol{m} \leftarrow 0$
  for $i \in \{1, \dots, n_{\text{Steps}}\}$ **do**
    $\mathbb{S} \leftarrow \text{Shuffle}(\mathbb{S})$
    $\mathbb{B} \leftarrow \mathbb{S}[0 : b]$
    $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} + (1 - \beta) \nabla_{\boldsymbol{\theta}} \left(\frac{1}{b} \sum_{(\boldsymbol{x}_i, y_i) \in \mathbb{B}} l(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i) \right)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{m}$
  end for
  Return $\boldsymbol{\theta}$
end procedure

Gradient Back-Propagation and Automatic Differentiation

One of the reason to use GD to optimise NN models lies in their graph structure that allows to efficiently compute gradient with regard to a very high number of parameters.

As described earlier, a NN can be thought of as a global function defined as the composition of simpler parametric functions called layers. When an input is submitted to the network, the result of the computation performed by the first layer is passed on to the next and so on until the overall function of the network has been computed. Such an input-to-output computation procedure of the NN is commonly called forward propagation. When considering the learning phase of the network, the calculation of the loss function is also conceptually integrated into this forward propagation. As described in the previous section, in order to optimise the network, one seeks to propagate the loss function information to the model parameters in order to update the latter, via the GD method, towards a more capable configuration. Analytically deriving the expression of the gradient of the loss with respect to one of the model parameters is not intrinsically complex, but being able to numerically evaluate said gradient efficiently for all variables of the model, especially over large NN, is a much more complex task. The BPROP algorithm [74] presents an elegant solution to that problem.

We emphasise that GD - *i.e.* the optimisation algorithm that updates the model parameters based on the loss gradient - should not be confused with BPROP - *i.e.* the algorithm that efficiently computes said gradient so that GD can be applied. Furthermore, just as GD is a very generic algorithm, it should be noted that BPROP is not specific to NN but to a wide class of graphical models.

Figure 2.27 describes, as an example, the detailed forward pass computational graph of a two-layer NN in a standard supervised learning setting. We will now describe how the BPROP algorithm can be applied to efficiently compute the gradient of the loss with respect to the model parameters. BPROP algorithm is based on a very common rule of calculus known as the *chain rule of derivatives*. This rule states that the derivative of a global function made of a composition of functions is equal to the product of their derivatives. For example let f,g and h be three real functions. Let $y = f(x)$, $z = g(f(x)) = g(y)$ and $w = h(g(f(x)) = h(g(y)) = h(z)$. The computation of the derivative of the output w with regard to the input x can be simplified to the following product:

$$\frac{\partial w}{\partial x} = \frac{\partial w}{\partial z} \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \tag{2.50}$$

Figure 2.28 shows a very simple application example of the chain rule to the above composition of functions. This figure also shows that while the local derivatives can be computed during the forward pass through each of the nodes (although this is just one possible implementation), the forward pass must be finished to compute all of the chained derivatives.

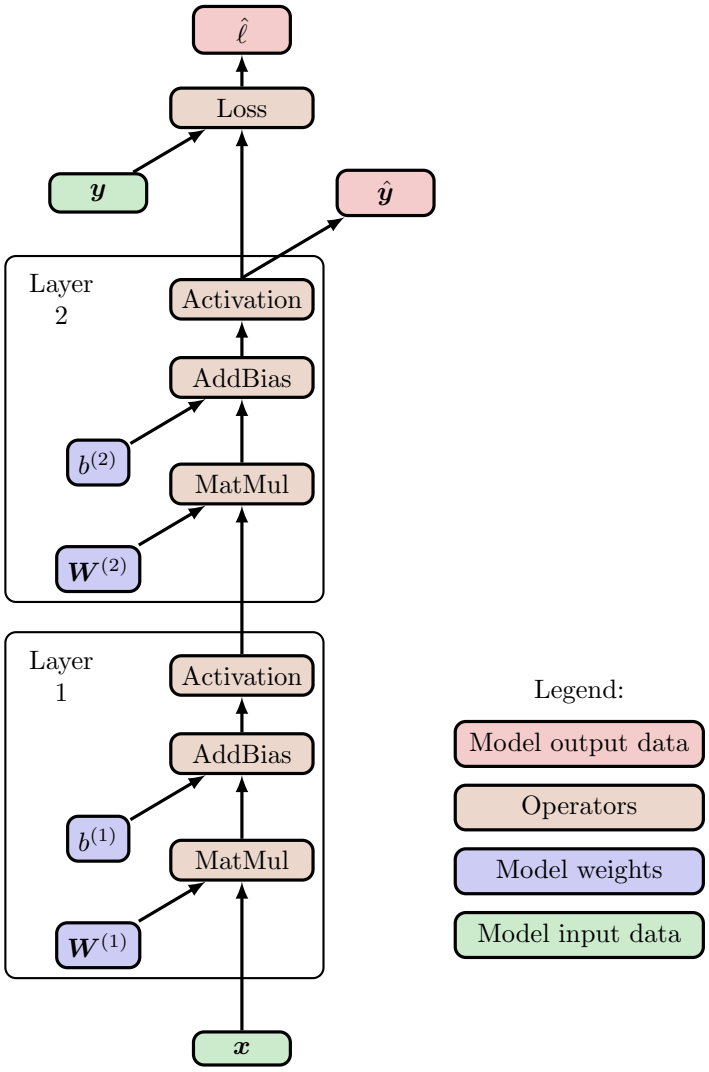


Figure 2.27: Computational Graph of a Two-Layer Neural Network
Representation of the forward pass computation graph of a two-layer neural network in a classical supervised learning context.

Obviously, there is nothing to prevent the use of the chain-rule on non-scalar variables. Instead of multiplying the partial derivatives, one must instead multiply the Jacobian matrices of each vector function. The Jacobian matrix is the matrix containing all of the partial derivatives of a vector function. For example, let $\boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$ and $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$. The Jacobian matrix is defined as:

$$\boldsymbol{J}_{\boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x}) = \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{pmatrix} \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla_{\boldsymbol{x}} y_1^T \\ \vdots \\ \nabla_{\boldsymbol{x}} y_m^T \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \tag{2.51}$$

The idea behind BPROP algorithm is to apply this chain rule to computational graphs in a clever way so that to avoid as much as possible duplicate derivative computations. Indeed, the back-ward computational graph can be seen as a tree where the calculations made by a parent node can be reused by children nodes to reduce the computational load. Figure 2.29 displays the complete computational graph, including the forward pass, the backward pass and the parameters update, of the two-layer NN described as an example in Figure 2.27. As can be seen, while the local derivatives of each of the functions can be computed during the forward pass in each operator, the backward pass can be performed intelligently by using the BPROP

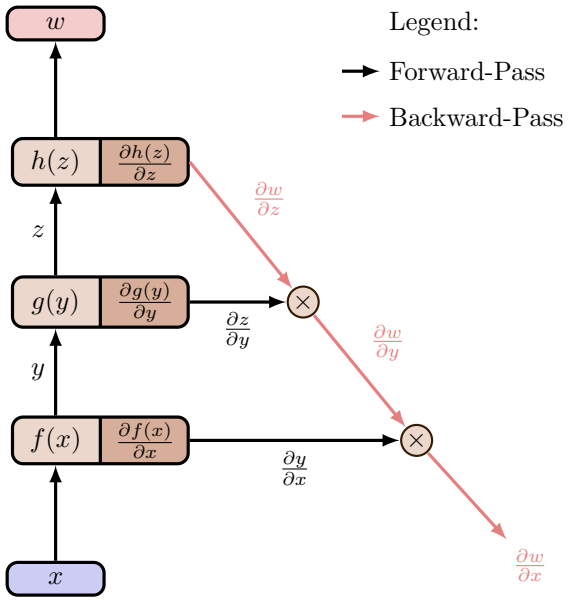


Figure 2.28: Chain Rule of Derivatives & Back-Propagation of Gradients
The chain rule of derivatives can easily be applied to calculate the derivative of a composition of functions as the product of their respective derivatives.

algorithm to combine these local derivatives to compute the gradient of the loss function with respect to the model parameters. As shown in the figure, instead of calculating each of the partial derivatives of the loss gradient starting from zero, many calculations can be reused thanks to the chain rule of derivatives and the graphical structure of the model, so as to efficiently calculate this gradient. Such a modular structure is also helpful to design and integrate new operators by simply defining their local derivatives and relying on automatic differentiation to build the global gradient back-propagation graph. Furthermore, in addition to showing the distinct roles of BPROP and GD, it is also interesting to note that all steps in the optimisation process can be represented in a single deterministic directed graph, including the optimiser, which allows for efficient implementations.

Finally, these descriptions emphasise the need for differentiable operators that enable the computation of all the partial derivatives. Should a single non-differentiable operator lie along the BPROP trajectory, all the parameters upstream of this non-differentiability cannot be updated. While these analytical considerations are paramount, other numerical considerations must also be taken into account. Indeed, and as previously mentioned in Section 2.2.3, some operators are not necessarily suitable to be chained several times from a gradient perspective. For example, some activation functions may display a very small gradient so that the use of the chain rule on a composition of these activation functions may lead to a contraction of the loss gradient with respect to the upstream parameters. This is known as *vanishing gradient*. The opposite phenomenon can also occur when gradients are numerically ill-conditioned such that they build up exponentially while applying the chain rule in what is commonly referred to as an *exploding gradient*.

2.5 Conclusion of the Chapter

The present manuscript looks at how AI and ML algorithms could be used at the PHY layer of future 6G networks. In this introductory chapter on AI and ML techniques, we have provided a description of all the tools that will be necessary for the full understanding of this manuscript. After a brief introduction on the history of AI and computer science, some classical NN structures have been detailed. Dense and convolutional layers have been described and will be mainly used in the Chapter 4. RNN and probabilistic graphical models such as FG were also presented and will be mainly used in Chapter 5. Next, a short section introduced hardware structures dedicated

to the execution of NN. Finally, an extensive section was devoted to describing the basics of NN optimisation, including the classical GD and BPROP algorithms.

Now that the relevant AI and ML techniques have been presented, the next chapter focuses on the subject of interest in this work - *i.e.* the PHY layer - on which we wish to apply these techniques.

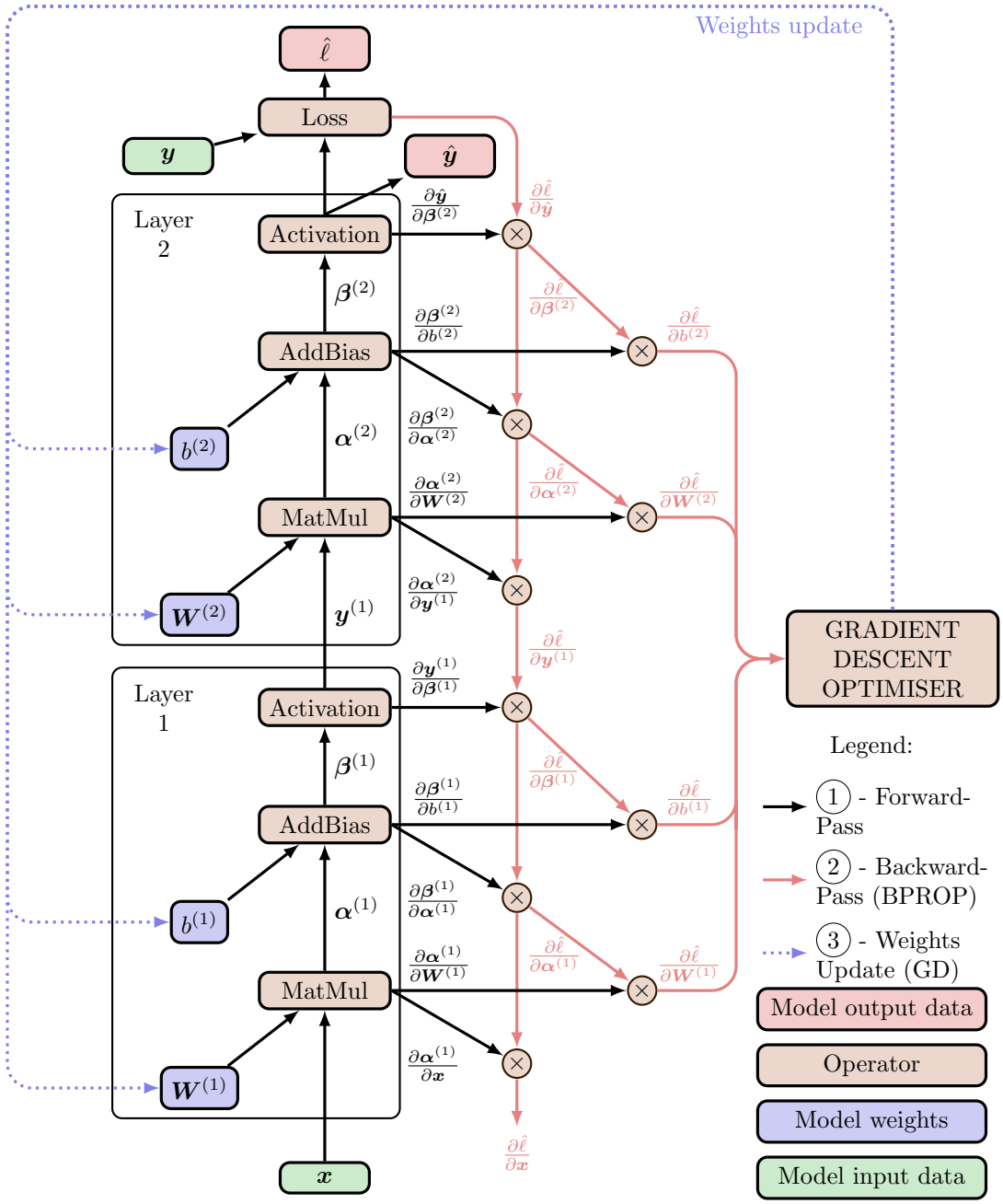


Figure 2.29: Complete Optimisation Graph of a Two-Layer Neural Network
This figure shows the complete optimisation graph of a simple neural network. The model is first executed on input data during the "forward-pass". The BPROP algorithm is then applied, during the so-called "backward-pass", to efficiently compute the gradient of the loss with respect to the variables of interest. Finally, the GD optimiser is used to update the model parameters according to the gradient thus obtained. It is particularly interesting to note that while the NN structures, presented in Section 2.2, were described as modular compositions of individually trivial operations, the same is true for the operations used for model training in the BPROP framework, as emphasised in this figure.

CHAPTER 3

ARTIFICIAL INTELLIGENCE AT THE PHYSICAL LAYER

Contents

3.1 A Brief Introduction to Digital Communication Systems and Physical Layer 50
3.1.1 General Role and Architecture of the Physical Layer in a Digital Communication System 50
3.1.2 Source Coding 51
3.1.3 Channel Coding 53
3.1.4 Line Encoding and Modulation 55
3.1.5 Transmission Medium 57
3.1.6 Detection, Synchronisation and Equalisation 59
3.1.7 Demodulation and Decoding 61
3.1.8 Specific Considerations in the Present Study 63
3.2 Artificial Intelligence at the Physical Layer - A Survey 63
3.2.1 An Historical Perspective on Artificial Intelligence at the Physical Layer 63
3.2.2 Approaches to the Integration of Artificial Intelligence at the Physical Layer 64
Functional Approaches: An Incremental Evolution 64
End-to-End Designs: A Paradigm Shift 66
The Interpretability of the Models 67
Research Axes in Artificial Intelligence for the Physical Layer 67
Integration in Existing 5G and Future 6G Networks 68
3.2.3 Expected Gain and Opportunities 69
Complexity and Performance in Real-Life Conditions 69
Efficient Uniform and Cost Effective Hardware Architecture 69
3.2.4 Challenges 71
Performance Guarantees and Bounds 71
Generalisation of Learning 71
Curse of Dimensionality 71
Security Concerns and Trustworthiness 71
Distributed Learning 71
Interoperability of AI-Based Networks 72
3.3 Conclusion of the Chapter 72

Foreword

As a result of the success of AI/ML in areas such as CV or NLP, the availability of increasingly powerful computing resources such as GPU or *Tensor Processing Unit* (TPU), and the development of widely available software libraries, AI/ML is now being used in an increasing number of domains. Among these new use cases is the application of AI/ML algorithms for digital signal processing at the PHY layer of communication systems. In the preceding chapter, we have discussed general concepts of ML, NN and probabilistic reasoning. In this chapter, we focus on the use case of interest to this work, *i.e.* the PHY layer. As a background, we start by providing the reader with a brief introduction to digital communication systems, in order to highlight the specific role and place of the PHY layer within a communication network, at least as we see it in the present work. Then, a survey on the use of AI at the PHY layer is provided to describe the main associated approaches, challenges and opportunities.

3.1 A Brief Introduction to Digital Communication Systems and Physical Layer

In this section, the role of the PHY layer of a communication system is described. An outline of the main functional blocks of the PHY layer of a digital communication system, as we conceive it in this work, is provided to define the scope of the study. The descriptions provided for each of the blocks are linked to the corresponding contribution chapters and their specific constraints within the context of IoT scenarios are specified.

3.1.1 General Role and Architecture of the Physical Layer in a Digital Communication System

The PHY layer, also called air or radio interface[1] in the case of wireless systems, constitutes one of the lowest layer of a communication system. Its role, from an information theory perspective, is to convert the raw binary digital signal, containing the information to be communicated, into a suitable form for an efficient and reliable transmission over physical channels connecting network nodes. Following, this definition, Figure 3.1 display the main signal processing blocks of a digital communication system. This layer usually includes signal processing operations such as FEC, modulations, filtering operations and, at the receiver side, channel equalisation and decoding processes.

Obviously, the use-case defines the specificity and technical constraints to which the communication system under consideration is subject. For example, a typical smart city IoT use-case could be the transmission of small datagrams from a massive number of low-power sensors, in an urban propagation environment, toward a few base stations. The high number and density of devices imposes additional constraints in terms of QoS, available power, *Total Cost of Ownership* (TCO) of the infrastructure and devices, and the resulting limited complexity of the latter. The design choices for the signal processing functional blocks of the communication chain must take all these constraints into account.

Within the frame of this work we are interested in the digital processing of the signal carried out in *base-band*, i.e. before the analogue conversion and transposition of the signal in carrier frequency at the level of the *Radio Frequencies* (RF) head of the system (or just after its transposition in base-band, in the case of the receiver). Hereafter, and following the sequential diagram of Figure 3.1, we provide a description of the commonly accepted view of the main signal processing blocks of the PHY layer of a digital communication system, or at least a description corresponding to the one adopted in our work.

[1] The terms physical layer, air interface or radio interface might be used interchangeably throughout the present manuscript.

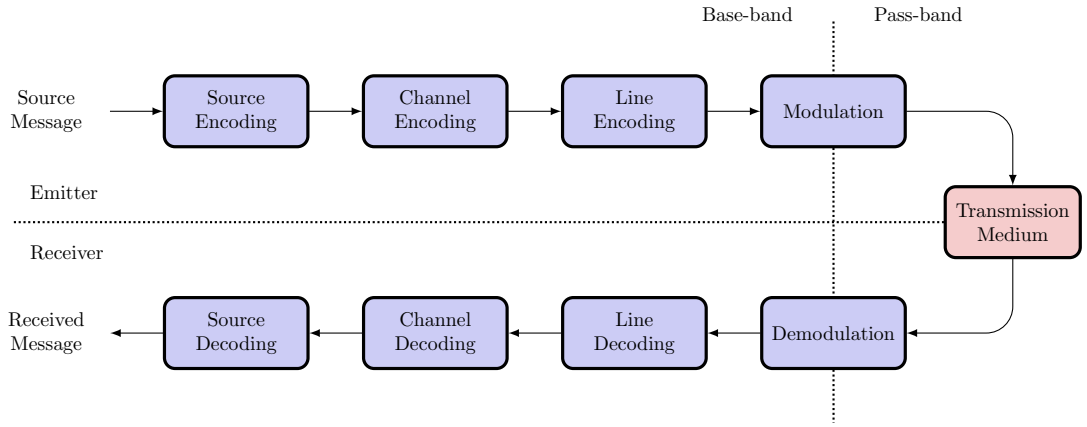


Figure 3.1: Simplified Digital Communication System
This schematic view of the PHY layer of a digital communication system highlights, from an information theory perspective, the main signal processing blocks used to communicate as effectively as possible over a physical transmission channel.

3.1.2 Source Coding

Source coding or *data compression* consists in extracting the useful information from the source in order to transmit only the necessary data and thus increase the efficiency of the communication by reducing the effective bit rate. For example, an image may contain a lot of redundant information, such as when a large portion of the image depicts a blue sky with the same colour on many adjacent pixels. In such a situation, a source coder could schematically pass the areas of the image where the colour is the same, instead of transmitting all the pixels in the raw image. When some information (hopefully not too much) is lost during the source coding process, it is called *lossy compression*. On the contrary, when the compression process is completely reversible, it is called *lossless compression*.

If we consider that the different messages that can be transmitted by a communication system belong to a discrete set, for example 0's and 1's, the efficiency of the communication is directly linked to the statistics of the messages. Let us suppose, in the extreme case, a communication system which always and only transmits 0's. Then the utility of the transmission is questionable since the receiver knows, even before receiving the message, what will be the content of the transmission. Such an idea is described by the notion of *entropy*. In information theory, the entropy of a random source x is the average *information* level (in other words, its uncertainty level) of its possible draws, defined as follows:

$$H_{\mathrm{x}\sim X}\left\{\mathrm{x}\right\} = -\sum_{x\in\Omega_{\mathrm{x}}} p_{\mathrm{x}}(x)\log_b p_{\mathrm{x}}(x) = -\mathbb{E}_{\mathrm{x}\sim X}\left\{\log_b p_{\mathrm{x}}(\mathrm{x})\right\} \tag{3.1}$$

where $p_{\mathrm{x}}(x)$ is the probability that the source x takes the discrete value x and $\log_b$ is the logarithm base b. According to the base of the logarithm, the unit of the entropy will be different: from *bits* for base 2 logarithm to *hartleys* for base 10 logarithm and *nats* for natural logarithm.

Figure 3.2 shows the entropy (in bits) of a binary information source as a function of the probability that a sample of the source is equal to 0. From this figure, we can see that the information content of the source is maximal when both possible outcomes of the binary source are equally likely, *i.e.* the uncertainty of the source is maximal. From the above considerations, we can summarise the role of source coding as maximising source entropy.

An example of lossless source coding is the famous Huffman entropic coding [75]. Huffman coding aims at associating the least probable source message with the longest *code-words* and the most probable one with the shortest code-words so as to maximise the transmission efficiency. Huffman algorithm is a two steps algorithm. At first, the algorithm construct a tree where the least probable message are iteratively grouped two by two and ordered by descending order of probability. When the tree is constructed, the algorithm runs through it the reverse way and append a data symbol to code-words each time a branch is taken. By doing so an *instantaneous*

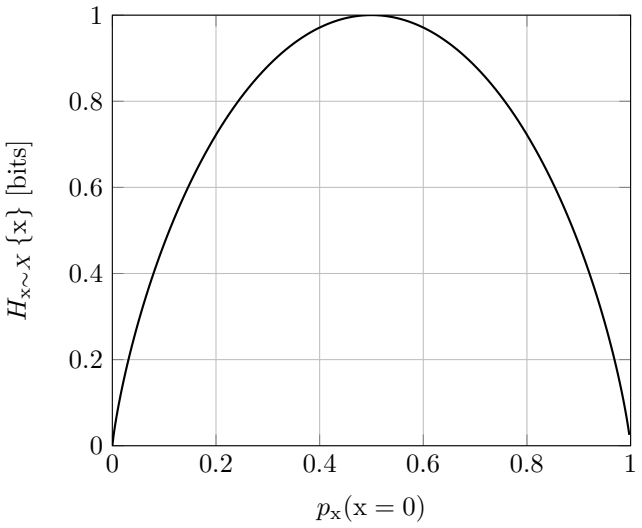


Figure 3.2: Entropy of a Binary Source
The entropy of the source is maximal when it is just as likely that the random variable x takes the value 0 or 1. The uncertainty about the message expected by the receiver of the transmission, and thus the utility of the latter, is then maximal.

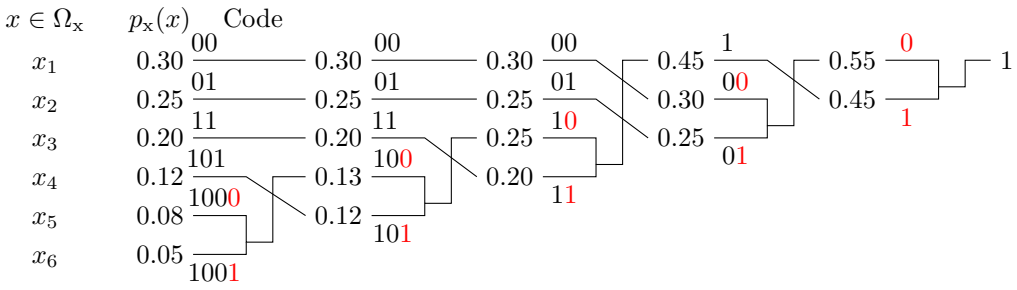


Figure 3.3: Huffman Entropic Coding
In this example, the Huffman tree is constructed from left to right starting with 6 possible symbols $x \in \Omega_{\mathrm{x}}$ and their respective probability of occurrence $p_{\mathrm{x}}(x)$, ordered by decreasing value. The probabilities of the least likely symbols are successively grouped in pairs until a single possible event with probability 1 is reached. Then, from right to left, the coded symbols are constructed. Each time a branch is taken, a 0 or a 1 (depending on whether it is the lower or upper path) is added to the codeword of the current branch, until the whole tree has been covered. With this procedure, the least likely source symbols are associated with the longest coded representation, thus increasing the entropy of the coded source. In this example, the source entropy is initially 0.77bits/binary symbols (or 2.36bits/symbol) while the coded source entropy is increased to 0.97bits/binary symbol.

code is constructed - *i.e.* a code in which each code-word can be decoded as soon as it is received - where the most likely source symbols are mapped to short coded representation so as to maximise the entropy of the coded source. Figure 3.3 shows how such a coding tree is constructed for an initial source ensemble of 6 symbols to be coded using binary symbols.

Owing to its fundamental role in any digital communication system, source coding is briefly developed here but is not further elaborated in the contributions of this thesis.

3.1.3 Channel Coding

Now that we have described how the information content of a source can be compressed so as to transmit only the strictly necessary data and thus increase the efficiency of the communication, we look at the mechanisms that make the latter robust against transmission impairments. Indeed, during a transmission over a noisy channel, the noise and all sort of physical impairments can cause transmission errors such that certain symbols are received with an erroneous value (see Section 3.1.5 for some details on the effects of the transmission medium). The role of *channel coding* is to enable the detection and eventually correction of such transmission errors to make the communication more reliable.

Let $\mathbb{S} \in \mathbb{F}_p^k$ be the set of messages of k information symbols from the finite field of p elements to be transmitted over a noisy channel. For binary symbols $p = 2$ and the size of $\mathbb{S}$ is usually chosen as a finite power of 2 such that $\text{Card}(\mathbb{S}) = 2^k$. When transmitting messages from $\mathbb{S}$ over a noisy channel, it is advisable to implement error detection/correction mechanisms. To be able to detect errors it is necessary to introduce redundancy in the transmitted message such that each of the message of size k from $\mathbb{S}$ is associated to a unique message of size n, with $n > k$, through the application $f : \mathbb{F}_p^k \to \mathbb{F}_p^n$. A code is the mapping function f and the sub-set $\mathbb{V} = f(\mathbb{S}) \in \mathbb{F}_p^n$ image of $\mathbb{S}$ through the application f. The application f must be injective such that all information words are associated with different code-words and the code is decipherable. A code with code-words of size n constructed from information words of size k is known as a $\mathcal{C}(n, k)$ code. The rate of the code is the ratio $r = k/n \in]0, 1[$ describing the part of each code-word used to convey meaningfull data (in contrast to code redundancy).

When receiving a message, the receiver check if it is a code-word of $\mathcal{C}(n, k)$. If so, it assumes that no transmission error occurred (although this is not necessarily true if the erroneous word happens to be a valid code-word). If not, the receiver can request a re-transmission, provided that a feedback channel exists. Such a procedure is commonly referred to as *Automatic Repeat Request* (ARQ). In such a scenario, the code is only used as an *error detection code*. To avoid these costly re-transmissions, the code can also be used to correct transmission errors. It is then referred to as a FEC code. In such a scenario, the receiver needs to choose a code-word $\hat{\boldsymbol{c}} \in \mathbb{V}$ as close as possible to the received code-word $\boldsymbol{y}$ so as to maximise the *a posteriori* probability:

$$\hat{\boldsymbol{c}} = \underset{\boldsymbol{c}}{\text{argmax}}\, \mathbb{P}_{\mathbf{c}|\mathbf{y}} \{\boldsymbol{c}|\boldsymbol{y}\} \tag{3.2}$$

When the prior distribution over the possible code-word is uniform, *i.e.* all code-words are equally likely, the MAPE is equivalent to the MLE:

$$\underset{\boldsymbol{c}}{\text{argmax}}\, \mathbb{P}_{\mathbf{y}|\mathbf{c}} \{\boldsymbol{y}|\boldsymbol{c}\} = \underset{\boldsymbol{c}}{\text{argmax}}\, \frac{\mathbb{P}_{\mathbf{c}|\mathbf{y}} \{\boldsymbol{c}|\boldsymbol{y}\} \mathbb{P}_{\mathbf{y}} \{\boldsymbol{y}\}}{\mathbb{P}_{\mathbf{c}} \{\boldsymbol{c}\}} = \underset{\boldsymbol{c}}{\text{argmax}}\, \mathbb{P}_{\mathbf{c}|\mathbf{y}} \{\boldsymbol{c}|\boldsymbol{y}\} \tag{3.3}$$

where the probability to receive code-word $\boldsymbol{y}$ knowing that the true code-word is $\boldsymbol{c}$, $\mathbb{P}_{\mathbf{y}|\mathbf{c}} \{\boldsymbol{y}|\boldsymbol{c}\}$, is simply the transmission channel conditional probability distribution.

Obviously, mixed strategy with both error detection and correction approaches can be defined such that words within the correction capabilities of the code are corrected at the receiver while a re-transmission is requested for code-words with too many errors. An example of such an approach are the *Hybrid Automatic Repeat Request* (HARQ) methods [76].

Hamming distance is defined as the number of positions between two sequences of same size where the symbols are different. The minimum Hamming distance of a code d_{min} is the minimum pairwise distance among all possible code-words pairs, *i.e.* the distance between the two closest code-words of the code. The error detection and correction capabilities of an ECC are directly related to this characteristic. A code is said to be *perfect* when the spheres of radius t centred on

the code-words form a partition of the code vector space $\mathbb{F}_p^n$. In summary, the code vector space is entirely and optimally packed with closed spheres of radius t. A perfect code is an optimal code for the chosen (n, k) parameters. Few linear codes are perfect such that a less constraining class of codes, known as *Maximum Distance Separable* (MDS) codes, is often considered in the research of performing coding schemes. While all linear codes respect the following equation:

$$n - k \geq d_{min} - 1 \tag{3.4}$$

Few reach the bound, known as the *singleton bound*, such that $n - k = d_{min} - 1$. Such codes are referred to as MDS and are optimal for the chosen (n, k) parameters, *i.e.* if a code of dimension n and length k is MDS then there are no other codes of the same length and dimension with a larger minimum distance.

FEC are often divided into convolutional codes and block codes families. Convolutional codes constructs each of the redundancy symbols via the sliding application of a logic function to the data stream. Block codes divides the data stream in blocks of fixed length k and construct code-words of size n through the application of an application $f : \mathbb{F}_p^k \to \mathbb{F}_p^n$. Among the block codes, *Linear Block Codes* (LBC) further impose the constraint that the application must be a linear map. In this manuscript, only LBC are considered.

A very simple example of LBC are repetition codes which repeat each transmitted bit of information a certain number of times to allow for error correction/detection mechanisms at the receiver. Let $\mathcal{C}(3, 1)$ be a three repetitions code with the following mapping:

$$\begin{cases} \boldsymbol{c}_1 = f(0) = (0, 0, 0) \\ \boldsymbol{c}_2 = f(1) = (1, 1, 1) \end{cases} \tag{3.5}$$

The code vector space can be represented as a cube where only the $(0, 0, 0)$ and $(1, 1, 1)$ code-words effectively belong to the $\mathcal{C}(3, 1)$ code as shown on Figure 3.4. As evidenced by the figure, the code has correction capacity of up to 1 error or a detection capacity of up to 2 errors. Using MLE decoding, the errors are simply corrected based on the Table 3.1.

The MLE decoder cannot usually be applied in such a naive way as it would require to compare the received code-word with all of the 2^k possible code-words of the code which get rapidly computationally intensive, or even intractable, when increasing k. This highlights the trade-off between adding redundancy to reduce the number of transmission errors and increasing the complexity and energy consumption of the system, hence the need for well constructed (de)coding scheme with balanced complexity and performance.

The stringent constraints of IoT systems directly impact the choice of ECC mechanisms. Indeed, these scenarios usually imply both challenging transmission conditions requiring error correction and reduced device complexity preventing from using complex, but performing, FEC mechanisms. Furthermore, while using longer codes (typically thousand or ten of thousands of

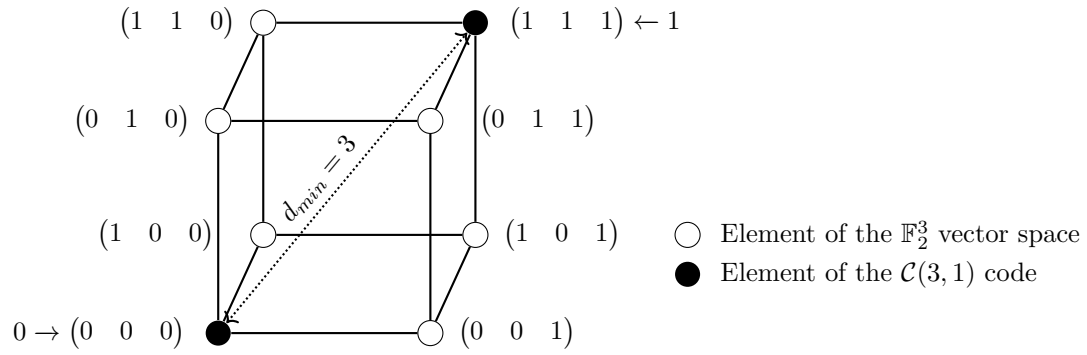


Figure 3.4: Illustration of the $\mathcal{C}(3, 1)$ Repetition Code
All of the 2^3 possible words of size 3 are represented as vertices on a three dimensional cube. The 2^1 code-words of the $\mathcal{C}(3, 1)$ repetition code are represented using black-filled circle. When a transmission error occurs the decoding strategy is simply to find which of the two $(0 \quad 0 \quad 0)$ or $(1 \quad 1 \quad 1)$ code-words is the closest to the received one.

Received code-word	Corrected code-word
$(0\ 0\ 0)$	$(0\ 0\ 0)$
$(0\ 0\ 1)$	$(0\ 0\ 0)$
$(0\ 1\ 0)$	$(0\ 0\ 0)$
$(0\ 1\ 1)$	$(1\ 1\ 1)$
$(1\ 0\ 0)$	$(0\ 0\ 0)$
$(1\ 0\ 1)$	$(1\ 1\ 1)$
$(1\ 1\ 0)$	$(1\ 1\ 1)$
$(1\ 1\ 1)$	$(1\ 1\ 1)$

Table 3.1: Correction Table of the $\mathcal{C}(3,1)$ Repetition Code

bits) is a way to asymptotically reach optimal performances (*i.e.* channel capacity) [77], the packet size encountered in IoT scenarios (typically tens or hundreds of bits) are not compatible with such code sizes. One could aggregate successive payloads until reaching a sufficient size, but this would incur intolerable latency and would not solve the highly increased complexity of the encoders/decoders. A concrete example of FEC mechanisms in IoT scenarios are the FEC used in LoRa[2]. In LoRa, the system can choose among 4 different Hamming codes of size $\mathcal{C}(n = \{5, 6, 7, 8\}, k = 4)$ depending on the transmission quality [78]. These are very small codes with low complexity encoding/decoding but very limited performance. The most efficient rates, $r = 4/5$ and $r = 4/6$, allow only for error detection. The least efficient rates, $r = 4/7$ and $r = 4/8$, allow to correct one-bit errors. Although, the $\mathcal{C}(7, 4)$ Hamming code is a perfect code[3] , *i.e.* there is no better ECC with such size and rate, its dimensions limit its performance. In this context, Chapter 5 discusses how one can design performing short to medium length (*i.e.* tens or a few hundreds of bits) coding and decoding schemes compatible with the constraint of IoT, both in terms of complexity and packet size. Two contributions of this Ph.D. work are proposed [13, 14] to answer this question.

3.1.4 Line Encoding and Modulation

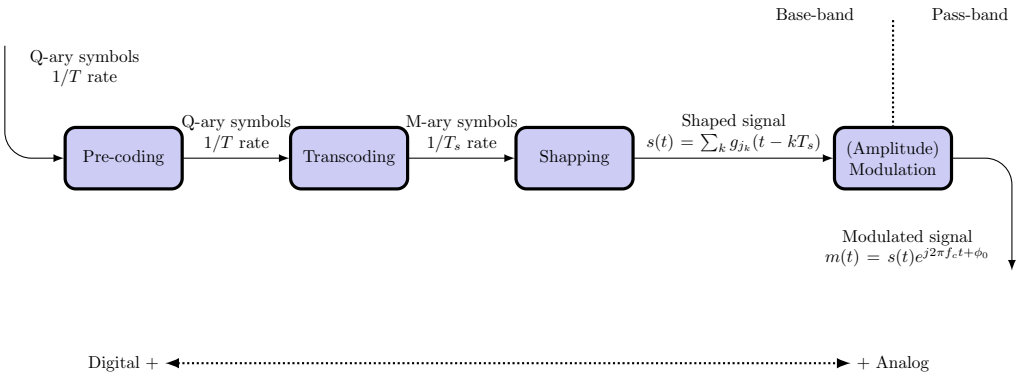


Figure 3.5: Simplified Line Coding and Modulation Chain in a Digital Communication System
The main steps of line coding and modulation are described. In this example, an amplitude modulation is considered, but any other modulation scheme could be equivalently considered.

The transmission of the information on the channel requires coding and shaping operations to transform the discrete digital signal[4] into a continuous analogue form suited to the transmission

[2]LoRa (acronyms for long range) is the PHY layer of LoRaWAN, which is an IoT network solution typically used for sensor networks on unlicensed band, providing vast coverage and low power consumption.

[3]Although the $\mathcal{C}(7, 4)$ Hamming code is perfect, it is not MDS. The singleton bound can thus not be attained for a (7,4) code.

[4]In this manuscript we only look at the case of a digital communication chain although the concept of modulation is obviously not specific to digital signals, *e.g.* we can think of the case of analogue amplitude modulation.

medium. These are the roles of the line coding and modulation functional blocks. In order to adapt the signal to the channel bandwidth, these blocks are responsible for defining the number of signal levels, the (coded) transmission rate, the filters and the symbols waveforms. All the operation conducted on the base-band signal are commonly referred to as line coding. The modulation[5] block carries out the frequency transposition of the base-band signal by means of the modulation of a carrier to make its transmission across longer distance possible and eventually provide additional immunity against noise, interference, etc.

These operations are usually decomposed into four steps: pre-coding, trans-coding, shaping and carrier modulation. Following, the channel coding, Q-ary symbols, usually binary, are provided to the line encoder at a rate $1/T$. Without modifying the symbol rate nor the alphabet size, a pre-coding step might be applied to linearly transform these symbols into other pre-coded symbols. The role of the pre-coder is to facilitate certain processing at the receiver side, or protect the transmitted data from malicious tapping. Then, the trans-coding step modify the alphabet size and symbol rate so as to adapt the transmission to the channel, and use case. Hence, it usually defines the effective number of signal levels M and symbol rate $1/T_s$. As an illustration one can decide to increase the effective bit-rate of a binary signal by grouping bits by pair so as to define 4 symbols and associated signal levels. Yet by doing so, and without changing the transmit power, the system get more sensitive to noise as the different symbol levels are now closer to each other. After trans-coding, the shaping step associate the symbols from the chosen M-ary alphabet at a rate $1/T_s$ with wave-forms to define the, usually continuous, signal:

$$s(t) = \sum_k g_{j_k}(t - kT_s) \quad j_k \in \{1, \ldots, M\} \tag{3.6}$$

The use of specific wave-forms can help reducing *Inter-Symbol Interference* (ISI), *e.g.* by using root-raised cosine filters, and/or improve the spectral efficiency of the transmission. Finally, the modulation step consists in modifying the properties (*e.g.* the amplitude, the phase, the frequency, etc.) of a sinusoidal carrier, based on the shaped signal $s(t)$. The carrier frequency has an impact on the achievable data rate and the propagation behaviour of radio waves in a given environment and scenario. As an example, a carrier signal modulated in amplitude by the shaped signal $s(t)$ would be defined as:

$$m(t) = s(t)e^{j2\pi f_c t + \phi_0} \tag{3.7}$$

where f_c is the frequency of the carrier and ϕ_0 the initial phase shift.

A very common modulation, which will be extensively considered in the rest of this manuscript, is the *Binary Phase Shift Keying* (BPSK). If considering no specific wave-form, the BPSK simply consists in modulating the phase of the carrier based on binary symbols, *i.e* 0's does not change the phase whereas 1's add 180° to the phase of the carrier.

Thanks to their orthogonality properties, a sine and cosine carriers can be separately modulated in amplitude by two signals and transmitted simultaneously as a sum, while allowing to be coherently demodulated and separated at the receiver. Such schemes are usually known as *quadrature* modulations. An example of a quadrature modulation is the 4-*Quadrature Amplitude Modulation* (QAM) where information bits are grouped in pairs to form complex symbols (the modulation symbols now belong to a quaternary alphabet). The first bits is used to modulate the amplitude of the cosine carrier, the I (In-phase) channel, while the second is used to modulate the amplitude of the sine carrier, the Q (Quadrature) channel. Quadrature modulation symbols are usually represented using complex symbols and referred to as *In-phase/Quadrature* (IQ) symbols. The real coordinate of the IQ symbol represents the amplitude of the cosine carrier, while the imaginary part represents the amplitude of the sine carrier. The modulus of the IQ symbol and its phase represent the amplitude and phase of the output signal resulting from the sum of the I and Q channel. The list of possible symbols as represented in the complex IQ plane, is commonly referred to as the *constellation* of the modulation. Such a 4-QAM modulation scheme is represented in Figure 3.6 as an example of a quadrature modulation.

[5]Because of model abstraction, the term modulation can be used somewhat improperly to refer to both the modulation of a carrier signal and the line encoder related operation.

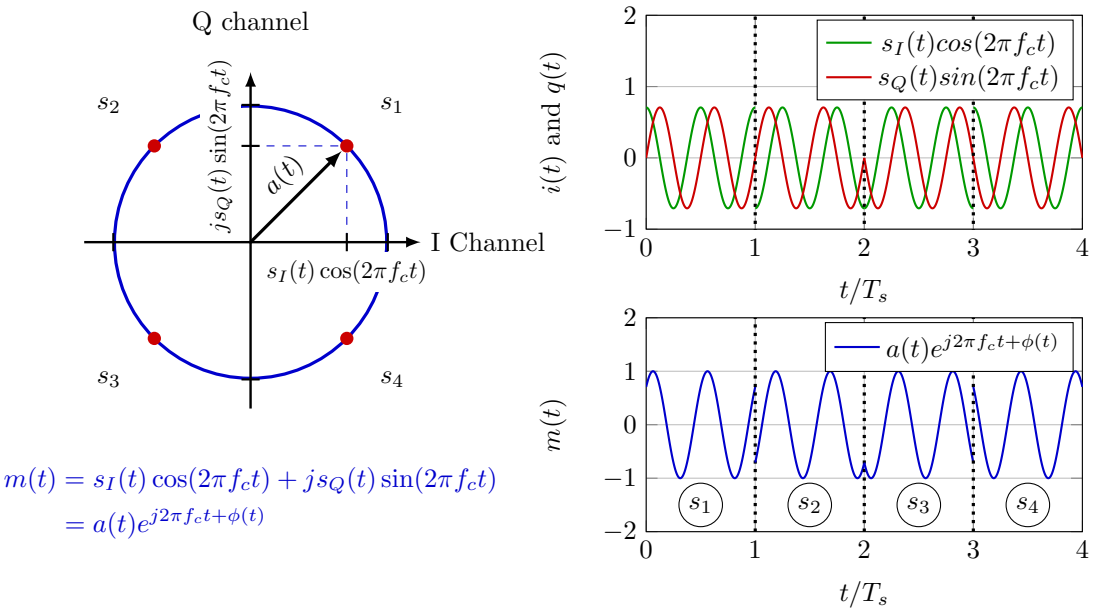


Figure 3.6: Quadrature Modulation - Example of the 4-QAM Modulation
The digital modulation considered is a four state modulation. The four possible symbols are represented in the complex plane by red points. The modulation in quadrature is realized by modulating separately a cosine carrier and a sine carrier by the amplitude on their respective channels (*i.e.* the real or imaginary part of the complex symbol IQ) before mixing them in order to obtain the final modulated signal.

Most modulations can be conveniently described using the IQ formalism. Even modulations that are not conceptually constructed around the amplitude modulation of two carriers in quadrature can be represented as a constellation on the IQ plane. For example, a BPSK modulation constellation can be described by two symbols symmetrically placed onto the real axis, or I axis. Similarly, a 4-*Quadrature Phase Shift Keying* (QPSK) modulation would have the same constellation as the previously described 4-QAM. Yet, the technical implementation of these modulations would not necessarily rely on similar modulator circuits (hence, the different names).

In the present manuscript, BPSK and *M-Ary Quadrature Amplitude Modulation* (M-QAM) digital modulations are considered. In particular Chapter 4 describes efficient and low-complexity NN-based demodulation structures for M-QAM modulations.

3.1.5 Transmission Medium

The communication between a transmitter and a receiver takes place over a *transmission medium* or *channel*, such as optical fibres, twisted copper pairs, electromagnetic waves in the air, etc. Every transmission medium, wired or wireless, has specific physical characteristics, including a limited bandwidth (which may, moreover, be restricted by legal regulations). The channel has an effect on the transmitted signal, which can vary depending on the frequency considered. Thus, a channel is generally characterised by its transfer function $H_c(f)$ which describes the attenuation of the channel as a function of the (pure) frequency of the input signal, or, equivalently, its impulse response $h_c(t)$ which describes the time response of the channel to an impulse signal. It should be noted that the transmission medium also includes the emitter and receiver analogue processing chains.

A frequently used channel model is the *Additive White Gaussian Noise* (AWGN) channel which models thermal noise in the electronic components. It is characterised by a bilateral power spectral density of $N_0/2$, where the power of the thermal noise N_0 is directly proportional to the temperature by a factor corresponding to the Boltzman constant, k_b. This noise is generated by the random thermal motion of electrons inside the conductor and is thus unavoidable.

When considering the sampling process to pass from analogue signal to digital signal, the

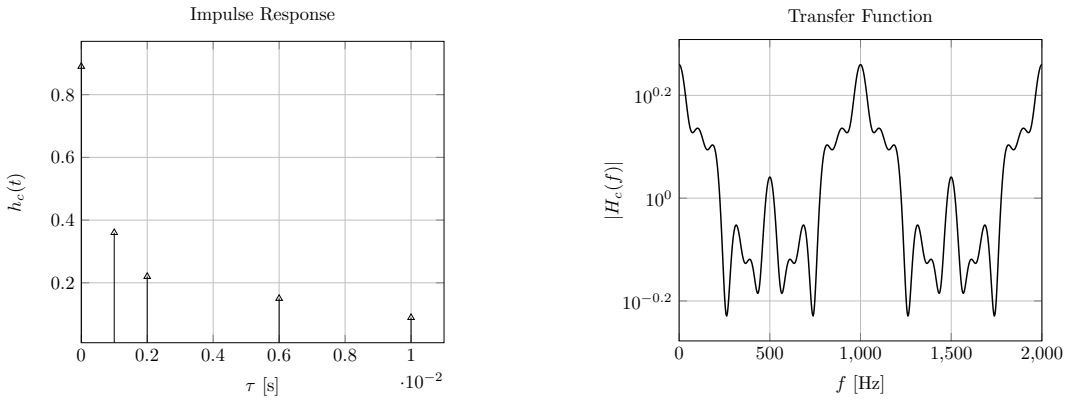


Figure 3.7: Characteristics of an Arbitrary Multi-path Channel
In this example, the channel impulse response is described using a discrete path model where each peak represents the reception of a delayed signal (related to multi-path). The time is normalised with respect to the first signal received. From the time impulse response, the corresponding frequency response of the channel can be deduced, which describes the frequencies at which the signal will be well received and those at which it will suffer potentially significant attenuation.

effective noise sample collected by the *Analogue to Digital Converter* (ADC) directly depends on its integration time, and thus the sampling time T. Indeed, although the noise power is considered constant across time:

$$\underset{T\to\infty}{\text{Lim}} \frac{1}{T}\int_0^T n^2(t)dt = N_0 \tag{3.8}$$

The average noise sample over an infinitesimal duration follows a Gaussian distribution whose variance is the thermal noise power:

$$\underset{T\to 0}{\text{Lim}} \frac{1}{T}\int_{\text{x}}^{\text{x}+T} n(t)dt \sim \mathcal{N}(0, N_0) \tag{3.9}$$

The samples are i.i.d such that the average noise tends to 0 as the measurement time increase:

$$\underset{T\to\infty}{\text{Lim}} \frac{1}{T}\int_0^T n(t)dt = 0 \tag{3.10}$$

During the sampling of an analogue signal in a ADC, while the constant signal of interest adds up coherently, the noises samples are destructively combined. Hence, the measured noise power P_n decreases as the measurement duration, *i.e* the sampling time T, increases, so that the effective *Signal to Noise Ratio* (SNR) improves:

$$P_n = \frac{1}{N}\sum_{k=1}^{N}\left(\frac{1}{T}\int_{k-T/2}^{k+T/2} n(t)dt\right)^2 = \frac{1}{N}\sum_{k=1}^{N} n^2[k] \tag{3.11}$$

The sampling time depends directly on the bandwidth of the signal of interest. To be reconstructed without ambiguity and following the Shannon-Nyquist theorem on sampling [77], the signal must be sampled at least at twice the highest frequency of interest.

From a digital base-band perspective, using the AWGN model thus consists in adding samples drawn from a Gaussian random variable whose variance depends on the bandwidth of the signal B_s (which is inversely proportional to the symbol time T_s). If we consider the transmitted base-band sampled signal $x[k]$, of power P_s, the received base-band sampled signal under AWGN channel is defined as $y[k] = x[k] + n[k]$. The noise samples follow a Gaussian distribution $n[k] \sim \mathcal{N}(0, P_n)\forall k$ and the signal-to-noise ratio in *Decibel* (dB) is $10\log_{10}(P_s/P_n)$. The perceived noise power P_n after sampling at the receiver depends directly on the transfer function of the receiver's signal processing chain before sampling. Under the consideration of a perfect processing

chain, it depends directly on the bandwidth B of the receiver's ADC, which is at least twice that of the signal, so that $P_n = BN_0 = 2B_sN_0 = \frac{2N_0}{T_s}$.

In addition to random noise, the communication channel, because of propagation effects, can also alter the phase and the amplitude of the received signal. This is commonly referred to as *fading*. For example, one can consider multi-path channels where the signal is scattered and reflected by physical obstacles in the environment, before being received multiple times by the receiver, causing constructive and destructive signal interference. An example of the impulse response of such a channel model and the corresponding transfer function is shown in the Figure 3.7. While Gaussian noise is a completely random and unavoidable process, propagation effects are caused by phenomena whose time constants are usually relatively long compared to the communication process. Therefore, it is usually possible to estimate them, which then allows their mitigation. Often, the channel is even considered to be constant over a certain period of time, under the simplifying assumption of *block-fading*. The latter consideration is quite realistic because the channel coefficients are primarily affected by the propagation environment which generally changes over time constants bigger than the symbol time of the transmitted signal. Under such channel, the received signal at the receiver is defined as the convolution of the transmitted signal by the (constant) channel impulse response upon which is added AWGN:

$$y(t) = x(t) * h_c(t) + n(t) \tag{3.12}$$

Clearly, countless channel models can be defined to describe various transmission impairments. A detailed description of these channel models is out of the scope of the present manuscript but we refer the interested reader to the very detailed book of *Proakis and Salehi* on Digital Communication for a more thorough description [79]. The following contribution of Chapters 4 and 5 always consider an AWGN source. In addition to AWGN, Chapter 4 also considers single and multi-path propagation effects.

3.1.6 Detection, Synchronisation and Equalisation

The previous section described how a transmission medium can affect the transmitted signal. We will now briefly describe the operations performed at the receiver end to mitigate the effects of this channel and thus effectively recover the transmitted signal.

The first operations performed at the receiver are the *detection* and *synchronisation* operations. Indeed, the receiver does not necessarily know when a transmission occurs and, even in the case of scheduled transmissions, the synchronisation of the emitter/receiver clocks may be imperfect. Therefore, the receiver must be able to detect the incoming signals and estimate the correct sampling time in order to receive the transmitted symbols. To do this, preamble sequences are usually added at the beginning of a transmission. These sequences have specific properties in order to increase their detection probability and to reduce as much as possible the error on the estimation of the sampling time and the beginning of the payload. An example of such sequences are *Zadoff-Chu* (ZC) sequences which have the very interesting property of having an auto-correlation of 0 except for a shift of 0 [80, 81]. This unique auto-correlation peak allows a receiver to efficiently detect and estimate the sampling time by constantly passing the received signal through a filter matched to the transmitted ZC sequence. When a correlation peak is detected, it means that a signal of interest is picked up and that the receiver should start sampling the payload at the end of the ZC sequence.

After the synchronisation, the receiver must perform channel *equalisation* to mitigate the effects of the channel. As described above, and assuming perfect synchronisation, the received signal can be modelled as the transmitted signal convolved with the channel impulse response and affected by AWGN. While there is no way to correct the individual noise samples which are, by definition, random, it is possible to estimate the channel impulse response. Under the previously described *block fading* hypothesis, the channel can be assumed to be constant over a certain time period. Since the payload is initially unknown to the receiver, it is generally not possible to use it to estimate the channel. Once again, and in the same way as for the preamble sequence (which can in fact be used for both synchronisation and equalisation), a pre-defined *pilot* sequence is added to the transmitted message to allow estimation of the channel impulse response by the receiver. Under the block fading assumption, such a pilot sequence must be

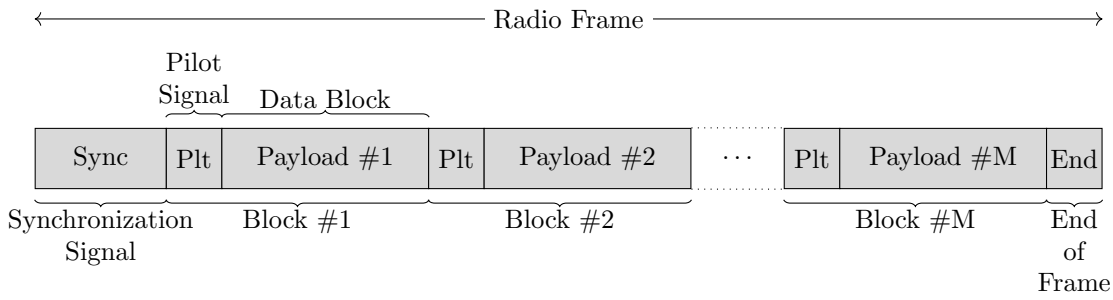


Figure 3.8: Typical Frame Structure for Synchronization and Block Equalization

regularly inserted into the transmitted signal to ensure correct channel estimation if the channel should change over time as shown in Figure 3.8.

Here, we provide the textbook example of the equalisation of a *single path*, or *single tap*, channel using the *Zero-Forcing* (ZF) method assuming perfect receiver synchronisation. Let $p[k]$ be the transmitted pilot and $x[k]$ the subsequent payload represented by vectors of complex symbols $\boldsymbol{p} \in \mathbb{C}^N$ and $\boldsymbol{x} \in \mathbb{C}^M$. Let's assume a single path channel *i.e.* a channel where the transmitted signal does not undergo multi-path propagation through reflection, diffraction, etc, and is received only once, without interference, at the receiver (or more precisely, if multi-path propagation occurs, its delay spread is smaller than the symbol time). The sampled channel impulse response $h[k]$ can be described by the vector $\boldsymbol{h} = \begin{pmatrix} h_1 & h_2 & \dots & h_L \end{pmatrix} \in \mathbb{C}^L$. In the case of a single path channel, all coefficients or *taps* of the channel are equal to 0, except the first one, h_1, which describes the attenuation and phase shift introduced by the one and only propagation path of the channel.

As described before, the effect of a block fading channel on the transmitted pilot is defined as the following convolution:

$$y_p[k] = p[k] * h[k] + n[k] \tag{3.13}$$

In vector/matrix notation such a convolution operation on a finite support can be defined as the product of the input signal by the *Toeplitz* matrix $\boldsymbol{H}$:

$$p[k] * h[k] \Longleftrightarrow \begin{pmatrix} p_1 & p_2 & \dots & p_N \end{pmatrix} \begin{pmatrix} h_1 & h_2 & h_3 & \dots & h_L & 0 & 0 & 0 & \dots & 0 \\ 0 & h_1 & h_2 & h_3 & \dots & h_L & 0 & 0 & \dots & 0 \\ 0 & 0 & h_1 & h_2 & h_3 & \dots & h_L & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & & & & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & h_1 & \dots & h_{L-2} & h_{L-1} & h_L & 0 \\ 0 & \dots & 0 & 0 & 0 & h_1 & \dots & h_{L-2} & h_{L-1} & h_L \end{pmatrix} \tag{3.14}$$

Which can be simplified in the case of the single tap channel where $h_i = 0 \forall i \neq 1$ by replacing $\boldsymbol{H}$ by a diagonal matrix (thus not taking into account the tail of the convolution which is irrelevant here because equal to 0):

$$p[k] * h[k] \Longleftrightarrow \begin{pmatrix} p_1 & p_2 & \dots & p_N \end{pmatrix} \begin{pmatrix} h_1 & 0 & \dots & 0 \\ 0 & h_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & h_1 \end{pmatrix} = h_1 \begin{pmatrix} p_1 & p_2 & \dots & p_N \end{pmatrix} \tag{3.15}$$

The received signal is thus of the form:

$$\boldsymbol{y_p}^T = \begin{pmatrix} h_1 p_1 + n_1 & h_1 p_2 + n_2 & \dots & h_1 p_N + n_N \end{pmatrix} \tag{3.16}$$

As the receiver knows the pilot sequence that has been transmitted, it can use that knowledge to estimate the channel coefficient h_1 by computing the aligned cross-correlation of the received

signal and the pilot signal, normalised by the *auto-correlation* of the pilots:

$$\begin{aligned}\hat{h}_1 = \frac{R_{py_p}[0]}{R_{pp}[0]} &= \frac{p_1^*(h_1p_1+n_1)+p_2^*(h_1p_2+n_2)+\ldots+p_N^*(h_1p_N+n_N)}{|p_1|^2+|p_2|^2+\ldots+|p_N|^2} \\ &= \frac{h_1\sum_{i=1}^N|p_i|^2+\sum_{i=1}^n p_in_i}{\sum_{i=1}^N|p_i|^2}\end{aligned} \tag{3.17}$$

Assuming an AWGN channel, the above estimator is an unbiased estimator of h_1:

$$\begin{aligned}\mathbb{E}_{\mathrm{n}\sim\mathcal{N}}\left\{\hat{h}_1\right\} &= \mathbb{E}_{\mathrm{n}\sim\mathcal{N}}\left\{\frac{h_1\sum_{i=1}^N|p_i|^2+\sum_{i=1}^n p_i\mathrm{n}_i}{\sum_{i=1}^N|p_i|^2}\right\} \\ &= \frac{h_1\sum_{i=1}^N|p_i|^2+\mathbb{E}_{\mathrm{n}\sim\mathcal{N}}\left\{\sum_{i=1}^n p_i\mathrm{n}_i\right\}}{\sum_{i=1}^N|p_i|^2} \\ &= h_1\end{aligned} \tag{3.18}$$

After estimating the channel based on the received sequence and under the hypothesis of block-fading, stating that the subsequent payload is affected by the same channel, the receiver can equalise the received signal by multiplying it by the inverse channel:

$$\tilde{x}[k] = \frac{y_x[k]}{\hat{h}_1} = \frac{y_x[k]\hat{h}_1^*}{|\hat{h}_1|^2} \tag{3.19}$$

If the channel estimation is correct, *i.e.* $\hat{h}_1 \approx h_1$, then this equalisation scheme corrects the channel distortions but can also amplify the noise terms as a side effect. Yet it remains the optimal equalisation scheme for single path channels:

$$\tilde{x}[k] = \frac{\hat{h}_1^*\left(h_1x[k]+n[k]\right)}{|\hat{h}_1|^2} \approx x[k] + \frac{\hat{h}_1^*}{|\hat{h}_1|^2}n[k] \tag{3.20}$$

Equalisation processes can get particularly complex and challenging for IoT scenarios. The Chapter 4 look at the question of expressing conventional signal processing algorithms, including single and multi-path equalisation schemes, using the framework of NN. Low complexity structures of linear equalisers are proposed, in line with the complexity constraints of IoT. Among other things, These structures enable the use of dedicated low-power and cost effective hardware, ideal for energy constrained IoT devices. Simple learning processes are also studied, showing how a simple NN-based equaliser can be trained using online methods from the ML literature.

3.1.7 Demodulation and Decoding

Finally, after equalisation the receiver must infer which symbols have been sent from the received samples. This step is commonly referred to as *demodulation*. In its most simple form, the demodulation consists in taking a decision on the symbol value, by outputting the most likely symbol. For example, in the case of a BPSK modulation with two possible binary symbols $\{0,1\}$ associated with the values $\{+1,-1\}$, if a sample is received with a value of 0.5, then the most likely BPSK symbol, *i.e.* the one with the shortest distance, is $+1$, associated to the binary value of 0. This process is known as *hard decision* (or demodulation). After the demodulation of successive samples, the obtained code-word can be provided to a FEC decoder to correct potential transmission errors. The main advantage of hard decision is that it is very simple and can easily be implemented using threshold mechanisms. The main disadvantage is that by performing the hard decision, some information is lost. Indeed, in our BPSK example, two samples of value $+0.1$ and $+0.99$ will lead to the same demodulated symbol value of 0, although the uncertainty on this decision is not the same for both samples. The first sample is very close to the decision boundary and is almost as likely to be associated with the symbol 0 as with the symbol 1. On the contrary, the second sample is very close to the symbol 0 and it is rather unlikely that the transmitted symbol was actually 1. As a result, after hard decision, all symbols

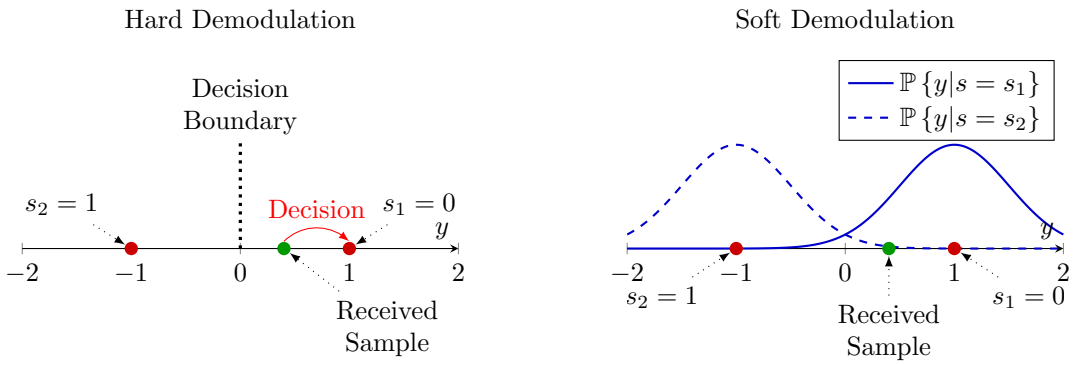


Figure 3.9: Hard versus Soft Demodulation - BPSK under AWGN
While in hard demodulation a decision is made on the individual samples before any error correction mechanism is applied, in soft decoding the probability law of the channel is used to estimate the probable values of the transmitted symbols. The final decision is then made only after the soft samples of a same code-word are combined during decoding.

in the resulting code-word are, from the FEC decoder perspective, equally reliable and the error correction, which consists in combining redundant information that might have been altered, will be performed accordingly.

A more sophisticated approach would instead compute the symbol probability distribution associated to each received samples. In our previous BPSK example, instead of associating both sample 0.1 and 0.99 to the symbol 0, the demodulator could provide a likelihood that they are associated to that symbol, *e.g.* $\mathbb{P}\{y = 0.1|s = s_1\} = 0.55$ and $\mathbb{P}\{y = 0.9|s = s_1\} = 0.95$. The latter although supposes that the conditional probability distribution of the channel is known to the receiver. Such probabilistic information are usually combined into a more compact form, so-called *(log-)likelihood ratios*, which, for a binary variable, is simply defined as:

$$\lambda(y) = \log \frac{\mathbb{P}\{y|s = s_1\}}{\mathbb{P}\{y|s = s_2\}} \tag{3.21}$$

Providing this finer information to the FEC decoder allows to greatly improve the error correction capabilities (at the cost of some additional complexity) as the decoder is now able to correct errors in a probabilistic manner, based on redundant information whose reliability is exploited. Such demodulation is know as *soft* demodulation.

To better understand the difference between soft and hard decision, let's take the example of the $\mathcal{C}(3, 1)$ repetition code described above. Assume we want to communicate the bit 0. After channel coding, the corresponding binary code-word $\begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$ is transmitted on the channel as the sequence of BPSK symbols $\begin{pmatrix} +1 & +1 & +1 \end{pmatrix}$. Let's now assume that because of channel effects and noise, the received samples are $\begin{pmatrix} -0.5 & -0.2 & 0.9 \end{pmatrix}$. Under hard demodulation, such samples would be individually associated to the BPSK symbols $\begin{pmatrix} -1 & -1 & +1 \end{pmatrix}$, corresponding to the binary code word $\begin{pmatrix} 1 & 1 & 0 \end{pmatrix}$. Hence, the FEC mechanism would decode it as the code word $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ associated to the binary digit 1. This would constitute a transmission error. The same example under a soft decision process would lead to a different result, as the definitive value of the symbols would be selected only after decoding. Indeed, assume that instead of working on the $\begin{pmatrix} -1 & -1 & +1 \end{pmatrix}$ hard demodulated sequence, the FEC decoder works directly on the received $\begin{pmatrix} -0.5 & -0.2 & 0.9 \end{pmatrix}$ sequence of samples. The Euclidean distances of that sequence with the two possible BPSK code words $\begin{pmatrix} +1 & +1 & +1 \end{pmatrix}$ (all zero code word) and $\begin{pmatrix} -1 & -1 & -1 \end{pmatrix}$ (all one code word) are 1.9 and 2.1 respectively. Based on a soft distance, *e.g.* Euclidean distance, the FEC decoder would select the BPSK code word $\begin{pmatrix} +1 & +1 & +1 \end{pmatrix}$ corresponding to the correct binary digit of 0.

It should be emphasised that this distinction between hard and soft demodulation only makes sense when considering FEC mechanisms. In the absence of an error correction mechanism, the use of a complex soft demodulation process is irrelevant.

While soft demodulation can substantially improve the performance of the decoder - it is

then referred to as a soft decoder, in contrast to a hard decoder - it also leads to a much higher decoding complexity. Chapter 5 describes low complexity soft decoding structure for LBC with sizes compatible with IoT scenarios. In particular, the proposed structure supports learning procedures to learn to decode codes from the literature with better decoding performance than that of the conventional solution. Finally, a structure is proposed to jointly design a code and the corresponding decoder *ex-nihilo* with performance comparable to the SOTA.

3.1.8 Specific Considerations in the Present Study

In the previous sections, we have provided the reader with an overview of digital communication systems that corresponds to the assumptions made in this work. The interested reader is referred to [79] for a much more detailed description of communication systems. We hope that the proposed description clarifies the context in which this study takes place. Hereafter, we would like to emphasise on some specific considerations in the present work:

- We consider only the digital part of the communication systems and not analogue processing, *e.g.* in the RF processing chain.
- We always consider base-band equivalent models which abstract the frequency transposition of the carrier.
- Both finite field signals, *e.g.* binary signal, and sampled continuous signals are considered.
- The source coding is not further elaborated in this work, and a maximum entropy random binary source is almost always considered, thus abstracting the nature of transmitted data.
- The IoT use cases underlying this work induce some specificities, especially in terms of complexity and size of information blocks.

3.2 Artificial Intelligence at the Physical Layer - A Survey

With the expansion of IoT use-cases, an efficient and low complexity PHY layer is a key requirement. In this context, the use of AI at the level of the PHY layer for various digital signal processing tasks is of growing interest. This section examines how AI and ML can be used at the PHY layer, discusses the reasons why this might be of interest, particularly for IoT scenarios, and outlines the main challenges and opportunities associated with such integration.

First, a brief historical perspective of ML methods for signal processing in communication systems is provided. Then, different approaches to the integration of artificial intelligence models at the PHY layer are presented, ranging from incremental approaches to end-to-end designs. The main opportunities, on the one hand, and challenges, on the other hand, of such integration are explored. Finally, the issue of practical integration into existing 5G and future 6G networks is discussed.

3.2.1 An Historical Perspective on Artificial Intelligence at the Physical Layer

The use of learning processes in telecommunication systems can be said to originate in the so-called *Adaptive Signal Processing* (ASP) field in the early 1960s [82]. In 1960 *Widrow and Hoff* developed the *Least-Mean Square* (LMS) algorithm [83] for electrical engineering applications, in particular adaptive antenna arrays and adaptive noise cancelling. This algorithm uses *Stochastic Gradient Descent* (SGD) methods to iteratively adapt the parameters of an adaptive linear filter, called *Adaline*. Developed at the same time as the invention of the Perceptron by *Rosenblatt*, Adaline is directly inspired by *McCulloch and Pitt* formal neuron. A few years later, in 1965, and following ideas similar to that of the LMS algorithm, *Lucky* invented the first adaptive equaliser with the *Maximum Likelihood Sequence Estimator* (MLSE) algorithm [84, 85]. In 1967, *Sondhi and Presti* developed similar idea applied to echo cancellation [86]. These algorithms apply an inverse filter to the signal, *i.e.* a filter whose response is inverse to that of the disturbance channel,

and whose parameters are found through a learning procedure. The underlying structures are in a sense conceptually close to those of simple NN. Although inspired by early work on AI and NN, they are generally not labelled as such.

Since the 1950s, AI has gone through several *winters*, *i.e.* periods of declining interest and thus funding for AI research due to disappointments and criticism. These winters have so far always been followed by a resurgence of interest years or decades later, in what are known as AI *springs*. In 2009, researchers from Stanford and Princeton universities inaugurated the ImageNet project [34], a database of millions of hand-annotated images, including 90+ dog and cat breeds. Since 2010, an annual contest sees teams of researchers from both the academia and the industry compete in classification and object detection contests. In less than 7 years of competition, contest winners' classification error rates dropped from around 25% to 7%, exceeding human recognition abilities on specific tasks [87]. This tremendous improvement in performance finds its roots in a few key technical advances, namely the democratisation of GPU accelerated computation, the development of *Deep Learning* (DL) techniques [88] and theoretical advances in ML techniques [89]. Thanks to the arrival of efficient and well-document programming frameworks such as TensorFlow [90] or PyTorch [91], and performing hardware processing platform (GPU, TPU), AI now not only supplants previous SOTA expert algorithms on CV tasks, but is also successfully applied to a wide range of applications.

The telecommunication domain and cellular networks are no exception. As illustrated by the *Hexa-X* flagship EU project on 6G network [3], AI is indeed envisioned as a critical enabler for next-generation cellular networks [6] and is, in fact, already discussed for several functional blocks of the 6G stacks such as *Radio Resource Control* (RRC), *Medium Access Control* (MAC) or, as far as this work is concerned, digital signal processing at the PHY. While signal processing for wireless communications was traditionally reserved for deterministic algorithms running on dedicated hardware - *e.g. Digital Signal Processor.ing* (DSP), *Field Programmable Gate Array* (FPGA), *Application Specific Integrated Circuit* (ASIC), etc. - the need for higher spectral efficiency has led to an ever growing algorithmic complexity. Implementing optimal signal processing algorithms while respecting the constraints of an embedded IoT platform has become an increasingly unattainable goal. Publications on AI-based signal processing at the air interface started to (re)appear around 2016, reflecting the recognition of AI techniques, and in particular NN, as interesting alternatives to conventional algorithms.

The following sections summarises contributions in AI for the PHY layer, discusses the opportunities and challenges of AI-based approaches, and finally exposes some perspectives of AI-based digital signal processing in future networks.

3.2.2 Approaches to the Integration of Artificial Intelligence at the Physical Layer

Functional Approaches: An Incremental Evolution

Traditionally, as in 4^{th} *Generation* (4G) or 5G networks, the base-band workflow is performed in a sequential way. As illustrated by Figure 3.10 which depicts the 5G NR *Physical Uplink Shared Channel* (PUSCH), each block of the processing chain performs a specialised task, *e.g.* the channel estimation block try to estimate the channel undergone by the transmitted signal in order to equalise it, the channel coding block adds redundancy to the information words to detect and correct errors, and so on. Each block's role, inputs and outputs are formally described in the *Technical Specification* (TS) documents, *e.g.* TS 38.211 [92] specifies the physical channels and modulation, TS 38.212 [93] specifies the multiplexing schemes and channel coding , etc. This section provides example of contributions that we refer to as incremental with respect to the SOTA. By incremental, we mean that these contributions do not require a change in the TS of existing communication systems:

- Some contributions, *e.g.* [94, 95, 96], borrow the Neural Network architectures used in image classification tasks to classify signals by their properties: modulations, IQ symbol patterns, etc. The hypothesis is that radio signals and spectrum can be seen as images, therefore efficient image classification techniques can be transposed to the classification of radio signals. Existing studies have shown deep learning identification methods to perform

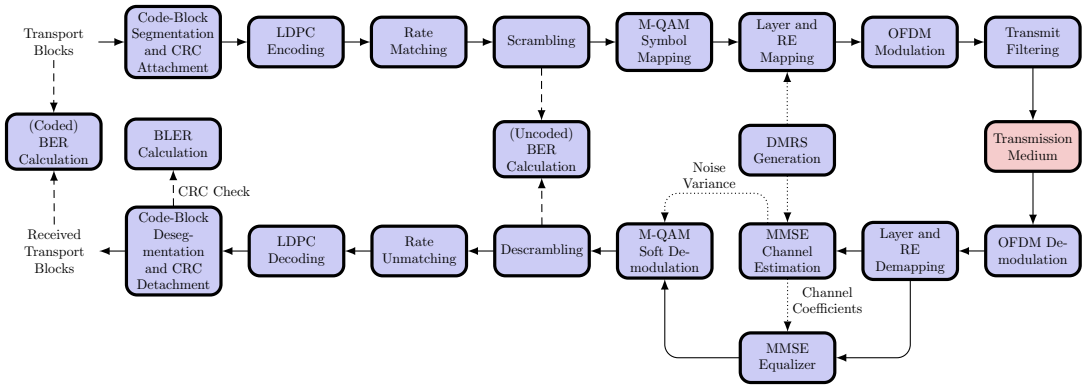


Figure 3.10: 5G NR PUSCH Chain
As specified in 3GPP TS 38.211[92] and TS 38.212[93]

better than traditional counterparts, effectively detecting and classifying modulations at SNR levels well below demodulation levels [96].

- NN have also been proposed as an alternative approach to channel estimation and characterisation. Notably, [97] discusses NN architectures that achieve equivalent performances to SOTA *Minimum Mean Square Error* (MMSE) estimation techniques for *Orthogonal Frequency Division Multiplexing* (OFDM) systems while being more robust to non-linear interference. [98] proposes to use *Channel State Information* (CSI) readings to localise transmitters in indoor environments and claims better performances than existing approaches.

- Significant efforts are invested in the study and design of NN Decoders. [99] shows that a properly trained RNN performs as well as SOTA turbo-decoders in a Gaussian channel setup and surpasses their performances for non-gaussian channels. [100] demonstrates that DNN are able to exploit common structures of different coding schemes - namely LDPC and polar codes – and effectively decode both. Several contributions study the regime of short to medium-sized block codes. In particular NN and ML are used in [101, 102, 103, 104, 105, 106] to improve the performance of BP decoding of short *Bose, Ray-Chaudhuri et Hocquenghem* (BCH) codes using trainable weighted decoder. Finally, some publications look at the question of designing new polar and LDCP codes using ML [107, 108]. In this manuscript, Chapter 5 focuses on these two latter issues.

However, as hinted by [99, 109, 110], AI-based approaches do not necessarily outperform traditional approaches on well-defined, well-modeled tasks for which there are also convenient derivations of closed-form algorithmic solutions. Furthermore, it should be emphasised that as telecommunication is a particularly mature field, several proposed solutions and algorithms are either proven optimal or closely approaching theoretical performance bounds with respect to specified criteria. As explained in [111], AI-based approaches might perform better than others is situation where there is either a model deficit or algorithmic deficit:

- A model deficit arises when the task at hand is set in a context that cannot be easily summarised by a formal model, *e.g.* challenging interference, non-Gaussian noise. The phenomena encountered become too complex to be adequately modelled and the associated problems to be solved are difficult to define formally. The use of NN and learning processes can sometimes overcome these limitations and enable the signal to be processed more accurately and finely, leading to a performance gain over standard models.

- An algorithmic deficit occurs when the task at hand is loosely described, *e.g.* distinguishing Wi-Fi from *Long-Term Evolution* (LTE) signals, or when the canonical algorithmic solution is too complex so that its computation is unbounded, *e.g.* it requires an exhaustive search over too large a set, too many iterations to obtain respectable performance, etc. Indeed, although an exact or optimal model is sometimes known, its algorithmic solution may not

be feasible within a set of operational constraints, *e.g.* exhaustive MLE decoding of large codes. The use of NN and ML allows, in some situations, to perform or assist complex signal processing operations at a reduced complexity compared to standard algorithms, while keeping the same level of performance [7]. This reduction in complexity at an equal level of performance is particularly interesting in the case of communication systems constrained in energy and/or computing capacity, such as the systems considered in the IoT context.

This discussion on model and algorithmic deficits questions the relevance of a transmission chain of individual and specialised functional blocks. On one hand, a block architecture facilitates the specification and design of a system, allowing each block to be studied, designed and optimised separately. Such choice allows for simpler models, simpler algorithmic solutions, easier to prove performances. On the other hand, this design choice hinders the optimisation of the system as a whole. Furthermore, while an optimal solution to a global problem can be found by founding optimal solution to its constituent sub-problems, this is not always possible or easy. Taking the example of channel and source coding, it has been proven that, under certain assumptions, an optimal solution for the joint source-channel coding problem can be found by optimising the source and channel coding problems separately, as indicated by the *Shannon* source-channel separation theorem [112]. However, this theorem, by considering asymptotically long code blocks, makes assumptions that cannot always be satisfied in real life, and especially not when considering the short code block sizes of IoT use cases. It might then become interesting to consider and optimise the system as a whole in order to hopefully achieve an optimal level of performance.

End-to-End Designs: A Paradigm Shift

From the previous observation, it appears that getting the full benefit of learning and optimisation algorithms might require to consider the joint optimisation of multiple signal processing blocks if not that of the complete communication chain. Hence, several publications, *e.g.* [113, 114], challenged the conventional functional approaches and proposed to use so-called end-to-end approaches:

- In [97], the authors propose to use a DNN to perform a joint channel estimation and symbol detection for an OFDM system. Initial findings reveal that the DNN approach allows to reduce the number of pilot symbols without significant degradation of the performance.
- In [115], DNN are trained to perform a direct mapping from binary words, *i.e.* data, to modulation symbols and vice versa: two DNN, one encoder and one decoder are arranged in a particular form of *Auto-Encoder* (AE) [116] where the encoding and decoding parts are the transmitter and the receiver respectively. The transmitter output is then fed to an AWGN channel model and injected in the receiver. Results show that the training allowed the AE to learn joint modulation and coding schemes, achieving similar performances to SOTA coding and modulation schemes on very simple channel models.
- By taking the AI optimisation process higher in the communication chain, recent publications discuss the problem of goal-oriented communications, also called *semantic* communications. In such a scenario, instead of being optimised on the basis of a low-level metric, *e.g. Bit Error Rate* (BER), the transmission chain is optimised on the basis of the achievement of the goal of the communication, *i.e.* the reason why the communication has taken place. We often speak of semantic communication because an effective communication, from a higher application level standpoint, is one that is able to extract from the source information the necessary semantics to achieve the communication goal at the recipient's end [117, 118, 119]. In particular, this idea introduces the notion of context and common knowledge from multiple and heterogeneous sources of information, which allow to further reduce the amount of data to be transmitted [120, 121]. This line of work is, in particular, related to the *Joint Source-Channel Coding* (JSCC) problem where one tries to define source and channel coding as a single operation [122, 123].

Although there are still relatively few publications following this new end-to-end approaches, results are encouraging, some hinting towards better performance and adaptability in complex

channels or better spectrum efficiency using DNN, etc. Obviously, those benefits come at the cost of a change of specifications. It should also be noted that such a large-scale optimisation process is not necessarily limited to the PHY layer, but can also encompass the higher layers of the network in order to optimise the network as a whole, from the high-level objectives of the application to the digital processing of the radio signal and the management of the distributed network resources.

The Interpretability of the Models

Another dimension to the question of AI integration at the PHY layer, beyond the breadth of the latter, is the type of AI models used, in particular their level of interpretability. In a very simplistic way, we can generally distinguish between what we will call *model-based* and *model-free* approaches[6], which is also often referred to as *black-box* and *clear-box* approaches respectively:

- The clear-box approaches rely on conventional signal processing algorithms as a back-bone structure for a trainable model, *e.g.* [124]. It can, in a sense, relate to ASP methods introduced in Section 3.2.1. Such structures and their results are thus usually easier to explain but the reduced model's expressiveness and the strong *inductive bias*[7] can also hinder innovative solutions. This work mainly considers such a clear box approach for the models proposed in Chapters 4 and 5.

- In contrast, black-box approaches rely on recent advances in DL and DNN to define larger models with important parameter spaces, to allow for greater model expressiveness and freedom in proposed solutions, [115, 125]. While these approaches have produced impressive results, for example in CV, they are generally less interpretable and potentially more expensive to run or train. Such approaches shift the problem of defining the telecommunication process itself to the problem of learning the latter.

Research Axes in Artificial Intelligence for the Physical Layer

From the above considerations, two principal research axis can be defined (see Figure 3.11):

Functional Versus End-to-End : in the functional approach, the communication chain is a composition of blocks with well identified functions that are optimised individually based on local metrics whereas end-to-end approach look at the complete chain as a single system to leverage joint optimisation based on global metrics.

Clear-Box Versus Black-Box : in the clear-box approach, the NN are defined based upon standard signal processing algorithms and expert knowledge leading to generally rather low-complexity and explainable models. On the contrary, in the black-box approach, the models are defined following more disruptive DL approaches with often less understandable but also more expressive and potentially more performing models.

End-to-end approaches are generally related to black-box models and clear-box models are generally used within a functional approach, but the other combinations also exist, as further described in examples from SOTA. In a clear-box and functional settings, signal processing blocks are translated into NN and eventually individually optimised using ML techniques. A particularly interesting example of this approach was recently proposed by *Nachmani et al.* [101]. The proposed *Neural Belief Propagation* (NBP) decoder introduces a promising and explainable way to improve the decoding performance of a BP iterative algorithm for short to medium length linear block codes as will be described in Chapter 5. This manuscript mainly considers such an approach. The functional approach is also used in several publication adopting more complex DNN models at the expense of a higher number of parameters and a reduced explainability. As an example, DNN decoders are proposed to learn the decoding of convolutional and turbo

[6] Not to be confused with model-based and model-free RL

[7] The inductive bias of an ML algorithm is the set of assumptions that are made to restrict the parameter and solution space based on expert knowledge of the problem. Although such a bias can help to improve the effective learning of a model, it also limits it to specific solutions. A basic example of inductive bias are weights regularisation methods.

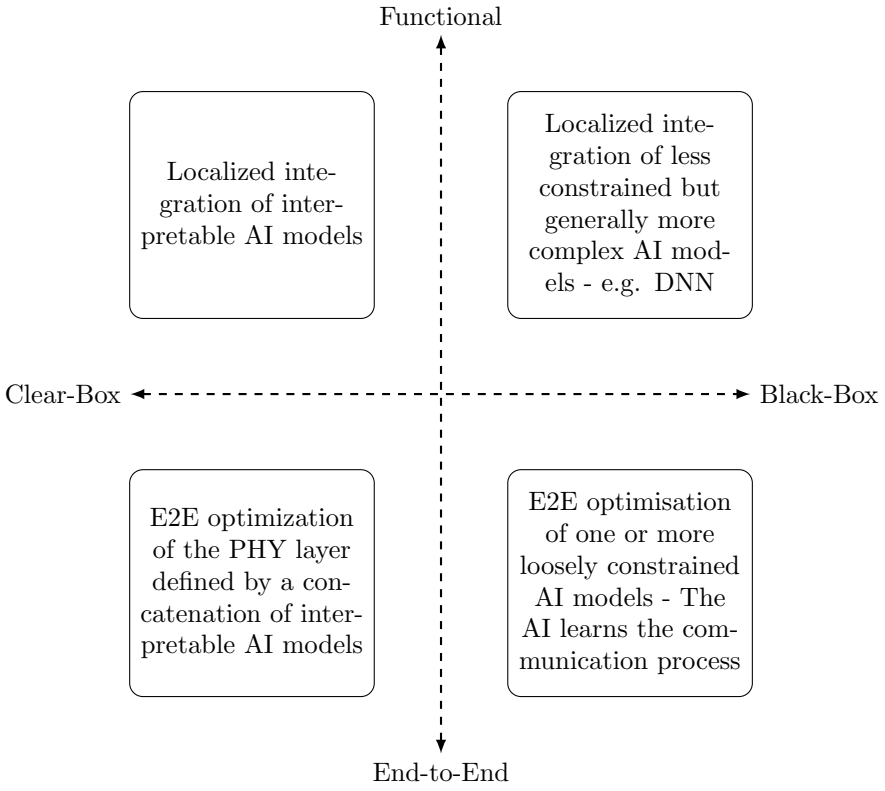


Figure 3.11: A Simple Typology of AI for PHY

codes [99] and polar codes [109]. Some interesting DL approach push this idea further, leading to hybrid LDPC/polar decoders [100].

The more disruptive black-box and end-to-end approaches try to answer the question of designing a full transceiver, jointly optimising all the functions that one can expect to be present in a communication chain (modulation, equalisation, etc.) [115, 126]. These end-to-end approaches, of a great interest, are yet facing many unanswered challenges: reduced interpretability of the models, non-differentiable channels, combinatorial explosion with respect to payload length, reduced generalisation capabilities, non-convergence of the models, etc. Yet, the end-to-end approach can also be combined with the clear-box one, by defining a fully interpetable and structured AE model based upon individually well-defined NN signal processing blocks to be optimised based on global performance metrics. This last approach will be treated superficially in Chapter 4 - where two interpretable NN-based models for equalisation and demodulation will be concatenated to form a larger NN model - and in Chapter 5 - where a fully interpretable AE structure will be proposed for the joint learning of a coding and associated decoding scheme.

Integration in Existing 5G and Future 6G Networks

As manufacturers sell products that meet specific technical specification, they do not necessarily need to explain how their solutions work. It is therefore difficult to determine where AI is present in current 5G networks, although it is most likely that it is already being used to perform specific operations within well-defined functional blocks. Manufacturers as Ericsson, Nokia and Huawei already showed their increasing interest for such techniques. In this respect, an incremental evolution towards AI may benefit to current 5G networks as the traditional functional block approach used in telecommunications can allow manufacturers to replace one block with an AI/ML solution, without affecting the specifications. Once the algorithm meets the performance and interoperability requirements defined by the 3GPP, it can be used.

As far as 6G networks are concerned, while the extent of the future AI integration remains

an open question, it is clear that AI is seen as a major technical enabler for future networks. Indeed, research has already showed that NN and ML could challenge complex conventional signal processing algorithms. Hence, Hexa-X, the EU flagship research project on future 6G networks, to which this thesis work contributes, devotes an entire WP to the following research question: *"AI-Driven Communication and Computation Co-Design"* [3]. An interesting question is that of the adoption and integration of these technologies into future networks, especially the timing of these changes. *Hoydis* describes in [11] probable steps of the integration of AI in the networks from clear-box and functional models to black-box end-to-end approaches. The methodology adopted in this thesis loosely follows such a road-map.

3.2.3 Expected Gain and Opportunities

After describing some approaches to the integration of AI/ML at the PHY we detail the identified gain and opportunities of using such techniques.

Complexity and Performance in Real-Life Conditions

Existing publications suggest that ML techniques allow for discovering NN configurations that perform similar tasks than complex SOTA approaches for a fraction of the complexity. One such example is the application to the *Sparse Code Multiple Access* (SCMA) decoder that is shown to be achieved by a NN with a computation cost eleven times lower than a SOTA approach [110]. This observation is an invitation to revisit the current architecture choices, including channel codes. Indeed, one reason to choose one scheme over another is the complexity of the decoder. In 5G networks, the complexity of polar decoders for large block lengths contributed, among other reasons, to the coexistence with LDPC codes and their corresponding BP decoders, the latter being used for data traffic and the former for control traffic. This last problem is partly at the origin of the study in Chapter 5 where an AE model is proposed to learn short-to-medium LBC codes compatible with the BP decoder in order to have an efficient and unified coding and decoding scheme for both control and data channels.

While performance of traditional approaches are often proven for given analytical models, *e.g.* polar codes are shown to be capacity achieving for symmetric binary-input memoryless channels [127], they tend to degrade significantly in a real-life scenario due to the mismatch between the analytical models used in the design of said techniques and the actual real-life scenario. In this context, NN exhibit interesting robustness characteristics. In [99], a neural decoder for turbo-codes trained using a novel loss function on an AWGN channel is shown to perform as well as a SOTA turbo-decoder on gaussian channels while outperforming it in non-gaussian channels.

Efficient Uniform and Cost Effective Hardware Architecture

As described in Chapter 2, NN lend themselves to efficient hardware architecture *e.g.* based on systolic arrays. While such implementations were usually aimed at being operated in data centers as co-processors [128], down-sized version are now available for integration in energy and size constrained IoT devices [57]. For the moment, those new chips are often restricted to inference tasks, *i.e.* they cannot train a model yet. However, they are already capable of running real-time complex vision models at 100+ frames per seconds [10], handling 2 trillion operations per Watts on 8bits integers, which is most likely enough for implementing complex signal processing tasks on the *User Equipment* (UE) side of a network as well. Google own first dedicated AI accelerator, called TPU [57], claimed performances ratio, *i.e.* relative performance per Watt, of 29 times that of contemporary GPU, and 83 times that of CPU (see Figure 3.12 and [128])[8]. TPU performance to Watts ratios make them promising enablers to reduce the energy consumption of computation-intensive tasks in a cellular network. This scenario exhibits multiple benefits from a network operator perspective:

[8]It should be noted that [128] studies only the TPUv1 generation from 2015, and that 3 new releases occurred since then, increasing the performance per Watt even further (although no relevant study were available at the moment of the writing of this manuscript).

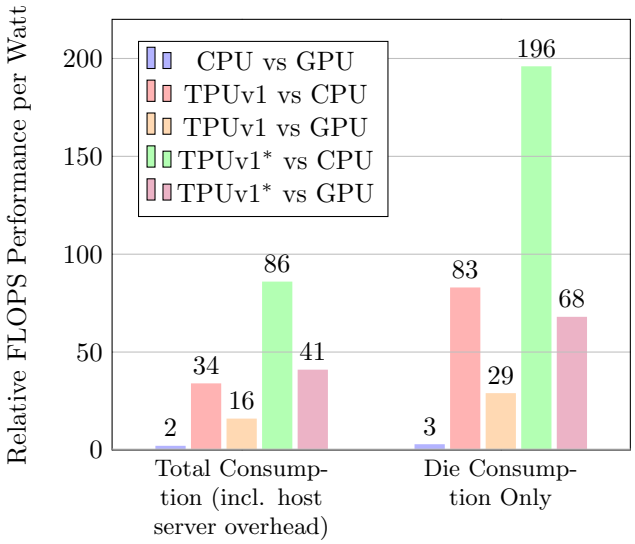


Figure 3.12: Benchmarking Neural Network Models on Different Hardware Platforms
Reproduced from [128] - Relative performance - measured in *Floating Point Operations Per Second* (FLOPS) - per Watt of power consumption on a realistic workload of 6 different neural network models running on various server targets. The study is conducted in a data centre and, therefore, the power consumption is measured either including the entire host server (left) or only the specific HW on which the models are run (right). The TPU v1 server (2015 version) shows a clear advantage over the Haswell CPU/ K80 GPU servers with up to 83 times the performance of the CPU server at equal power consumption. The TPUv1* represents a hypothetical version of the TPUv1 where equivalent technology generations to the CPU/GPU are used (*e.g.* GDDR5 memory technology, input/output bandwidth, clock frequency, etc.).

- As discussed earlier, this hardware is generic in the sense that it can change models on the fly. In addition to reducing the gross power consumption of such a hardware target, it would therefore be possible to reuse the same hardware target for multiple non-simultaneous tasks (possibly not only related to radio signal processing in a hardware pooling scenario), further reducing the device (or *Integrated Circuit* (IC)) size and power consumption.

- Since this hardware is designed as a co-processor, it may be easier and cheaper to scale up to the computational load. In addition, as mentioned above, hardware resource sharing could be considered in edge computing scenarios to offload, for example, AI-based CV tasks to the edge when the *Base Station* (BS) does not have to perform radio signal processing tasks.

In addition, the use of a similar hardware architecture between data centre *System on Chip* (SoC) and embedded SoC could be considered, which would have several advantages. As the hardware is similar, the development of ML algorithms and models should require similar skills, both on the UE (embedded) side and on the gateway side. As an example, the different TPU models produced by Google can be defined and trained within the same framework, TensorFlow [90], albeit with some limitations on the size and types of data that can be processed by the smaller embedded TPU. In addition to the skill factoring, this also opens up the possibility of distributed and/or federated learning concepts via the learning of UE models within the more capable and less constrained TPU of the infrastructure, provided that a side channel is available to transmit the information required for training and transmit the result of the training back to the UE. Such distributed learning scenarios are described in two patent applications filled during the course of this Ph.D. work [15, 16].

While such hardware is developed by several manufacturers, including Google [57], Intel [58], Nvidia [59], Qualcom [60] or Apple [61], its cost is even further reduced. The wide development of high-performance hardware architectures dedicated to the execution of NN models makes it possible to consider their use in future networks that are more performing, energy-efficient and less expensive.

3.2.4 Challenges

While NN and ML opens many opportunities, they also come with their own set of challenges which are described in this section.

Performance Guarantees and Bounds

Even-though NN are made of simple atomic components and operated on simple hardware architectures, complete models are often complex and their parametrisation remains a challenge that usually rely on ML techniques. This introduce a paradigm shift in the design of algorithms: While the traditional design approach is model-based, *i.e.* first model the environment and task then devise an algorithmic solution, ML techniques are data-based, *i.e.* the algorithmic solution is tightly bound to training data-sets. Contradictory to conventional approaches which usually allow for an analytical study of the performance of algorithms, there is often no guarantee that a NN model is able to deliver expected performances as there is no proof of correctness of the model involved in the process of learning. The actual performance is only tested on limited validation data-sets. Training and validation data-sets are therefore of critical importance to ensure models perform as expected in a live network with online learning scenarios.

Generalisation of Learning

The data used in the training of models must fit with the real-life conditions in which models will then be run. This is particularly challenging due to the radio environment differing significantly from one location to another. Caution is especially required when off-line training is in use. While online training seems therefore an interesting alternative, it raises other equally challenging issues, *e.g.* the definition of loss/error to monitor the performances, the optimisation of hyper-parameters which traditionally requires a level of human intervention and the support of training procedure at the constrained edge of the network.

Curse of Dimensionality

NN model can be challenging to train notably because of the so called "*Curse of dimensionality*". This phenomenon, well-known to the ML community, is related to the fact that the solution space of a problem grows exponentially with its dimensionality. This makes the available training data sparse and thus non-statistically significant or the needed volume of training samples too important. Similarly, the "*Curse of (code-word) dimensionality*" appears in the context of PHY layer when one tries to train models with increased information block sizes. The number of code-words associated to information blocks of size k is of 2^k which can rapidly become prohibitively large when increasing k, making the learning of large codes a challenging problem [115, 5]. In most cases, it is impractical (or impossible) to train models on exhaustive data-sets, *e.g.* all possible combination of data.

Security Concerns and Trustworthiness

The increasing complexity of NN, with too many parameters to be fully understood, raises the question of their unpredictability, or at least the difficulty of fully testing them. Obviously, testing models before operating them provides a degree of confidence but does not fully prevent from attacks based *e.g.* on adversarial ML techniques [129]. Network reliability and security are even more difficult to ensure when considering fully distributed and online learning techniques, which argues for the adoption of specific performance and monitoring procedures.

Distributed Learning

The distributed nature of a telecommunication network makes the optimisation of NN models spread across different physical devices particularly challenging, notably when considering online training procedures. As discussed earlier, introducing AI at the PHY to learn new communication schemes questions the notion of communication standard itself. In such a scenario, how to make sure that multiple devices training local AI models converges toward a globally valid

communication scheme, *a fortiori* without side-channels and coordination? How can the training metrics used to orient the model toward a better solution be exchanged in the network, even before the communication scheme is learned? In particular, when considering gradient based ML methods, how to back-propagate gradient information on a non-differentiable channel so that to update the emitter weights?

Interoperability of AI-Based Networks

How to ensure interoperability between fully AI-defined wireless systems is another open question: The concept of AI-defined air interface, by definition, means that it will not be possible anymore to specify an air interface as wireless standards do. What could be standardised is the data-set used for training, that could be built so as to include the interoperability constraint.

3.3 Conclusion of the Chapter

This preliminary chapter has provided, in a first part, the basic knowledge on the PHY layer of digital wireless communications necessary to understand the present study. The proposed description has highlighted the role of the PHY layer and its main constituent blocks while emphasising on the specific IoT context and constraints considered in the present work. In a second section, a general survey on the current state of research on AI for PHY has been proposed. The main approaches - functional versus end-to-end and clear-box versus black-box - have been presented, as well as the main challenges and opportunities of such an integration in future networks.

We now conclude the two introductory chapters on AI/ML and AI at the PHY layer before detailing the contributions of this thesis work. In particular, the next chapter describes, as a first step towards the integration of AI at the PHY layer, conventional signal processing algorithms for equalisation and demodulation in the form of efficient and low-complexity NN structures.

CHAPTER 4

DESCRIBING SIGNAL PROCESSING OPERATIONS USING NEURAL NETWORKS

Contents

4.1 Equalisation 74
4.1.1 Base-Band Equivalent Tap Channel Model 74
4.1.2 Linear Equalisation 77
Least Square Channel Estimation and Zero-Forcing Equalisation . . . 77
Minimum Mean Squared Error Equaliser 79
4.1.3 Single-Path Equalisation using Neural Networks 79
Channel Model and Theoretical Solution 80
Corresponding Neural Network Structure 81
Associated Learning Process 82
4.1.4 Extension to Multi-Path Equalisation 83
4.1.5 Perspectives 84
4.2 Designing a Neural Network Based Detection System 84
4.2.1 QAM Demodulator - Naive approach 85
Quadrature Amplitude Modulation 85
Architecture of the Demodulator Neural Network 85
Demodulator Weights Learning using Gradient Descent 87
4.2.2 Low Complexity Neural Network Based QAM Demodulator 88
Configuration and Performance of the Neural Network Demodulator . 91
Soft Demodulation 94
4.3 Experimentation on a Software Defined Radio Test-Bed 97
4.3.1 System Model and Working Assumptions 98
4.3.2 Proposed Neural Network Based Receiver and Experimental Performance 99
4.4 Conclusion of the Chapter 102

Foreword

While modern DNN leverage several hidden layers to express sophisticated non-linear functions to solve complex problems which do not have any known analytical solution, their important number of parameters might lead to complex and time-consuming inference and learning processes. These limitations might be incompatible with the energy, computational power and cost constraints of IoT and low-power networks. Moreover, due to their inherent black-box design, DNN are often not interpretable, *i.e.* they may not offer the analytical guarantees usually required for a trustworthy communication system. Such an observation advocates for the translation of conventional signal processing algorithm into their low-complexity NN counterparts which can make for a good basis to the design of structured NN or DNN models with efficient learning procedures, better generalisation capabilities and increased interpretability.

The aim of this chapter is thus to illustrate this approach with concrete examples by studying how conventional signal processing algorithms can be expressed using NN structures. As explained in previous chapters, such an approach is a first step towards the integration of efficient AI hardware platforms at the PHY layer. Moreover, it potentially enables the use of ML procedures. Furthermore, the definition of a differentiable graph structure for each of the signal processing blocks allows the use of end-to-end learning even in systems where standardised blocks have to be used. An example of such a system could be the learning of an equaliser model upstream of a demodulator model based on an output metric, *e.g.* a differentiable relaxation of the BER, calculated downstream of said demodulator (and not immediately after the equaliser). At first, we will look at the question of single and multi-path channel equalisation using simple NN structures. Then, low complexity demodulation structures will be introduced. Finally, a simple NN-based PHY layer with realistic implementation constraints, including single path equalisation and M-QAM demodulation, will be implemented using SDR, as described in the first publication of this PhD [12]. It should be emphasised that the goal of the present chapter is not to describe methods that offers raw performance improvements over conventional algorithms but rather to demonstrate how said conventional algorithms can be translated into equivalent NN structures, thus enabling the use of AI/ML inside communications systems.

4.1 Equalisation

As briefly introduced in Sections 3.1.6 and 3.1.5, several types of distortions can affect the signal transmitted over a wireless communication channel (*e.g.* shadowing, diffraction, fading, Doppler shift, scattering, path loss, reflection, etc.), thus provoking many undesired alterations of the received signal. Mitigating these effects is thus crucial for a reliable and efficient transmission process and that is why equalisation schemes are put in place. This section will look into the question of expressing conventional linear equalisation methods for single and multi-path channels using low complexity NN structures with strict mathematical equivalence.

4.1.1 Base-Band Equivalent Tap Channel Model

Before describing some linear equalisation methods, we first provide a quick description of the channel effects in the form of a sampled channel model. Indeed, although propagation effects directly affect the physical, *i.e.* analogue, signal, a base-band equivalent is often used to describe the wireless channel of a digital communication system. It is then presented as a band-limited digital filter defined by its discrete transfer function (*cf* Section 3.1.5).

A particularly important concept is that of multi-path propagation and the resulting spread of the received signal. Due to the propagation of the radio signal in an open and wide environment, and the subsequent effects of scattering, reflection, diffraction, etc., the same transmitted signal may follow different paths before reaching the receiver at different times. In simplified terms, the channel delay spread is the time interval between the first and last reception of the same signal in a multi-path channel. It is therefore directly dependent on the propagation environment. For example, in a mountain environment, the channel spread is large because of the reflection of the signal on the distant mountains. On the contrary, in an urban environment, although the number

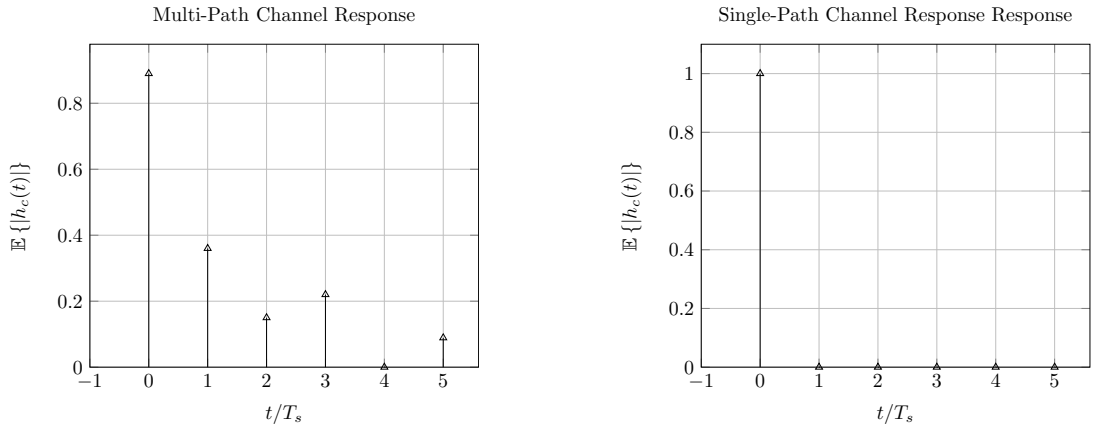


Figure 4.1: Tap Channel Models
The different taps represent the effect of the channel at a given sampling time. Time zero is set to correspond to the reception of the first path. Obviously, the channel theoretically has an infinite impulse response, although in practice the amplitude decreases rapidly, so we usually only need to consider a reasonable number of taps (or at least we set the number of taps according to the computational capacity of our equaliser).

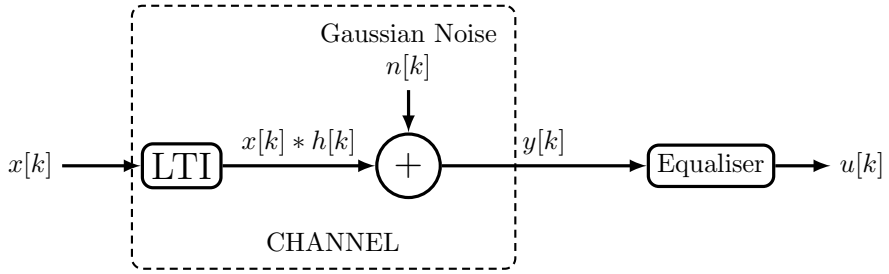


Figure 4.2: Digital Base-Band Equivalent System Model
In this work we always consider a digital base-band equivalent system model, where the output of the channel is a discrete signal $y[k]$ defined as the convolution of the discrete input signal $x[k]$ by the tap-channel coefficients $h[k]$ (represented as a Linear Time Invariant filter) and to which Gaussian noise samples $n[k]$ are added. After attenuating the effects of the channel as much as possible, the equaliser produces a signal $u[k]$.

of reflections may be higher due to the multiplicity of obstacles (buildings, cars, etc.), the delay spread is generally lower due to the smaller characteristic distances between said obstacles [130].

Assuming a discrete signal with a sample time T_s, the base-band equivalent of the impulse response of the channel can be represented by a *tap delay model*, where each tap represent the sampled impulse response of the channel at a given sample time as represented on Figure 4.1:

$$h[k] = \sum_{i=1}^{L} a_i e^{j\theta_i} \delta[(k-i)T_s] \tag{4.1}$$

When the delay spread of the channel is greater than the symbol time, one symbol interferes with subsequent ones in a so-called ISI phenomenon. If not compensated, ISI as-well as the other channel effects can introduce severe error levels.

Considering a discrete signal $x[k]$, transmitted through a *Linear Time Invariant* (LTI)[1] multi-path channel, with discrete tap impulse response $h[k]$, the received signal $y[k]$ constituted of multiple overlapping replicas of original signal will be defined as:

$$y[k] = x[k] * h[k] = \sum_{m=-\infty}^{+\infty} x[m]h[m-k] \tag{4.2}$$

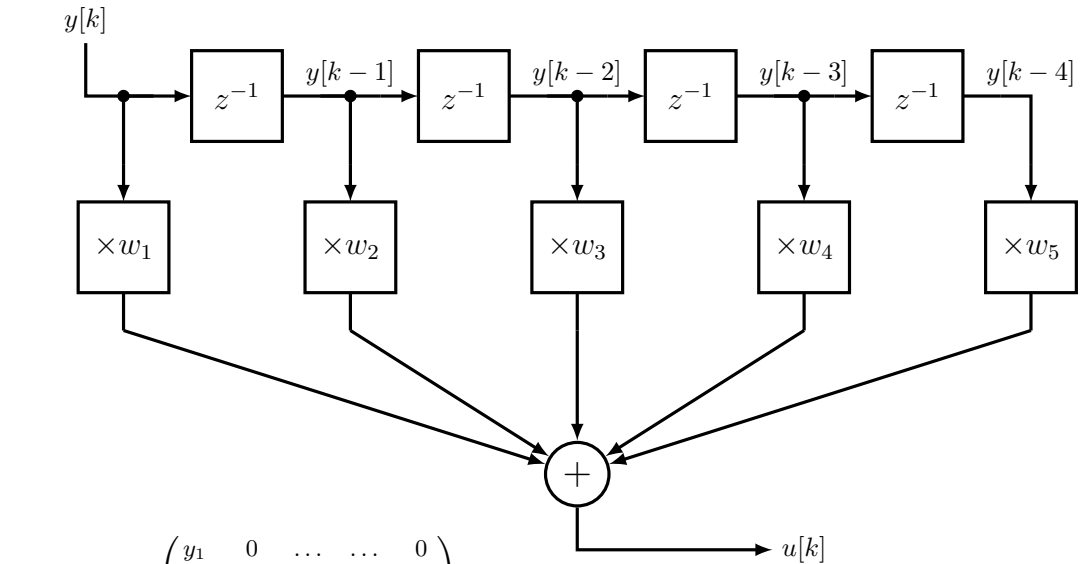
[1] A Linear Time Invariant system is a filter that produces an output signal from an input signal under linearity and time-invariance constraints. The response $y(t)$ of such a system to a given input $x(t)$ is defined by the convolution $y(t) = x(t) * h(t)$ where $h(t)$ is known as the system's impulse response. Such a system can typically be used to model a block fading channel.

In fact, as both the channel impulse response and the signal have finite support, the span of the convolution sum is reduced on that support. The operation can thus be represented using vector/matrix notations using a Toeplitz matrix as (see Section 3.1.6):

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N+L-1} \end{pmatrix} = \begin{pmatrix} x_1 & 0 & & \dots & & 0 \\ x_2 & x_1 & 0 & & & \\ x_3 & x_2 & x_1 & \ddots & & \vdots \\ \vdots & x_3 & x_2 & \ddots & 0 & \\ x_N & \vdots & x_3 & \ddots & x_1 & 0 \\ 0 & x_N & \vdots & \ddots & x_2 & x_1 \\ & 0 & x_N & & x_3 & x_2 \\ \vdots & & 0 & \ddots & \vdots & x_3 \\ & & & \ddots & x_N & \vdots \\ 0 & & \dots & & 0 & x_N \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_L \end{pmatrix} \Longleftrightarrow \boldsymbol{y} = \boldsymbol{X}\boldsymbol{h} \tag{4.3}$$

where $\boldsymbol{x} \in \mathbb{C}^N$ is the input signal and $\boldsymbol{h} \in \mathbb{C}^L$ the L-tap fading channel.

Such a channel can be modelled as a *tap-delay line*, in a similar way to that of the linear equaliser described in the next section. To reduce the channel effects one idea is to find an inverse filter, called an equaliser, and apply it to the received signal (see Figure 4.2).



$$\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N+5-1} \end{pmatrix} = \begin{pmatrix} y_1 & 0 & \dots & \dots & 0 \\ y_2 & y_1 & \ddots & & \vdots \\ \vdots & y_2 & \ddots & 0 & \vdots \\ y_N & \vdots & \ddots & y_1 & 0 \\ 0 & y_N & & y_2 & y_1 \\ \vdots & 0 & \ddots & \vdots & y_2 \\ \vdots & & \ddots & y_N & \vdots \\ 0 & \dots & \dots & 0 & y_N \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_5 \end{pmatrix} \Longleftrightarrow \boldsymbol{u} = \boldsymbol{Y}\boldsymbol{w} \Longleftrightarrow u[k] = y[k] * w[k]$$

Figure 4.3: Schematic of a Linear Equaliser Represented Using a Transversal Filter
The discrete input signal to be equalised is delayed by a 5-tap delay line and each delayed version of the signal is multiplied by the equaliser parameters and linearly combined to obtain the equalised signal.

4.1.2 Linear Equalisation

One very common family of equalisers, which will be the only one treated in this manuscript, are the linear equalisers which aim at applying to the received signal a linear filter which seeks to reduce the channel effects. Such a filter is usually defined and implemented as a finite transversal filter or so called tap-delay line as represented in Figure 4.3. In such filter, the received discrete signal $y[k]$ is delayed on a delay-line of a certain length (or *memory*) and each of the delayed samples are multiplied by the equaliser weights and linearly combined so as to form the equalised signal $u[k]$. The memory of the filter should be adequately selected so as to match the delay spread caused by the channel, but can also be limited by the technical implementation complexity of the equaliser system.

A question arising from the above description is that of the parametrisation of such an equaliser. We will cover in the upcoming sections two famous methods, namely ZF and (linear) MMSE.

Least Square Channel Estimation and Zero-Forcing Equalisation

In most digital communications applications, the channel response $\boldsymbol{h}$ is not known in advance. Although some use-case dependent expert knowledge is known (such as the expected delay spread of the channel and therefore the number of taps needed to represent it with a good level of fidelity), an online estimate of the channel is almost always required. One such channel estimation method, known as the *Least Square* (LS) channel estimation, aims at performing the following minimisation:

$$\hat{\boldsymbol{h}} = \underset{\boldsymbol{h}}{\operatorname{argmin}} ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{h}||_2^2 = \underset{\boldsymbol{h}}{\operatorname{argmin}} ||\boldsymbol{X}\boldsymbol{h}^\star - \boldsymbol{X}\boldsymbol{h}||_2^2 \tag{4.4}$$

where $\hat{\boldsymbol{h}}$ is the estimated channel and $\boldsymbol{h}^\star$ the effective channel.

By finding the value $\hat{\boldsymbol{h}}$ such that the derivative of the function with respect to $\boldsymbol{h}$ is zero, we find that the LS channel estimate is:

$$\hat{\boldsymbol{h}} = (\boldsymbol{X}^\dagger \boldsymbol{X})^{-1} \boldsymbol{X}^\dagger \boldsymbol{y} \tag{4.5}$$

where $\boldsymbol{X}^\dagger$ denotes the *trans-conjugate* of matrix $\boldsymbol{X}$, also referred to as *Hermitian* transpose of $\boldsymbol{X}$.

Obviously, and as described in Section 3.1.6, this channel estimation supposes the knowledge of the transmitted sequence $\boldsymbol{x}$ by the use of a pilot sequence.

After channel estimation, the ZF equaliser can be used to equalise the received signal on the basis of the estimated channel (usually in the frequency domain). The ZF enforces a completely flat (constant) frequency response of the channel/equaliser combination, by choosing the equaliser transfer function as [79]:

$$E(f) = \frac{1}{\hat{H}(f)} \tag{4.6}$$

where $E(f)$ is the frequency response of the equaliser and $\hat{H}(f)$ the least-square estimate of the channel frequency response.

The ZF equaliser is optimal in single-path channels and, in the absence of channel noise, in multi-path channels as well, where it can completely eliminate ISI. However, in practice, ZF equalisation does not work well in most applications because, under noisy channels, the ZF equaliser strongly amplifies the noise at frequencies f where the channel response $H(f)$ has a low amplitude (commonly referred to as a channel zero) in an attempt to completely negate its effect [79]. An example of such noise enhancement is described in Figure 4.4.

In fact, the purpose of an equaliser is not to remove ISI at all costs, but rather to minimise the final BER. The noise amplification property of ZF equalisers makes them unsuitable for such a task. A more balanced linear equaliser in this case is the MMSE equaliser, which generally does not completely eliminate ISI but rather minimises the total power of the noise and ISI components in the output [79].

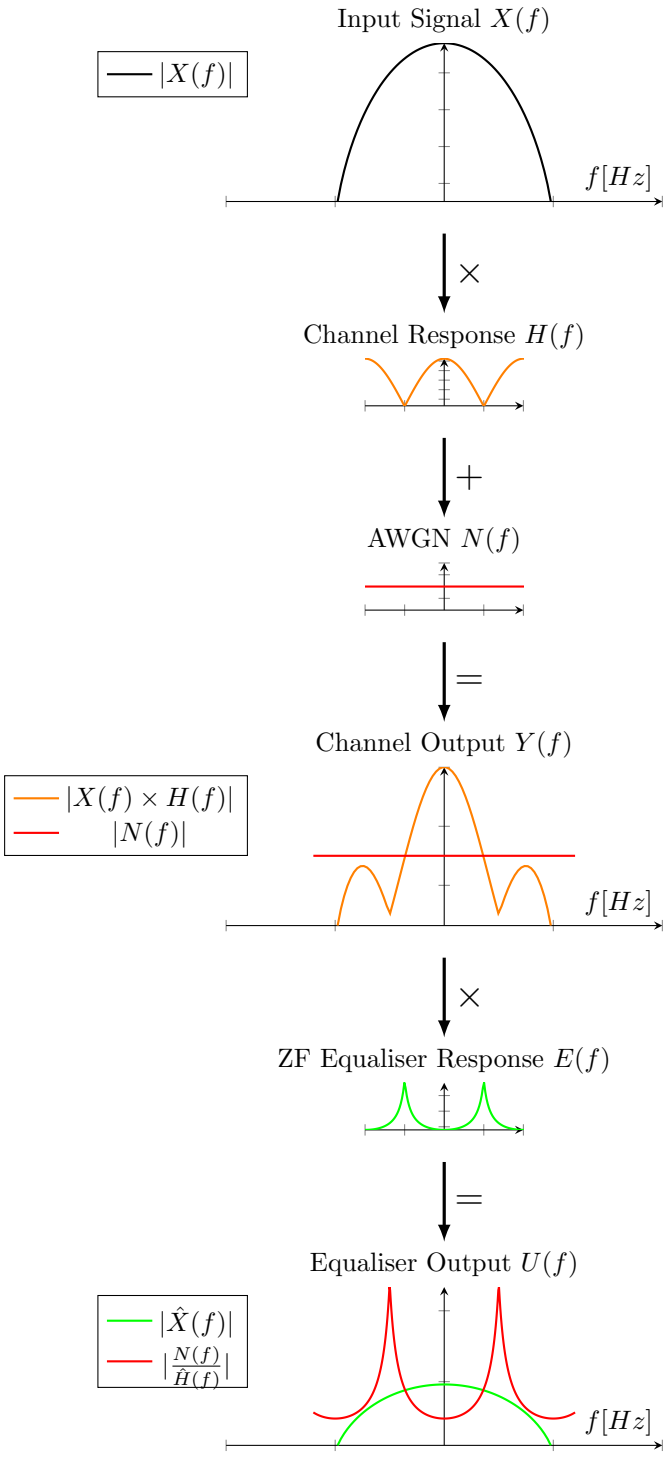


Figure 4.4: Noise Enhancement Problem in Zero-Forcing Equalisers
One of the main problems with the ZF equaliser is that, by imposing a completely flat response of the channel/equaliser ensemble to eliminate ISI, the ZF equaliser significantly degrades the SNR and colors the noise by increasing its level at the channel nulls. Note that a perfect channel estimation is considered here (*i.e.* $\hat{H}(f) = H(f)$.

Minimum Mean Squared Error Equaliser

Contrarily to ZF where the channel is first estimated and then an inverse filter which minimizes the ISI is designed, MMSE seeks to directly optimise the linear equaliser parameters so as to minimise the errors. The goal of the MMSE equaliser is thus to solve the following equation (real signals are considered here):

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathbb{E}\left\{||\mathbf{u} - \boldsymbol{x}||_2^2\right\} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathbb{E}\left\{||\mathbf{Y}\boldsymbol{w} - \boldsymbol{x}||_2^2\right\} \tag{4.7}$$

where $\boldsymbol{w}$ are the parameters of the L-tap (or order) of the linear equaliser filter (also referred as a *Wiener* filter in the case of MMSE equalisation [131]), $\boldsymbol{x}$ the known vector of pilot symbols, $\mathbf{u}$ the random vector of equalised samples and $\boldsymbol{Y}$ the random Toeplitz representation of the received samples $\mathbf{y}$.

By developing the above expression we obtain:

$$\begin{aligned}\mathbb{E}\left\{||\mathbf{Y}\boldsymbol{w} - \boldsymbol{x}||_2^2\right\} &= \mathbb{E}\left\{\boldsymbol{w}^T\mathbf{Y}^T\mathbf{Y}\boldsymbol{w} - 2\boldsymbol{w}^T\mathbf{Y}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{x}\right\} \\ &= \boldsymbol{w}^T\mathbb{E}\left\{\mathbf{Y}^T\mathbf{Y}\right\}\boldsymbol{w} - 2\boldsymbol{w}^T\mathbb{E}\left\{\mathbf{Y}^T\boldsymbol{x}\right\} + \boldsymbol{x}^T\boldsymbol{x} \\ &= \boldsymbol{w}^T\boldsymbol{R}_{yy}\boldsymbol{w} - 2\boldsymbol{w}^T\boldsymbol{r}_{xy} + P_x\end{aligned} \tag{4.8}$$

where $\boldsymbol{R}_{yy} = \mathbb{E}\left\{\mathbf{Y}^T\mathbf{Y}\right\}$ is the auto-correlation matrix of the received signal $y[k]$, $\boldsymbol{r}_{xy} = \mathbb{E}\left\{\mathbf{Y}^T\boldsymbol{x}\right\}$ the cross-correlation vector between received signal $y[k]$ and transmitted one $x[k]$ and P_x is the power of transmitted signal.

To find the vector $\boldsymbol{w}$ which minimizes the expression above, we calculate its gradient:

$$\nabla_{\boldsymbol{w}}\mathbb{E}\left\{||\mathbf{Y}\boldsymbol{w} - \boldsymbol{x}||_2^2\right\} = 2\boldsymbol{R}_{yy}\boldsymbol{w} - 2\boldsymbol{r}_{xy} \tag{4.9}$$

The minimum of the function is found by solving the system of equations defined by a gradient equal to zero, commonly known as the *Wiener-Hopf* equations:

$$2\boldsymbol{R}_{yy}\boldsymbol{w} - 2\boldsymbol{r}_{xy} = 0 \Longleftrightarrow \begin{pmatrix} R_{yy}[0] & R_{yy}[1] & \dots & R_{yy}[L-1] \\ R_{yy}[1] & R_{yy}[0] & \dots & R_{yy}[L-2] \\ \vdots & \vdots & \ddots & \vdots \\ R_{yy}[L-1] & R_{yy}[L-2] & \dots & R_{yy}[0] \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_L \end{pmatrix} = \begin{pmatrix} r_{xy}[0] \\ r_{xy}[1] \\ \vdots \\ r_{xy}[L-1] \end{pmatrix} \tag{4.10}$$

where $R_{yy}[n] = \sum_i y[i]y[i+n]$ and $r_{xy}[n] = \sum_i x[i]y[i+n]$ (the expectation is approximated by the empirical mean). The auto-correlation matrix $\boldsymbol{R}_{yy}$ is a symmetric *Positive Definite* (PD) Toeplitz matrix, such that the Wiener filter has a unique solution $\boldsymbol{w} = \boldsymbol{R}_{yy}^{-1}\boldsymbol{r}_{xy}$. Furthermore, *Levinson-Durbin* algorithm can be used to solve the Wiener-Hopf equations without requiring an explicit inversion of $\boldsymbol{R}_{yy}$ [131].

When considering complex signal, the above system of equation becomes [132]:

$$\begin{pmatrix} R_{yy}[0] & R_{yy}^{\dagger}[1] & \dots & R_{yy}^{\dagger}[L-1] \\ R_{yy}[1] & R_{yy}[0] & \dots & R_{yy}^{\dagger}[L-2] \\ \vdots & \vdots & \ddots & \vdots \\ R_{yy}[L-1] & R_{yy}[L-2] & \dots & R_{yy}[0] \end{pmatrix} \begin{pmatrix} w_1^{\dagger} \\ w_2^{\dagger} \\ \vdots \\ w_L^{\dagger} \end{pmatrix} = \begin{pmatrix} r_{xy}[0] \\ r_{xy}[1] \\ \vdots \\ r_{xy}[L-1] \end{pmatrix} \tag{4.11}$$

Although we speak of an MMSE equaliser, it could also be referred to as a Wiener equaliser, as we in fact seek to find the parameters of the Wiener filter that minimise the MMSE criterion.

4.1.3 Single-Path Equalisation using Neural Networks

Now that the classical solutions of ZF and MMSE equalisers have been introduced, this section examines the issue of expressing such linear equalisers in the form of a simple NN structures, more specifically in the case of single path equalisation.

In the simplest case, there is only one channel path[2], or more precisely one dominant path, to be considered. This is referred to as a single path propagation channel. This situation is considered favourable because the channel introduces mainly delay and attenuation but no ISI, and is therefore easy to estimate and correct. This situation is typically encountered in *Line of Sight* (LoS) scenarios where the transmitter and receiver are in direct view. Some technical solutions, such as *Cyclic Prefixed Orthogonal Frequency Division Multiplexing* (CP-OFDM), also allow practical consideration of single-path equivalent sub-channels, which considerably reduces the complexity of equalisation (this is however outside the scope of this study).

Channel Model and Theoretical Solution

In this section we consider the single-path base-band equivalent channel defined by the following combination of propagation effects and hardware impairments[3]:

AWGN: real and imaginary parts of the noise are sampled from a bi-variate zero-mean Gaussian distribution:

$$\boldsymbol{n}_k = \begin{pmatrix} n_{i_k} \\ n_{q_k} \end{pmatrix} \text{ with } n_{i_k}, n_{q_k} \overset{i.i.d}{\sim} \mathcal{N}\left(0,\ \sqrt{N_0/2}\right) \tag{4.12}$$

where $\boldsymbol{n}_k$ denotes the k-th bi-variate sample of the noise signal $\boldsymbol{n}[k]$

Phase shift: symbols are rotated by an angle ϕ with the following rotation matrix:

$$\boldsymbol{R} = \begin{pmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{pmatrix} \tag{4.13}$$

IQ amplitude imbalance: a different scaling factor is applied to I and Q channels as described in the following diagonal stretching matrix:

$$\boldsymbol{A} = \begin{pmatrix} \alpha_i & 0 \\ 0 & \alpha_q \end{pmatrix} \tag{4.14}$$

IQ offset: a different offset is applied to I and Q channels:

$$\boldsymbol{o} = \begin{pmatrix} \omega_i \\ \omega_q \end{pmatrix} \tag{4.15}$$

Phase shift, IQ amplitude imbalance and IQ offset are considered constant under block-fading hypothesis[4]. The overall channel effect on a given sample is:

$$\boldsymbol{y_k}^T = \boldsymbol{x_k}^T \boldsymbol{R}\boldsymbol{A} + (\boldsymbol{o} + \boldsymbol{n_k})^T \tag{4.16}$$

$$\begin{pmatrix} y_{i_k} \\ y_{q_k} \end{pmatrix}^T = \begin{pmatrix} x_{i_k} \\ x_{q_k} \end{pmatrix}^T \begin{pmatrix} \alpha_i \cos(\phi) & \alpha_q \sin(\phi) \\ -\alpha_i \sin(\phi) & \alpha_q \cos(\phi) \end{pmatrix} + \begin{pmatrix} \omega_i + n_{i_k} \\ \omega_q + n_{q_k} \end{pmatrix}^T$$

where $\boldsymbol{x_k}$ is the k-th input symbol and $\boldsymbol{y_k}$ the corresponding sample, affected by the channel.

On the receiver side, the effect of the channel must be mitigated by a single-path equaliser. perfect synchronisation is considered. The equalised sample is noted as:

$$\boldsymbol{u}_k = \tilde{\boldsymbol{x}}_k + \tilde{\boldsymbol{n}}_k \tag{4.17}$$

where $\boldsymbol{u}_k$ is the equalised sample and $\tilde{\boldsymbol{n}}_k$ is the noise sample after equalisation that can be described, without loss of generality, as a bi-variate AWGN.

[2]We use the terms tap and path interchangeably here to refer to the base-band equivalent paths of the channel. It should be noted that the term path is often used to describe the physical path of the channel (i.e. a continuous channel), while the term tap is used to describe the digital processing of these paths.

[3]Complex IQ samples are represented in vector notation for direct conversion to NN structures.

[4]As described in Section 3.1.5, the block-fading hypothesis assumes that the channel coefficients remain constant for a block of K symbols. This duration depends on the channel coherence time.

The objective of the equalisation block is to retrieve the original samples before the channel impairments. The optimal solution given the single-path channel impairments described in Equation (4.16), commonly referred to as ZF, is defined as:

$$\begin{aligned} \boldsymbol{u_k}^T &= (\boldsymbol{y_k} - \boldsymbol{o})^T (\boldsymbol{RA})^{-1} \\ &= \boldsymbol{y_k}^T (\boldsymbol{RA})^{-1} - \boldsymbol{o}^T (\boldsymbol{RA})^{-1} \\ &= \boldsymbol{x_k}^T + \boldsymbol{n_k}^T (\boldsymbol{RA})^{-1} \end{aligned} \tag{4.18}$$

where

$$(\boldsymbol{RA})^{-1} = \frac{1}{\alpha_i \alpha_q} \begin{pmatrix} \alpha_q \cos(\phi) & -\alpha_q \sin(\phi) \\ \alpha_i \sin(\phi) & \alpha_i \cos(\phi) \end{pmatrix} \tag{4.19}$$

As shown in Equation 4.18 the ZF methods simply consists in multiplying the received signal by the inverse channel (which obviously needs to be estimated first, eventually using the previously described LS method).

Corresponding Neural Network Structure

From the optimal equalisation scheme presented previously, one can deduce a straightforward NN implementation. Indeed, under the considered channel model, the optimal equalisation scheme from Equation 4.18 is simply the equation of a linear biased layer of two neurons:

$$\boldsymbol{u_k}^T = \boldsymbol{y_k}^T (\boldsymbol{RA})^{-1} - \boldsymbol{o}^T (\boldsymbol{RA})^{-1} = \boldsymbol{y}_k^T \boldsymbol{W} + \boldsymbol{c}^T \tag{4.20}$$

where $\boldsymbol{y_k}$ is the k-th input sample (real and imaginary part of the sample form a vector of two elements) and $\boldsymbol{u_k}$ the k-th equalised sample. $\boldsymbol{W}$ and $\boldsymbol{c}$ are the (2×2) weights matrix (each column representing one of the two kernels) and the bias vector of size two, respectively. Complex-valued NN are still at the stage of research such that this representation using separate I and Q channel is convenient to describe and process complex-valued signals.

Given the optimal solution described in equation (4.18), the optimal NN weights $\boldsymbol{W}$ and bias $\boldsymbol{c}$ are:

$$\begin{cases} \boldsymbol{W}_{\text{optimal}} = (\boldsymbol{RA})^{-1} \\ \boldsymbol{c}^T_{\text{optimal}} = -\boldsymbol{o}^T (\boldsymbol{RA})^{-1} \end{cases} \tag{4.21}$$

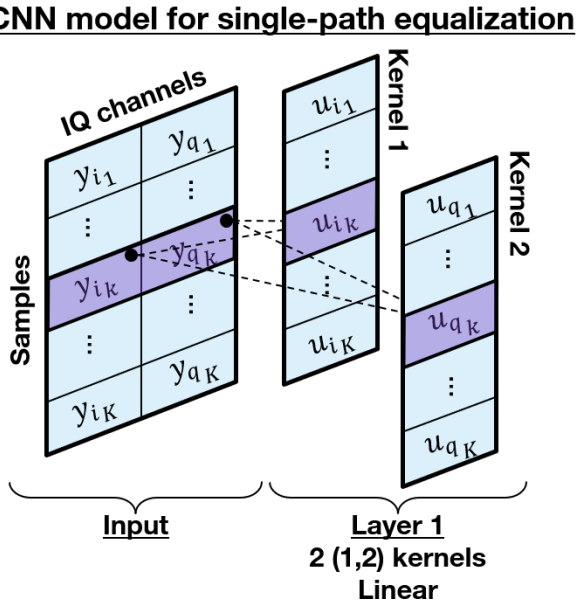


Figure 4.5: Proposed CNN Architecture for Single-Path Equalisation
The proposed layer uses six shared parameters (two weights and one bias per kernel) to equalise a complete batch of sample whose size must not exceed the coherence time of the block-fading channel.

The single-path equalisation operation can be applied independently to all samples of a given frame affected by the same channel under block-fading hypothesis. A CNN is therefore particularly suited in this case and allows for an efficient parallel processing of samples blocks. The CNN shift invariant architecture, based on shared weights, drastically reduces the number of trainable parameters, independently of the number of samples to be processed [133]. The proposed minimal CNN to perform single-path equalisation consists of one linear layer (*i.e.* without a non-linear activation function) with only two kernels of size $(1, 2)$ as described in Figure 4.5. An input matrix of shape $(K, 2)$ is considered, where K corresponds to the number of samples and 2 to real (I channel) and imaginary (Q channel) parts of these samples.

Figure 4.6 shows the result of the operation applied by this minimal CNN to a 16-QAM constellation altered by a channel with arbitrarily defined impairments. One can see that the samples are perfectly equalised under the, manually configured, optimal parameterisation of Eq. (4.21).

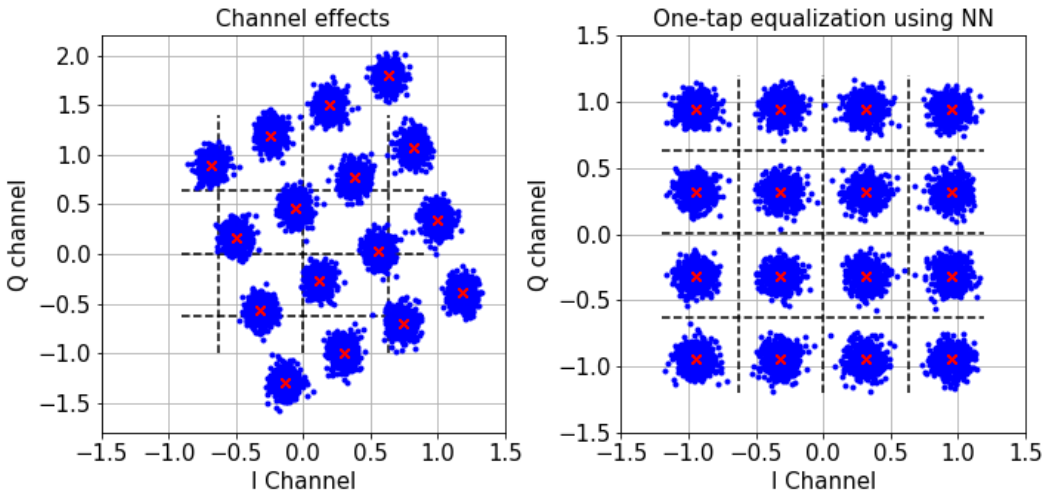


Figure 4.6: Single Path Equalisation Using Convolutional Linear Neural Network Layer
Left figure: Received samples with impairments - Right figure: Samples equalised by the NN.

Associated Learning Process

The previous sections proposes a mathematical definition of a minimal NN architecture compatible with an optimal single-path equalisation procedure, and an analytical derivation of its parameters which corresponding to ZF solution described in Section 4.1.2. Obviously, such NN structure could be configured and continuously adapted to the channel evolution using a ML algorithm.

In this section, such an approach is described. The latter aims to find the weight matrix $\hat{\boldsymbol{W}}$ and bias vector $\hat{\boldsymbol{c}}$ that minimise the MSE between the transmitted samples $\boldsymbol{x}[k]$ and the equalised one $\boldsymbol{u}[k]$, as described by the following equation:

$$\hat{\boldsymbol{W}}, \hat{\boldsymbol{c}} = \underset{\boldsymbol{W},\boldsymbol{c}}{\operatorname{argmin}} \sum_k \left(\boldsymbol{x}_k^T - \boldsymbol{u}_k^T\right)^2 = \underset{\boldsymbol{W},\boldsymbol{c}}{\operatorname{argmin}} \sum_k \left(\boldsymbol{x}_k^T - \mathbf{y}_k^T \boldsymbol{W} + \boldsymbol{c}^T\right)^2 \tag{4.22}$$

This method corresponds exactly to the MMSE equaliser solution. Obviously, as the transmitted samples x_k are unknown during relevant date transmission, pilot sequences known by the receiver must be regularly inserted. This learning process will be implemented in Section 4.3 where a prototype NN-based transmission chain will be proposed and evaluated in real conditions using SDR cards

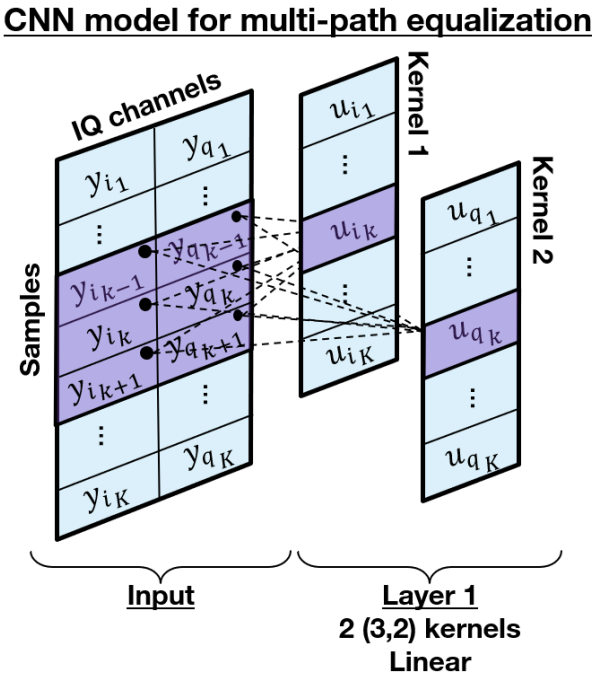


Figure 4.7: Multi-Path Equalisation Using Linear Convolutional Neural Network Layer
Example of a 3-tap CNN-based linear equaliser.

4.1.4 Extension to Multi-Path Equalisation

Previous section described how a conventional single-path equalisation algorithm can be translated into a very simple NN architecture. This section briefly extends previous results to the case of multi-path channel using a similar approach. As in previous section, the linear equaliser is defined as a CNN as shown in Figure 4.7. Although, we now consider a multi-tap linear equaliser, so as to match the channel memory, and the CNN kernel size is thus increased accordingly. In the example of Figure 4.7 a 3-tap linear IQ equaliser is considered. The computation performed by such an equaliser can be summarised by the following equation:

$$\begin{aligned}\begin{pmatrix} u_1^{(I)} & u_1^{(Q)} \\ u_2^{(I)} & u_2^{(Q)} \\ \vdots & \\ u_{N+3-1}^{(I)} & u_{N+3-1}^{(Q)} \end{pmatrix} &= \begin{pmatrix} y_1^{(I)} & 0 & 0 \\ y_2^{(I)} & y_1^{(I)} & 0 \\ \vdots & y_2^{(I)} & y_1^{(I)} \\ y_N^{(I)} & \vdots & y_2^{(I)} \\ 0 & y_N^{(I)} & \vdots \\ 0 & 0 & y_N^{(I)} \end{pmatrix} \begin{pmatrix} w_{I_1}^{(I)} & w_{Q_1}^{(I)} \\ w_{I_2}^{(I)} & w_{Q_2}^{(I)} \\ w_{I_3}^{(I)} & w_{Q_3}^{(I)} \end{pmatrix} \\ &+ \begin{pmatrix} y_1^{(Q)} & 0 & 0 \\ y_2^{(Q)} & y_1^{(Q)} & 0 \\ \vdots & y_2^{(Q)} & y_1^{(Q)} \\ y_N^{(Q)} & \vdots & y_2^{(Q)} \\ 0 & y_N^{(Q)} & \vdots \\ 0 & 0 & y_N^{(Q)} \end{pmatrix} \begin{pmatrix} w_{I_1}^{(Q)} & w_{Q_1}^{(Q)} \\ w_{I_2}^{(Q)} & w_{Q_2}^{(Q)} \\ w_{I_3}^{(Q)} & w_{Q_3}^{(Q)} \end{pmatrix} + \begin{pmatrix} c_I & c_Q \end{pmatrix} \\ \boldsymbol{U} &= \boldsymbol{Y}^{(I)}\boldsymbol{W}^{(I)} + \boldsymbol{Y}^{(Q)}\boldsymbol{W}^{(Q)} + \boldsymbol{c} \end{aligned} \tag{4.23}$$

Similarly, to the case of single-path equalisation, one can train the CNN model so as to

minimise the MSE loss function:

$$\hat{\boldsymbol{W}}^{(I)}, \hat{\boldsymbol{W}}^{(Q)}, \hat{\boldsymbol{c}} = \underset{\boldsymbol{W}^{(I)}, \boldsymbol{W}^{(Q)}, \boldsymbol{c}}{\operatorname{argmin}} \sum_{k} \left(\boldsymbol{x}_k^T - \boldsymbol{u}_k^T \right)^2 \tag{4.24}$$

The proposed system is tested on a 16-QAM signal affected by a 3-taps channel. Without surprise, the CNN model trained to minimize the MSE criterion achieves exactly the same performance as a classical MMSE filter (see Figure 4.8). This result stems from the rigorous translation of the standard linear equaliser into the strictly equivalent CNN structure.

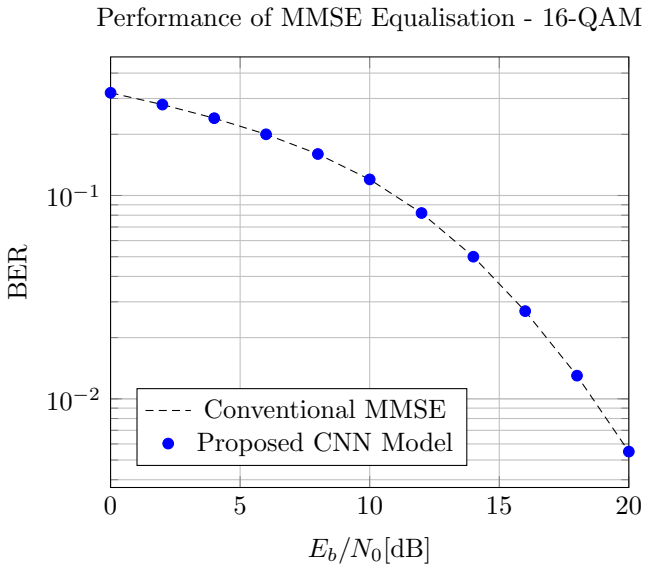


Figure 4.8: Performance of the proposed CNN-based Multi-Path Equaliser over a 3-Tap Channel

4.1.5 Perspectives

While this section demonstrated the translation of some conventional equalisation algorithms into their corresponding NN structures, other equalisation algorithm could be used as a starting point for more complex NN structures. As an example the *Direct Feedback Equaliser* (DFE) equaliser which uses previously equalised samples to better equalise current sample could easily lends itself to an RNN-based implementation. More compute intensive MLSE equaliser and the associated Viterbi trellis algorithm could also be described as NN structure. Furthermore, one could imagine to use DNN structure so as to learn efficient approximation of these very costly types of equaliser although at a cost of a reduced interpretability. An example of such an approach is the *DeepRX* model [126] which use a complete end-to-end receiver using a deep CNN to executes a complete receiver pipeline including channel estimation and equalisation.

4.2 Designing a Neural Network Based Detection System

The step involved in the demodulation of a digital modulation, *e.g.* M-QAM, is typically a decision problem (at least when faced with a hard demodulation process) which are known to be efficiently solved using NN [134, 135]. Indeed, a formal neuron associated with an activation function expresses a non-linearity over an hyper-plane that can then be used to define decision boundaries as will be further described in the following sections. This section describes a minimal NN architecture solving the decision problem faced in a M-QAM demodulator so as to associate the received samples to the most probable symbols within the complex plane of the constellation, and then to the corresponding binary code-words. This minimal model is derived from the theoretical demodulation process and offers a particularly low-complexity NN-based solution to

this problem. Finally, the question of NN-based soft demodulation process is briefly addressed, so as to enable soft FEC decoding procedures as will be studied in Chapter 5.

4.2.1 QAM Demodulator - Naive approach

In this section, a NN-based solution for demodulating a complex signal from a QAM of arbitrary order is described. To demodulate a QAM of order M, a naive algorithm calculates the Euclidean distance between the received IQ sample and the different symbols in the constellation. At first, the aim is here to study a NN-based algorithm that offers an optimal level of performance, and if possible with a lower complexity to that of the naive algorithm.

Quadrature Amplitude Modulation

As briefly introduced in Section 3.1.4, QAM is a form of digital modulation which modulate the amplitudes of a carrier and the same carrier in quadrature based on the information carried by two input channels commonly referred to as I (In-phase) and Q (Quadrature) channels. To illustrate, let's take the example of a rectangular QAM of order 16, whose constellation is shown in Figure 4.9. The amplitude of the in-phase channel (I) is shown on the horizontal axis and the amplitude of the quadrature channel (Q) is shown on the vertical axis. It can be seen that each channel can be modulated using 4 discrete amplitude levels: $\{-3, -1, +1, +3\}$. This results in 16 combinations of amplitudes of the I and Q channels commonly referred to as symbols, each of which encode a 4 bits word. As an example, if one wishes to transmit the binary sequence '0110', the amplitude of the in-phase carrier will be modulated by a factor of -1 and the amplitude of the quadrature carrier, by a factor of -3.

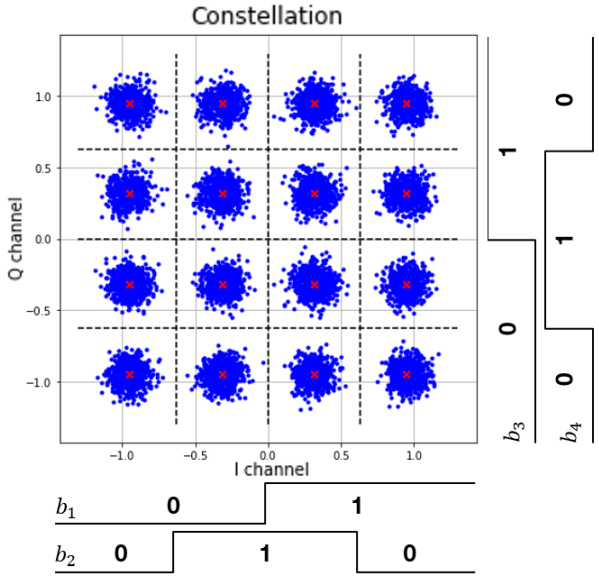


Figure 4.9: Constellation of a 16th Order Gray Mapped Quadrature Amplitude Modulation

Architecture of the Demodulator Neural Network

In the case of a 16-QAM there are six boundaries allowing to distinguish the 16 symbols, as described in the Figure 4.9. A NN is known to be efficient to solve the previous decision problem [134, 135]. Indeed, a formal neuron associated with an activation function such as a sigmoid or a ReLU expresses a non-linearity over an hyper-plane.

Upon reception of the transmitted symbols, the received samples must be associated back to the corresponding binary words, in the best possible way w.r.t the transmission noise. The input data of the demodulator are the amplitudes of the I and Q channels after equalisation

(perfect equalisation and synchronisation are considered here, *i.e.* only gaussian noise remains). We want the NN to associate an input IQ pair with the corresponding binary word. If we take the constellation diagram of a rectangular 16-QAM modulation from Figure 4.9, we can see that we can easily square the space by defining 6 decision boundaries allowing us to separate the 16 symbols of the constellation without ambiguity (see Figure 4.10). The first layer of the NN must be able to differentiate these 16 possible symbols. From Figure 4.10, we can see that a number of neurons certainly allowing to distinguish the M possible symbols of a QAM modulation of order M, corresponds to the number of decision boundaries present. The number of decision boundaries of a QAM of order M is given by:

$$N = 2\left(\sqrt{M} - 1\right) \tag{4.25}$$

In the case of a 16-QAM, we can see that there are 6 decision boundaries, hence we use 6 neurons on the first layer (at least) in order to distinguish the 16 possible symbols (see Figure 4.10).

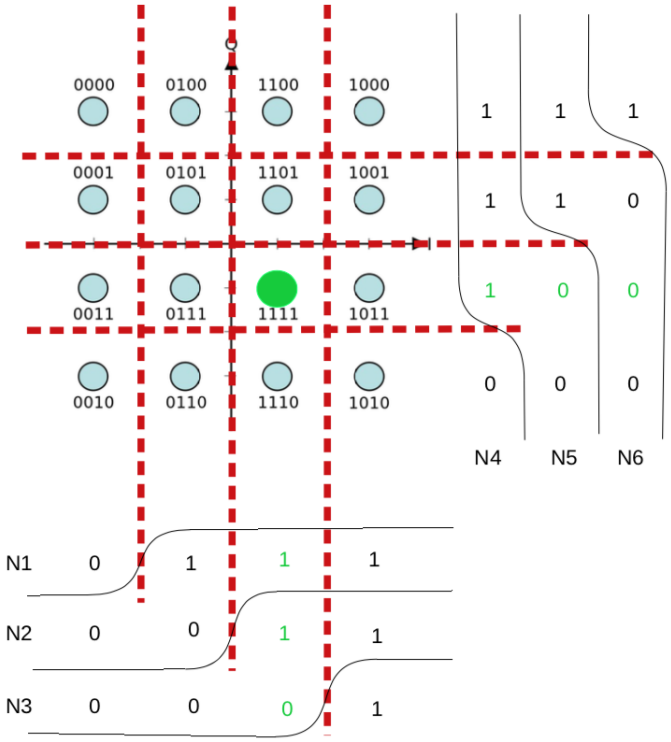


Figure 4.10: The QAM Demodulation - Neural Network Decision Problem Perspective
In red, ideal decision boundaries for a QAM modulation of order 16 - In black, illustration of possible activation functions for the 6 neurons of the first layer - In green, an example of I and Q inputs belonging to the decision zone of the word '1111' leads to the activation of neurons N1, N2 and N4, while the 3 other neurons remain idle. Each symbol is associated with a unique combination of activations of the 6 neurons

The six neurons of this first layer can be configured as shown on Figure 4.10 (considering non normalised constellation *i.e.* symbols with $\{-3, -1, +1, +3\}$ levels):

$$\begin{cases} n_1(i,q) = \sigma(1 \times i + 0 \times q + 2) \\ n_2(i,q) = \sigma(1 \times i + 0 \times q + 0) \\ n_3(i,q) = \sigma(1 \times i + 0 \times q - 2) \\ n_4(i,q) = \sigma(0 \times i + 1 \times q + 2) \\ n_5(i,q) = \sigma(0 \times i + 1 \times q + 0) \\ n_6(i,q) = \sigma(0 \times i + 1 \times q - 2) \end{cases} \tag{4.26}$$

The last layer of the NN aims to format the result of the calculation according to the type of output desired. In our case, we wish to have in output the word, in bits, which was determined

by the NN demodulator. The output layer therefore necessarily has K neurons, K being the number of bits used to encode a word in an M-order QAM modulation:

$$K = \log_2(M) \tag{4.27}$$

For example, for a 16-QAM modulation, words are coded on $\log_2(16) = 4$ bits. The output layer of the NN of a 16-QAM demodulator must therefore use at least 4 neurons.

Other output formats could have been chosen, such as the *one-hot* encoding, which consists in having as many outputs as the number of different classes, *i.e.* one output per constellation symbol. This type of encoding was not chosen because it becomes inefficient when considering higher modulation orders.

Under the chosen encoding and following the decision boundaries of the first layer, as presented in Figure 4.10, the second layer could combine the outputs $\boldsymbol{n} = (n_1, \ldots, n_6)$ of the first layer to form the binary word as follow:

$$\begin{cases} b_1(\boldsymbol{n}) = 0 \times n_1 + 1 \times n_2 + 0 \times n_3 + 0 \times n_4 + 0 \times n_5 + 0 \times n_6 + 0 \\ b_2(\boldsymbol{n}) = 1 \times n_1 + 0 \times n_2 - 1 \times n_3 + 0 \times n_4 + 0 \times n_5 + 0 \times n_6 + 0 \\ b_3(\boldsymbol{n}) = 0 \times n_1 + 0 \times n_2 + 0 \times n_3 + 0 \times n_4 + 1 \times n_5 + 0 \times n_6 + 0 \\ b_4(\boldsymbol{n}) = 0 \times n_1 + 0 \times n_2 + 0 \times n_3 + 1 \times n_4 + 0 \times n_5 - 1 \times n_6 + 0 \end{cases} \tag{4.28}$$

Figure 4.11 describe a NN-based demodulator architecture built following the above design rules.

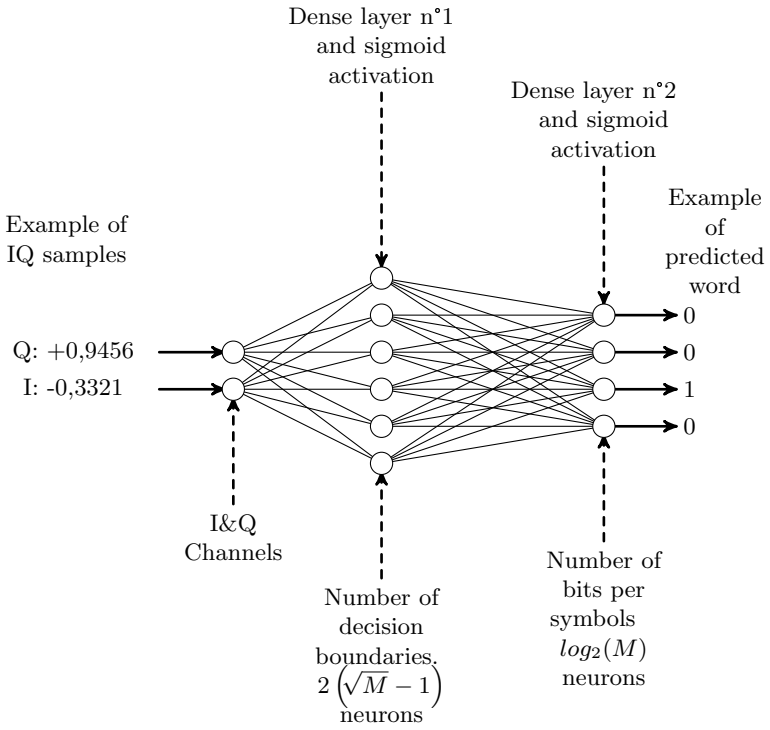


Figure 4.11: Naive Neural Network Based QAM Demodulator
Example of a 16-QAM demodulator with 6 neurons on the input layer and 4 neurons on the output layer. Sigmoid activation functions are used due to the binary nature of the problem.

Demodulator Weights Learning using Gradient Descent

Although such a simple model can be configured manually as demonstrated above, one can apply a supervised learning procedure and the gradient descent method to learn them. IQ samples noised under AWGN are thus provided to the NN as inputs while the corresponding

symbols are used as target prediction labels. The SGD algorithm is then expected to iteratively update the model weight so as to reduce the error between predicted and theoretical symbols. The presence of noise plays a very important role in that process so as to ensure that the model

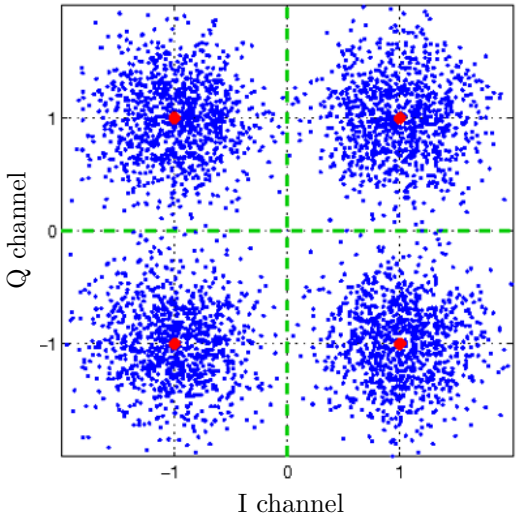


Figure 4.12: Optimal 4-QAM Decision Boundaries
Gaussian noise and equally likely symbols are considered.

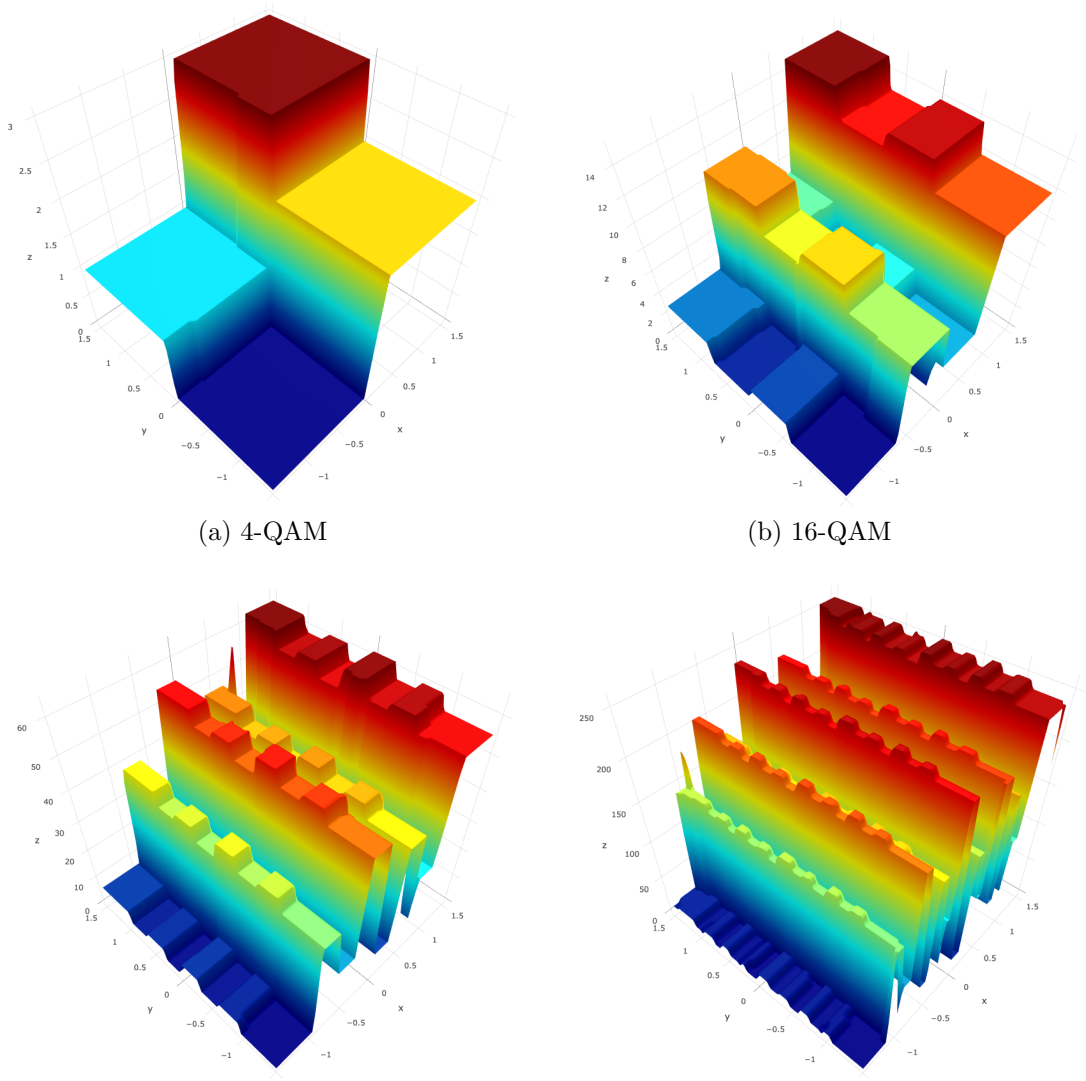
select decision boundaries that minimise the error rate. As shown on Figure 4.12, if the model only sees samples corresponding to the exact symbols (red dots) during training it might choose decision boundaries that does not generalise well under a noisy channel. Still, certain errors are irreducible errors from a pure demodulation perspective (*i.e.* without considering any error correction mechanisms) in the sense that the received data belong to what should be the optimal decision area of a given symbol but is associated to another symbol label. As such, using a noise level too important during training reduce the sample efficiency of the learning algorithm, *i.e* increase the needed volume of training samples and, subsequently, the convergence time. Indeed, irreducible errors provide no useful information to the training, or worse, provide deleterious information. For example, an IQ sample initially belonging to the lower left quadrant of the constellation (corresponding to the word '00') but noised in such a way that it belongs to the upper right quadrant upon reception (corresponding to the word '11'), hinders the learning and generalisation of the problem by the NN because it is labelled '00' but belongs to the decision zone where the network should ideally return the result '11'. The learned demodulator for QAM order comprised between 4 and 256 are provided on Figure 4.13.

4.2.2 Low Complexity Neural Network Based QAM Demodulator

The previous model, is naively build based on the number of decision boundaries of the considered constellation and the number of output bits per word. Obviously, several simplifications can be noted so as to propose a lower complexity NN for M-QAM demodulation.

The NN should associate to a sample the corresponding binary code-word. Therefore, the NN need to solve both decision and demapping problems. The simplified model in the case of a 16-QAM, as proposed in Figure 4.14, is based on the following observations:

- The same demodulation operations can be applied independently to each sample of a batch of inputs. For reasons similar to those described in Section 4.1, a CNN with kernels of the size of one sample is particularly well-suited in this case and allow for an efficient parallel processing of samples blocks. Their shift invariant architecture based on shared weights drastically reduce the number of trainable parameters to a constant value, independent of the number of samples to process.



(a) 4-QAM

(b) 16-QAM

(c) 64-QAM

(d) 256-QAM

Figure 4.13: Learned M-QAM Demodulators
Visualisation of the decision boundaries as learned by the neural networks for different modulation orders. The coordinates (x, y) correspond to the value of the IQ pair given as input to the network and the value in z corresponds to the value of the word predicted (in decimal) by the latter - In the case of successful learning, we expect to observe the most regular grid possible, corresponding to the constellation considered. *N.B.*: The arrangement of the words in the constellation corresponds to a *Gray* mapping, hence the observation of very clear borders passing from low value words to high value words.

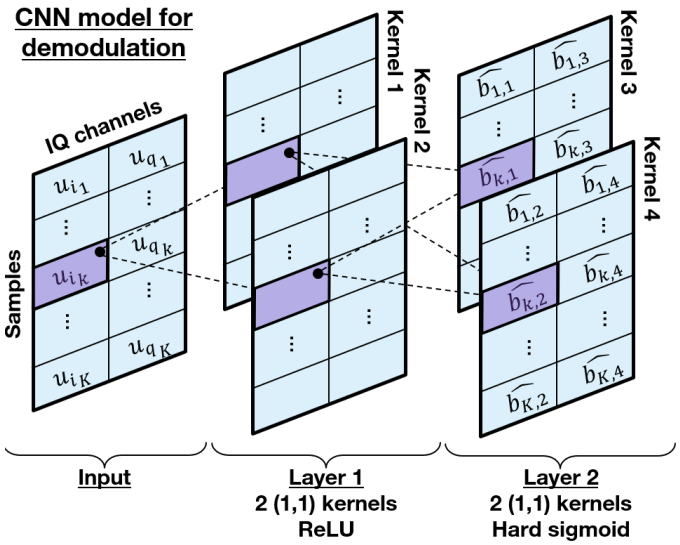


Figure 4.14: Proposed CNN Model for 16-QAM Demodulation
The proposed model leverages properties of CNN to perform the same processing on both I and Q channels and on all samples. It thus uses only 10 shared parameters.

- The NN carries out both decision and demapping tasks. Following a similar approach to the one proposed in [136], a bit-level decision process is considered instead of symbol-level decision process followed by a symbol-to-bit demapping. The objective of the NN is to approximate the functions stated as b_1,b_2,b_3 and b_4 in Figure 4.9. Considering the usual monotonic activation functions offered by the majority of NN frameworks, the NN needs at least two layers to compute the desired functions, as b_2 and b_4 aren't monotonic:
  - Regarding the input layer, the minimum number of kernels corresponds to the number of decisions boundaries associated to the *Least Significant Bit* (LSB) on each channel (namely b_2 and b_4). Consequently, four non-linear kernels are needed ($2^{\log_2(M)/2}$ in the general case of a M-QAM). Considering computational power constraints, ReLU activation is chosen. All the decision boundaries can be expressed as a combination of the four kernels by the output layer.
  - The output of the NN needs to represent the values of the four bits associated to each symbol. As a result, the output layer needs four kernels ($\log_2(M)$ for a M-QAM). The outputs representing binary values, a sigmoid activation function is particularly well-suited.
- Under the assumption that the samples are perfectly equalised, I and Q channels can be processed separately with the same operations, therefore dividing the number and size of the aforementioned kernels by two.

The architecture of this model is scalable to any QAM order by following the number of parameters described in Table 4.1.

The choice of a hard sigmoid instead of a sigmoid activation on the output layer of the NN is proposed to lower the computational complexity. As shown in equation (4.29), the hard sigmoid requires at most two comparisons, one addition and one multiplication.

$$\sigma_{\text{hard}}(x) = \begin{cases} 0 & \text{if } x < -2.5 \\ 1 & \text{if } x > \quad 2.5 \\ 0.2x + 0.5 & \text{otherwise} \end{cases} \tag{4.29}$$

As a reference, a demodulator based on minimal Euclidean distance needs $3M$ additions, $2M$ multiplications and $M-1$ comparisons, with M the order of the QAM modulation [136]. Table 4.2 presents a comparison in terms of computational complexity between the regular demodulator

Table 4.1: Minimal Neural Network Model Architecture Proposed for M-QAM Demodulation

CONVOLUTIONAL LAYER 1	
Input matrix dimensions	$(K, 2)$
Kernel number and properties	$n_1 = 2^{\log_2(M)/2-1}$, size $(1,1)$, stride $(1,1)$
Activation	ReLU
Output tensor dimensions	$(K, 2, n_1)$
CONVOLUTIONAL LAYER 2	
Kernel number and properties	$n_2 = \log_2(M)/2$, size $(1,1)$, stride $(1,1)$
Activation	Hard sigmoid
Output tensor dimensions	$(K, 2, n_2)$
FLATTEN LAYER	
Output vector dimension	$K\log_2(M)$

and the proposed NN model. One can see that the NN outperforms, in terms of complexity, the proposed reference demodulator.

Table 4.2: Complexity Comparison for Different Demodulation Schemes

QAM Order (M)	**Real Multiplications**			**Real Additions**			**Real Comparisons**		
	Regular (2M)	**NN Model**	**Improv. (%)**	**Regular (3M)**	**NN Model**	**Improv. (%)**	**Regular (M-1)**	**NN Model**	**Improv. (%)**
4	8	6	25	12	6	50	3	6	-50
16	32	16	50	48	12	75	15	12	20
64	128	38	70	192	20	90	63	20	68
256	512	88	83	768	32	96	255	32	87
1024	2048	202	90	3072	52	98	1023	52	95

Configuration and Performance of the Neural Network Demodulator

As pointed out earlier, the application of a learning procedure to the configuration of such a demodulator is not necessarily interesting for several reasons. The first is that demodulation is a deterministic process which is not expected to change over time, as may be the case with equalisation. The second is that since the model under consideration is not very complex, a valid configuration of its parameters can easily be found analytically, as we will demonstrate in this section. One of the main advantages of defining such a demodulator as a NN model is that it allows, thanks to the differentiability of the structure, the end-to-end learning of communication schemes, which may possibly involve conventional modulations. For example, suppose we want to optimise an equalisation scheme based on a BER metric calculated following demodulation. In such a situation, the (possibly non-trainable) demodulator model must necessarily be differentiable in order to allow the propagation of the gradient information to the equaliser weights, upstream of the demodulator.

In the case of 16-QAM, one possible configuration of the weights and bias that is optimal with regard to the theoretical decision boundaries is proposed[5]:

$$\begin{aligned} \text{Layer}_1 &\begin{cases} \boldsymbol{w}_1 = (-1) & c_1 = 0 \\ \boldsymbol{w}_2 = (1) & c_2 = 0 \end{cases} \\ \text{Layer}_2 &\begin{cases} \boldsymbol{w}_3 = (-1 \quad 1) & c_3 = 0 \\ \boldsymbol{w}_4 = (-1 \quad -1) & c_4 = 2\delta \end{cases} \end{aligned} \tag{4.30}$$

where $\boldsymbol{w}_i$ and c_i are respectively the weights vectors and bias of the i-th kernel (following numbering of Figure 4.14). 2δ corresponds to the inter-symbol distance of the considered constellation.

[5]Proposed configuration is not unique and assume a hard demodulation process with a rounding of the output values to either zero or one. Higher dynamic of the output layer parameters might be used to approximate the step function with hard sigmoid activation and avoid rounding the outputs.

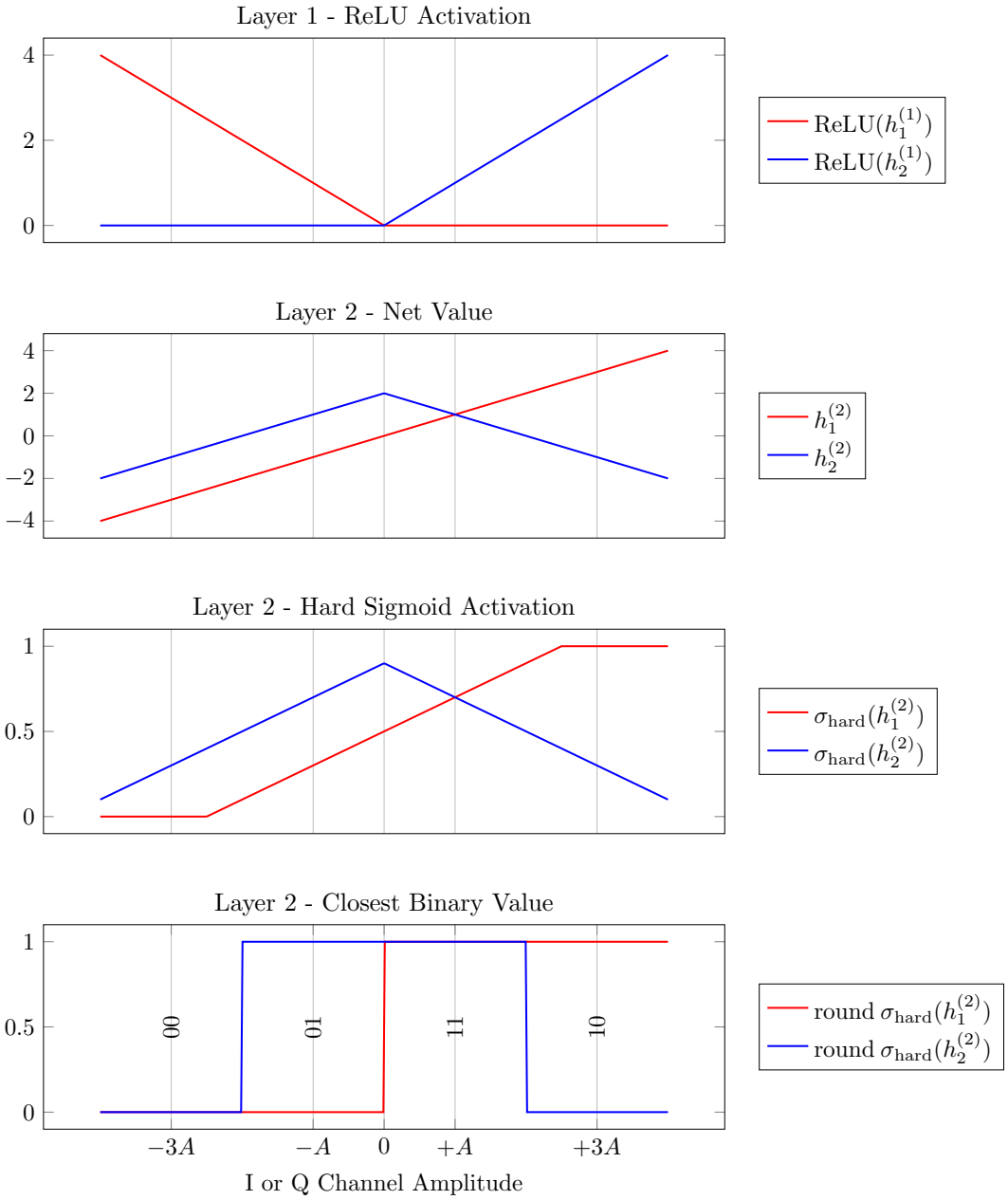


Figure 4.15: Hard Demodulation of 16-QAM Using the Proposed CNN Model
The above figure presents the value of the layers outputs at different level of the NN with regard to the I or Q channel input amplitude. The first layer use ReLU activation to express piece-wise linear function while the second layer combine them so as to ensure the correct sign of the outputs bits with respect to the input value. The hard sigmoid activation of the second layer ensures output values between 0 and 1. The final plot shows the value rounded to the closest integer of the output, demonstrating the validity of the proposed demodulation process.

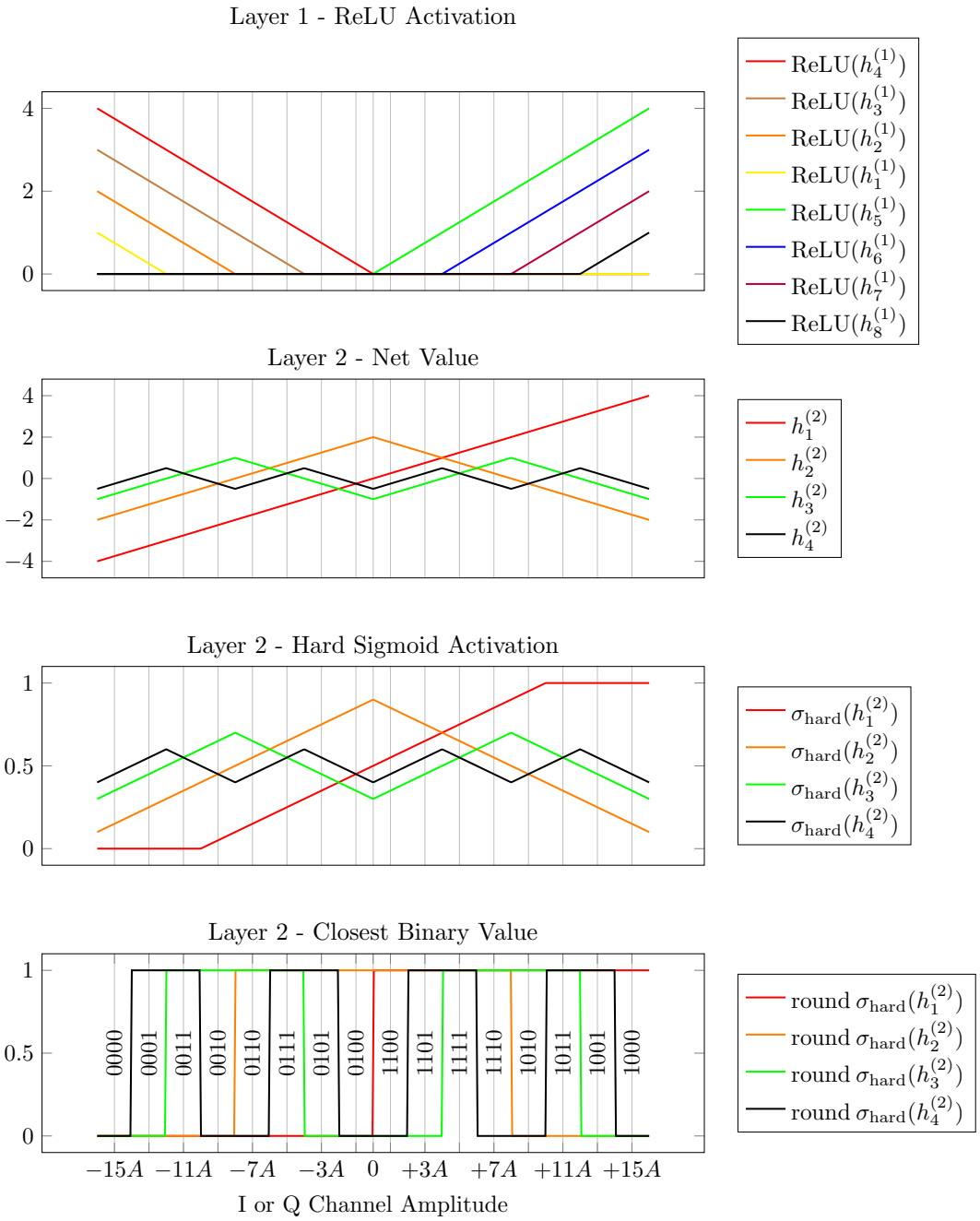


Figure 4.16: Hard Demodulation of 256-QAM Using the Proposed CNN Model
The configuration of the M-QAM demodulator simply consists in positioning and combining piecewise linear functions. In this configuration, the Layer 2 net value highlights the greater robustness of some bits - typically the *Most Significant Bit* (MSB) - compared to others - typically the LSB. This reminds of the idea of soft demodulation and LLR.

$\delta = 1/\sqrt{10}$ in the case of a 16-QAM with a power normalised to one. Such a configuration is simply obtained by the composition of piece-wise linear functions obtained using the first NN layer with a ReLU activation. A visual support of such a method is proposed in Figure 4.15 in the case of 16-QAM and Figure 4.16 in the case of 256-QAM.

The performance of the proposed model are represented on Figure 4.17. Without much surprise because of the analytical definition of the proposed model, it reaches theoretical optimal BER performance[6] over AWGN channel for different QAM orders with the appropriate configurations of the NN.

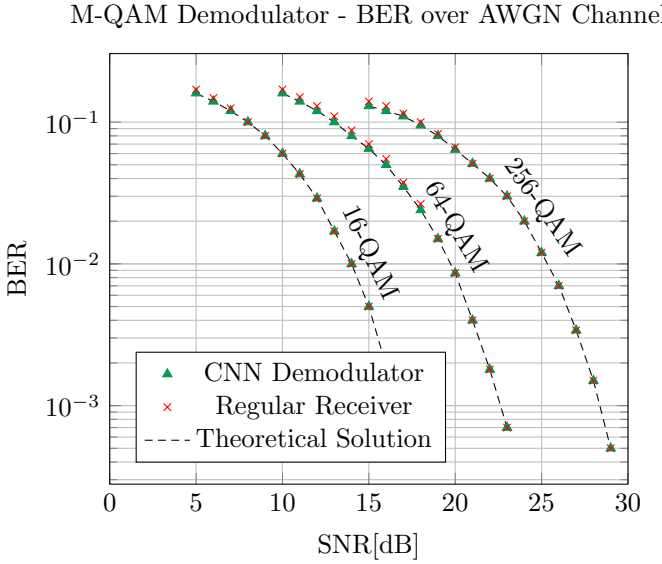


Figure 4.17: Comparison of the BER Demodulation Performance
A regular, minimal Euclidean distance based, and the proposed CNN based demodulators are compared under a simulated AWGN channel for different QAM orders.

As shown in Figure 4.15 for the 16-QAM and even more on Figure 4.16 for the 256-QAM, the more we consider a bit of low importance, the less important the dynamics of the NN output, which highlights the lower robustness of these bits and the lower confidence we can place in the demodulation results for them. These considerations lead us to the notion of a soft demodulation process and the associated probabilistic view of the demodulation output. We will show in the next section that the proposed model can, with only minor modifications, produce soft information.

Soft Demodulation

Hard demodulation is particularly computationally efficient, yet it hinders potential performance gain from a subsequent FEC decoder as explained in Section 3.1.7. Instead of first hard demodulating the received sample and providing the demodulated bits to the FEC decoder, soft demodulation aims at providing to the FEC decoder probabilistic information on the received bits. Such probabilistic information usually takes the form of a so-called LLR, defined for each

[6]Optimal BER over AWGN channel is computed considering the nearest neighbour approximation and Gray mapping:

$$\mathrm{BER}_{\mathrm{M-QAM}} \approx \frac{\sqrt{M}-1}{\sqrt{M}\log_2\sqrt{M}} Q\left(\sqrt{\frac{6\log_2(\sqrt{M})E_b}{(M-1)N_0}}\right) \tag{4.31}$$

where the Q function is defined as:

$$Q(x) = \frac{1}{\sqrt{2\pi}}\int_x^{+\infty} exp\left(-\frac{y^2}{2}\right)dy = \frac{1}{2}\mathrm{erfc}\left(\frac{x}{\sqrt{2}}\right) \tag{4.32}$$

where $\mathrm{erfc}(x)$ is the complementary error function.

bit of a word $\boldsymbol{x}$ as:

$$\boxed{\lambda_i(y) = \log \frac{\mathbb{P}\{x_i = 0|y\}}{\mathbb{P}\{x_i = 1|y\}}} \tag{4.33}$$

where x_i is the value corresponding to the i-th bit of the transmitted word $\boldsymbol{x}$ and y is the received sample (usually complex).

Lets take as a warm-up example the case of a BPSK modulation represented on Figure 4.18, where the input word is constituted of a single bit of information that can take on either the value $x = 0$ or $x = 1$. The BPSK symbol s_1 is associated to word 0, while the symbol s_2 is associated to the word 1. The BPSK symbols s_1 and s_2 are respectively mapped onto the real axis with amplitude $+A$ and $-A$. An AWGN channel is considered with noise power σ^2. The LLR calculation associated to the received sample y in such a situation is straightforward:

$$\begin{aligned}
\lambda(y) &= \log \frac{\mathbb{P}\{x = 0|y\}}{\mathbb{P}\{x = 1|y\}} \\
&= \log \frac{\mathbb{P}\{s = s_1|y\}}{\mathbb{P}\{s = s_2|y\}} \\
&= \log \left[\frac{\mathbb{P}\{y|s = s_1\}\,\mathbb{P}\{s = s_1\}}{\mathbb{P}\{y\}} \times \frac{\mathbb{P}\{y\}}{\mathbb{P}\{y|s = s_2\}\,\mathbb{P}\{s = s_2\}} \right] \\
&= \log \frac{\mathbb{P}\{y|s = s_1\}\,\mathbb{P}\{s = s_1\}}{\mathbb{P}\{y|s = s_2\}\,\mathbb{P}\{s = s_2\}}
\end{aligned} \tag{4.34}$$

As both symbols are equally likely, the above expression simplify to:

$$\lambda(y) = \log \frac{\mathbb{P}\{y|s = s_1\}}{\mathbb{P}\{y|s = s_2\}} \tag{4.35}$$

where $\mathbb{P}\{y|s\}$ is the channel conditional probability distribution which equals, in the case of an AWGN channel, to:

$$\mathbb{P}\{y|s\} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y \pm A)^2}{2\sigma^2}} \tag{4.36}$$

The LLR expression for BPSK symbols under AWGN channel is thus defined as:

$$\boxed{\lambda(y) = \frac{(y + A)^2}{2\sigma^2} - \frac{(y - A)^2}{2\sigma^2} = \frac{2Ay}{\sigma^2}} \tag{4.37}$$

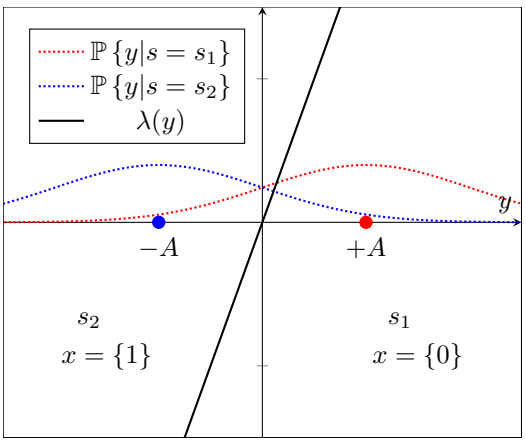


Figure 4.18: Soft Demodulation of a BPSK Under AWGN Channel
The noise power of the AWGN channel is set to half the inter-symbol distance, *i.e.* $\sigma^2 = A$.

A similar approach can be taken for higher order modulation. In this section, we want to investigate how the proposed demodulator model could be used for soft demodulation of QAM

signals. For the sake of simplicity, we will consider the case of a 16-QAM. As described above, the processing applied to the I and Q channels for QAM demodulation is, under perfect equalisation and synchronisation, strictly equivalent. We will therefore study only one channel, as the results can easily be extended to the two-channel case.

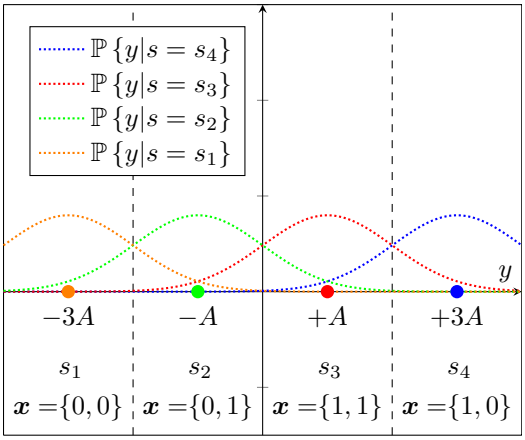


Figure 4.19: Single Channel 16-QAM Soft Demodulation under AWGN
In the present example, the considered noise level of the AWGN channel is equivalent to the half inter-symbol distance, *i.e.* $\sigma^2 = A$.

Let us first define the expression of the LLR in the case of a 16-QAM modulation. Under such a modulation the (half-)words associated to one of the two I and Q channels contain two bits and the LLR can thus be seen as a vector function:

$$\boldsymbol{\lambda}(y) = \begin{pmatrix} \lambda_1(y) \\ \lambda_2(y) \end{pmatrix} = \begin{pmatrix} \log\frac{\mathbb{P}\{x_1=0|y\}}{\mathbb{P}\{x_1=1|y\}} \\ \log\frac{\mathbb{P}\{x_2=0|y\}}{\mathbb{P}\{x_2=1|y\}} \end{pmatrix} = \begin{pmatrix} \log\frac{\mathbb{P}\{y|x_1=0\}}{\mathbb{P}\{y|x_1=1\}} \\ \log\frac{\mathbb{P}\{y|x_2=0\}}{\mathbb{P}\{y|x_2=1\}} \end{pmatrix} = \begin{pmatrix} \log\frac{\mathbb{P}\{y|s=s_1\}+\mathbb{P}\{y|s=s_2\}}{\mathbb{P}\{y|s=s_3\}+\mathbb{P}\{y|s=s_4\}} \\ \log\frac{\mathbb{P}\{y|s=s_1\}+\mathbb{P}\{y|s=s_4\}}{\mathbb{P}\{y|s=s_2\}+\mathbb{P}\{y|s=s_3\}} \end{pmatrix} \tag{4.38}$$

As shown on Figure 4.19, we have, under AWGN channel:

$$\boxed{\lambda_1(y) = \log\frac{e^{-\frac{(y+3A)^2}{2\sigma^2}} + e^{-\frac{(y+A)^2}{2\sigma^2}}}{e^{-\frac{(y-A)^2}{2\sigma^2}} + e^{-\frac{(y-3A)^2}{2\sigma^2}}}} \tag{4.39}$$

and:

$$\boxed{\lambda_2(y) = \log\frac{e^{-\frac{(y+3A)^2}{2\sigma^2}} + e^{-\frac{(y-3A)^2}{2\sigma^2}}}{e^{-\frac{(y+A)^2}{2\sigma^2}} + e^{-\frac{(y-A)^2}{2\sigma^2}}}} \tag{4.40}$$

Such functions can be approximated by piece-wise linear functions:

$$\lambda_1(y) \approx \begin{cases} \log\frac{e^{-\frac{(y+3A)^2}{2\sigma^2}}}{e^{-\frac{(y-A)^2}{2\sigma^2}}} = \frac{-4A(y+A)}{\sigma^2} & \forall\, y \ll -2A \\ \log\frac{e^{-\frac{(y+A)^2}{2\sigma^2}}}{e^{-\frac{(y-A)^2}{2\sigma^2}}} = \frac{-2Ay}{\sigma^2} & \forall\, -2A \gg y \ll 2A \\ \log\frac{e^{-\frac{(y+A)^2}{2\sigma^2}}}{e^{-\frac{(y-3A)^2}{2\sigma^2}}} = \frac{-4A(y-A)}{\sigma^2} & \forall\, y \ll -2A \end{cases} \tag{4.41}$$

and:

$$\lambda_2(y) \approx \begin{cases} \log\frac{e^{-\frac{(y+3A)^2}{2\sigma^2}}}{e^{-\frac{(y+A)^2}{2\sigma^2}}} = \frac{-2A(y+2A)}{\sigma^2} & \forall\, y \ll 0 \\ \log\frac{e^{-\frac{(y-3A)^2}{2\sigma^2}}}{e^{-\frac{(y-A)^2}{2\sigma^2}}} = \frac{-2A(y-2A)}{\sigma^2} & \forall\, y \gg 0 \end{cases} \tag{4.42}$$

Figure 4.20 shows the exact and approximate values of the LLR calculated for the 16-QAM under AWGN channel. First, it can be noted that the piece-wise approximations are almost equal to the exact value, except when considering values too close to the theoretical decision boundaries. Another interesting fact shown in this figure is that the net output values of the second layer of the proposed CNN demodulator form a good first approximation of the LLR. A more accurate approximation, could simply be obtained by combining more piece-wise linear functions than the one provided by the two ReLU neurons currently used in the first layer of the CNN. In this case, two additional ReLU neurons at the first layer would allow the second layer to express the two inflection points of $\lambda_1(y)$ displayed in $-2A$ and $+2A$. These considerations highlight the fact that, for any QAM order, the proposed CNN can be improved towards soft demodulation simply by adding a few neurons (or kernels in the CNN vocabulary) to the first layer.

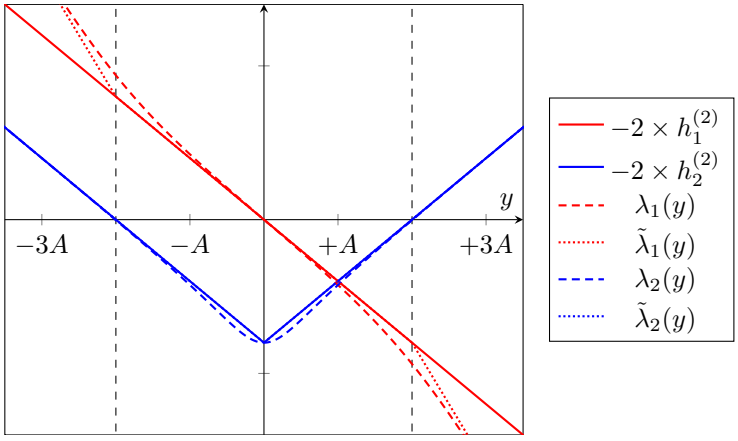


Figure 4.20: LLR function for a 16-QAM Modulation under AWGN
This figure present the exact and approximate LLR values for both bits of the (half-)word of a Gray-mapped 16-QAM modulation (considering $\sigma^2 = A$). The net output values of the layer 2 of the CNN demodulator are also shown, to highlight that they can be used, as is, as a first approximation of an LLR (up to a multiplying constant).

Soft demodulation, as highlighted before, allows to improve the FEC decoding performance. Furthermore, in the case of NN, the ReLU activation used to express soft demodulation allows a better differentiability of the model when compared to sigmoid-like binary activation functions or worse step functions (for exact hard binary demodulation). In fact, ReLU function or one of its related activation functions (*e.g.* leaky ReLU, GeLU, etc.), are almost exclusively used in the intermediate layers of modern NN for this exact reason. As explained above, for a good approximation of the LLR, it is necessary to add some parameters to the model, which can be adjusted by hand in the simplest cases or purely through model training. It should also be noted that, while the expression of LLR is relatively straightforward for an AWGN channel and/or small modulation orders, it may be difficult or impossible to express an analytical form in a more complex scenario. In such situations, the use of a NN and a learning procedure becomes even more interesting, as shown in the article [8] which uses a NN to approximate LLR functions at a lower computational cost.

4.3 Experimentation on a Software Defined Radio Test-Bed

In the previous section, different NN structures for M-QAM equalization and demodulation were presented. Based on the proposed NN models for single-path equalisation and M-QAM demodulation, this section examines the implementation of a functional prototype of a simplistic PHY layer using SDR, thus demonstrating the use of NN structures as an alternative to conventional algorithms. While the previous descriptions were primarily concerned with the structural aspects of the NN models with respect to the targeted algorithms, this section implements the MMSE

learning procedure proposed in Section 4.1.3 for the online learning of an equalisation scheme based on pilot sequences.

4.3.1 System Model and Working Assumptions

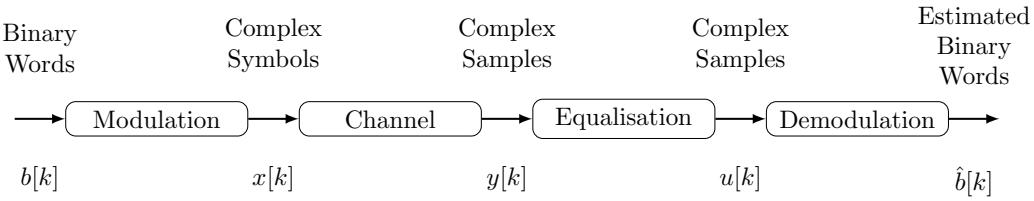


Figure 4.21: Baseband Description of the Considered M-QAM Communication Chain

A simple communication system is considered, as described in Figure 4.21. Binary sequences are modulated using a Gray mapped 16-QAM modulator. A single-path AWGN channel model is adopted, similar to the one described in Section 4.1.3. On the receiver side, the effect of the channel must be mitigated by a single-path equaliser and the original binary code-words recovered thanks to a hard 16-QAM demodulator. Decoding, and especially soft decoding, is not considered in this section but will be extensively discussed in Chapter 5.

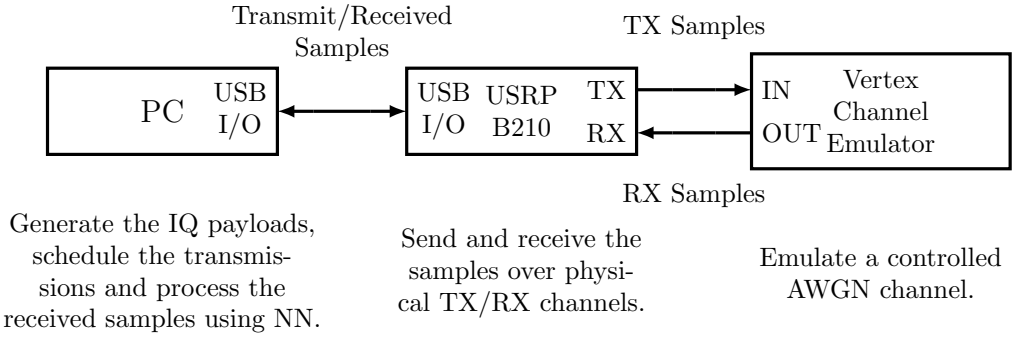


Figure 4.22: Functional Diagram of the Test Bench Implementation

The considered signal, generated and processed by a computer, is constituted of several blocks of symbols. These blocks are transmitted and received back by an Ettus USRP B210 SDR [137] from TX channel to RX channel through a Spirent Vertex channel emulator [138] as shown on Figures 4.22, 4.23 and 4.24. The later adds a controlled amount of Gaussian noise. An unchecked phase shift, considered invariant during the transmission of a block, is also introduced because of the propagation time and the *Local Oscillator* (LO) misalignment and needs to be corrected during the equalisation step. To this end, each block begins with a ZC sequence[7] of length 256. The ZC sequence is followed by a payload of 4096 random 16-QAM symbols. On the receiver side, the signal is sampled with an up-sampling factor of 4. The ZC sequence is used as a pilot sequence to identify payload start and ideal sampling instant as well as channel effects. For each block, the detection and synchronisation of the payload is performed by a regular maximal correlation based algorithm, but a NN architecture could be envisioned to perform such task. After down-sampling, a ZC sequence of size 256 and a payload of size 4096 are thus recovered. The objective of the proposed solution is, for each block, to learn in an online fashion the channel effects based on the comparison of the received down-sampled ZC sequence with the theoretical one, and use this knowledge to equalise and demodulate the down-sampled payload.

[7] ZC sequences are *Constant Amplitude Zero Auto-Correlation* (CAZAC) wave-forms that exhibits interesting properties [80, 81]. They are notably used in 3GPP LTE and 5G air interfaces.

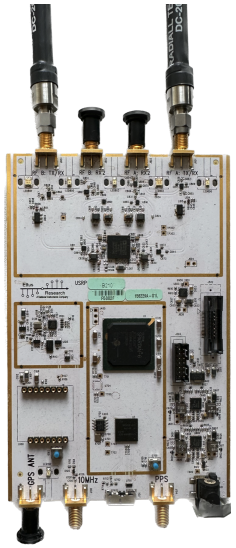


Figure 4.23: Ettus USRP B210



Figure 4.24: Spirent Vertex Channel Emulator

4.3.2 Proposed Neural Network Based Receiver and Experimental Performance

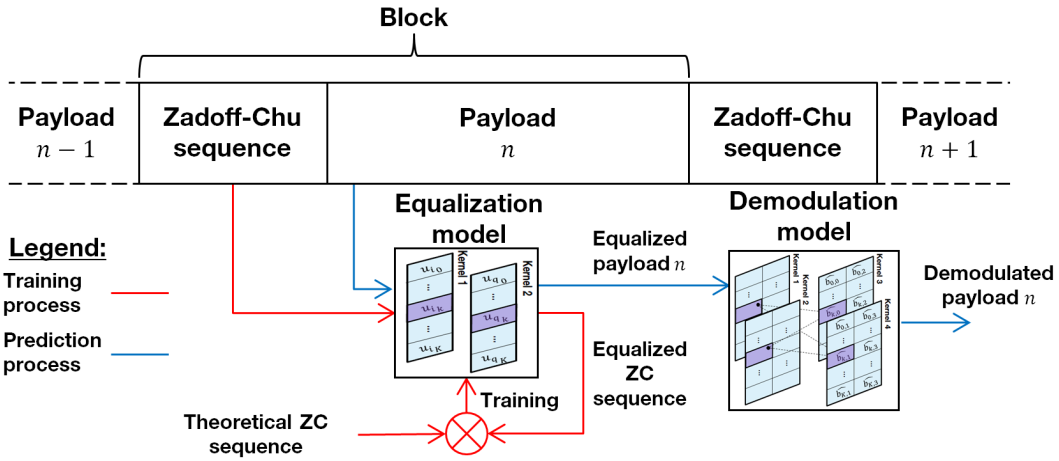


Figure 4.25: Description of the Proposed Processing Chain
The Equalisation model is trained using the received ZC sequence and the theoretical one, known by the receiver. Using the trained equalisation model, the payload is equalised and then demodulated using a predefined demodulation model.

The performances of the NN-based equaliser and demodulator, proposed in Section 4.1.3 and 4.2.2 respectively, are jointly evaluated in this more realistic setting where SDR is used to transmit IQ samples over the physical channel generated by a channel emulator. The parameters of the demodulation model are set following the configuration proposed in Section 4.2.2. Under the hypothesis of block fading, it is necessary to perform online learning of the equalisation model. As described in Figure 4.25 the proposed system works as follow:

- For each block, the equalisation model is trained by comparing the received and theoretical ZC sequences. The 256 samples of the ZC sequences are divided in two parts: two third for training and one third for validation. MSE loss and *ADAM* optimiser with a learning rate of 0.1 are used.
- After learning, the 4096 IQ samples of the payload are corrected using the newly trained

equalisation model before being fed to the demodulation model.

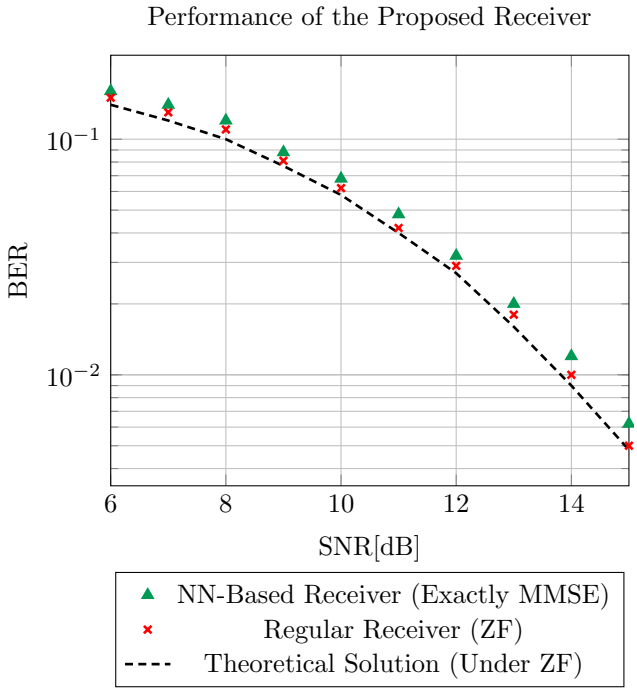


Figure 4.26: Comparison of the Performance of a Regular Receiver and the Proposed CNN-Based Receiver over an Emulated AWGN Channel

As described in Figure 4.26, this simple system achieves performance close to the theoretical performance of a system under assumptions of perfect synchronisation and no channel coding. As a comparison, the regular receiver estimates the channel parameters by computing the correlation between the received ZC sequence and the theoretical one to perform single-path equalisation. A minimal Euclidean distance algorithm is then used for demodulation. One can see a slight degradation of the BER performance of the MMSE NN model compared to that of a regular ZF receiver. Indeed, MMSE equalisers are known to converge toward a biased solution [139], that can be expressed in the considered case of single-path channel by:

$$w_{\text{MMSE}} = \frac{h^*}{|h|^2 + \frac{\mathbb{E}(|n_k|^2)}{\mathbb{E}(|x_k|^2)}} \tag{4.43}$$

where w_{MMSE} is the complex scalar coefficient of the equaliser that minimises the L_2 loss function between the equaliser output and the expected one (which can be expressed as a (2×2) real-valued matrix transformation). h is the complex scalar coefficient of the single-path channel and $\frac{\mathbb{E}(|n_k|^2)}{\mathbb{E}(|x_k|^2)}$ is the noise-to-input complex signals power ratio (i.e. inverse of the SNR). The existence of such a learning bias is demonstrated in the simpler case of a real BPSK modulation and adopting the same notations as in the Section 4.1:

Proof. Lets consider a simple channel which multiply the signal $x[k]$ by an attenuation h and add Gaussian noise $w_k \sim \mathcal{N}(0, \sigma^2)$ to the signal:

$$y_k = hx_k + n_k \tag{4.44}$$

A simple equaliser which aims at applying the inverse filter using the following parametric function (which correspond to a single biased neuron with one entry) is considered:

$$u_k = wy_k + b \tag{4.45}$$

By ERM principle (see Chapter 2), and under MSE loss, the ML problem to find the parameters w and b of the equaliser, given a batch of N pilot samples is:

$$\hat{w}, \hat{b} = \underset{w,b}{\operatorname{argmin}}\, \hat{\ell}(w,b) = \underset{w,b}{\operatorname{argmin}}\, \frac{1}{N}\sum_{k=1}^{N} l_k \tag{4.46}$$

where $l_k = (u_k - x_k)^2$ is the loss sample associated to the k^{th} element of the batch.

$$\hat{\ell}(w,b) = \frac{1}{N}\sum_{k=1}^{N} l_k = \frac{1}{N}\sum_{k=1}^{N}(u_k - x_k)^2 = \frac{1}{N}\sum_{k=1}^{N}(wy_k + b - x_k)^2 = \frac{1}{N}\sum_{k=1}^{N}\left[w(hx_k + n_k) + b - x_k\right]^2 \tag{4.47}$$

Since the signal and noise samples are both i.i.d., centred on zero and uncorrelated with each other, the expected empirical risk is defined as follows:

$$\begin{aligned}
\mathbb{E}_{\mathrm{x}_k \mathrm{n}_k}\left\{\hat{\ell}(w,b)\right\} &= \mathbb{E}_{\mathrm{x}_k,\mathrm{n}_k}\left\{\left[w(h\mathrm{x}_k + \mathrm{n}_k) + b - \mathrm{x}_k\right]^2\right\} \\
&= \mathbb{E}\left\{w^2(h\mathrm{x}_k + \mathrm{n}_k)^2 + 2bw(h\mathrm{x}_k + \mathrm{n}_k) - 2\mathrm{x}_k w(h\mathrm{x}_k + \mathrm{n}_k) + b^2 - 2b\mathrm{x}_k + \mathrm{x}_k^2\right\} \\
&= \mathbb{E}\left\{w^2(h\mathrm{x}_k + \mathrm{n}_k)^2 - 2h\mathrm{x}_k^2 w + b^2 + \mathrm{x}_k^2\right\} \\
&= \mathbb{E}\left\{\mathrm{x}_k^2(h^2w^2 - 2hw + 1) + \mathrm{n}_k^2 w^2 + b^2\right\}
\end{aligned} \tag{4.48}$$

Let $\operatorname{Var}\{\mathrm{x}_k\} = m\sigma^2$, with m the SNR and σ^2 the noise variance:

$$\begin{aligned}
\mathbb{E}_{\mathrm{x}_k \mathrm{n}_k}\left\{\hat{\ell}(w,b)\right\} &= \left(h^2w^2 - 2hw + 1\right)\operatorname{Var}\{\mathrm{x_k}\} + w^2\sigma^2 + b^2 \\
&= \left[(mh^2 + 1)w^2 - 2hwm + m\right]\sigma^2 + b^2
\end{aligned} \tag{4.49}$$

We now compute the gradient of the empirical risk:

$$\nabla_{w,b}\mathbb{E}_{\mathrm{x}_k \mathrm{n}_k}\left\{\hat{\ell}(w,b)\right\} = \begin{pmatrix} \frac{\partial \hat{\ell}}{\partial w} \\ \frac{\partial \hat{\ell}}{\partial b} \end{pmatrix} = \begin{pmatrix} \left[2(mh^2 + 1)w - 2mh\right]\sigma^2 \\ 2b \end{pmatrix} \tag{4.50}$$

We compute the Hessian:

$$\nabla^2_{w,b}\mathbb{E}_{\mathrm{x}_k \mathrm{n}_k}\left\{\hat{\ell}(w,b)\right\} = \begin{pmatrix} \frac{\partial^2 \hat{\ell}}{\partial w^2} & \frac{\partial^2 \hat{\ell}}{\partial w \partial b} \\ \frac{\partial^2 \hat{\ell}}{\partial b \partial w} & \frac{\partial^2 \hat{\ell}}{\partial b^2} \end{pmatrix} = \begin{pmatrix} 2(mh^2 + 1)\sigma^2 & 0 \\ 0 & 2 \end{pmatrix} \succ 0 \tag{4.51}$$

The Hessian being a diagonal matrix with positive diagonal entry is thus PD and the expectation of the empirical risk convex. Hence, to find the global minimum of the function we simply need to find the values of w and b where the derivatives equal 0:

$$\begin{cases} \left[2(mh^2 + 1)w - 2mh\right]\sigma^2 = 0 \\ 2b = 0 \end{cases} \iff \begin{cases} \hat{w} = \dfrac{mh}{mh^2 + 1} = \dfrac{h}{h^2 + 1/m} \\ \hat{b} = 0 \end{cases} \tag{4.52}$$

This shows that the SNR has a direct impact on the learned weight as previously described in Equation (4.43) (complex case). More precisely:

$$\boxed{\begin{cases} \underset{m \to \infty}{\operatorname{Lim}}\, \hat{w} = \dfrac{1}{h} = w^\star \\ \underset{m \to 0}{\operatorname{Lim}}\, \hat{w} = 0 \end{cases}} \tag{4.53}$$

In the absence of noise the equaliser converges toward the optimal solution and MMSE solution become equivalent to that of LS. When the noise level increase, the learned solution is biased towards 0. □

As demonstrated above, the MMSE solution to the single-path equalisation problem is a biased solution because of the noise present in the signal. Instead of reducing the error between the equalised sample and the theoretical pilot value, one can consider the inverse problem, *i.e.* minimising the error between the output of the model and the received samples, when its input is the pilot sequence, as in the LS method. The NN then acts as a channel model whose parameters are expected to converge towards the channel parameters. The weight of the model can then be used as an unbiased channel estimate, which could be inverted to perform ZF equalisation. Yet, such an approach does not really fit in the usual NN philosophy where a functional model (*i.e.* the equaliser) is usually directly trained to perform a task (*i.e* the equalisation) without having to act on the parameters of the model at the end of the training phase. Furthermore, while LS estimation and ZF equalisation leads to an optimal solution in the case of single-tap channel, this is not necessarily the case for multi-tap channel where a MMSE equaliser might be more suitable.

4.4 Conclusion of the Chapter

In this chapter we have demonstrated how conventional PHY layer signal processing algorithms can be expressed using equivalent NN structures. In particular low-complexity NN-based linear equalisation and M-QAM demodulation structure were described and studied. The proposed models have been experimentally assessed using SDR based communication chain and a channel emulator, achieving near-optimal performance, and demonstrating simple online learning procedures.

The study of NN complexity is of great importance in the context of future cellular networks for IoT, where it is not always possible to use deep NN structures. Using low-complexity NN offers the advantages of higher efficiency and reduced learning and inference time. It also mitigates the lack of interpretability, and therefore the consequent lack of trustworthiness, often reproached to DNN. The models proposed in this chapter are in this sense compatible with the stringent operational constraints of IoT devices and networks. As already explained in Chapter 2, the expression of all signal processing blocks using NN offers advantages in various ways. First, it allows the use of specific AI hardware accelerators that are particularly interesting for their intrinsic energy and compute efficiency. Furthermore, since the latter are generic, they can be used to run different types of signal processing algorithms based on NN, thus constituting a unified hardware platform. This hardware generality is interesting in terms of the scalability of the PHY layer, but also in terms of the possible mutualisation of hardware in a distributed learning and inference system or in a *Mobile Edge Computing* (MEC) context where the hardware resources of the *Radio Access Network* (RAN) could, in case of inactivity, be allocated to the end-user's AI processing. Moreover, recent versions of the aforementioned hardware, such as Google Edge TPU [57], have been developed to address embedded and low power applications. Hence, they are particularly appealing for IoT devices. As more and more companies develop NN dedicated hardware, the TCO of a NN-based communication systems is expected to decrease. Thus, the more elements of the PHY layer (and beyond) are replaced by NN algorithms, the more one can expect to benefit of the properties described above. Therefore, it is interesting to study NN models for all base-band functionalities and even those that might have a known optimal analytical solution.

Obviously, the use of NN is also a way to enable learning procedures that can be interesting for discovering efficient signal processing solutions as well as for allowing a fast online model adaptation. In this respect, it should be noted that the translation of some conventional algorithms is also interesting not for the direct improvement of said algorithms, but for the implementation of end-to-end learning procedures on communication systems requiring some conventional (possibly non-trainable, but differentiable) blocks in the middle of a globally trainable chain. The expression of all blocks as a differentiable structure is thus a mean of enabling the use of gradient back-propagation algorithms. While learning was briefly addressed in this chapter, notably for the online parameterisation of the equalisation model of Section 4.3, the goal was not to improve over the corresponding conventional methods but rather express an equivalent NN-based structure and corresponding learning algorithm. The following chapter, will look at

the question of using ML algorithm at the PHY more in depth and notably how it can improve the performance of certain algorithms w.r.t the SOTA. One particularly challenging problem of learning algorithms at the PHY is the recurrent use of finite field (*e.g.* binary source) and many other source of non-differentiability as well as the problematic "curse of dimensionality". One such functional block particularly concerned by these problems is the channel (de)coding which will thus be studied in depth in the following chapter.

CHAPTER 5

LEARNING PROCESS AT THE AIR INTERFACE: APPLICATION TO CHANNEL CODING

Contents

5.1 Motivations 107
5.2 An Introduction to Iterative Methods for the Decoding of Linear Block Codes 108
5.2.1 Fundamentals of Linear Block Codes 109
5.2.2 A Step Towards Soft Iterative Decoding 111
5.2.3 Factor Graph Representation of a Linear Block Code 111
5.2.4 Applying Message Passing Algorithm to the Factor Graph of a Code . 112
Sum-Product Rule at a Variable Node - Equality Constraint 113
Sum-Product Rule at a Check Node - Zero-Sum Constraint 114
Belief Propagation Decoding of a Linear Block Codes 116
5.2.5 Degree Distribution Notations 117
5.3 Belief Propagation Decoder as an Efficient Recurrent Neural Network for Short Codes 117
5.3.1 Description of the Neural Belief Propagation Algorithm and its Variants 117
Neural Belief Propagation Decoder 117
Parallel Neural Belief Propagation Decoders 118
5.3.2 Neural Belief Propagation as an Efficient Gated Recurrent Neural Network 118
Variable to Check Nodes Computation 119
Check to Variable Nodes Computation 122
Output Marginalisation 123
Recurrent Neural Network Cell 125
5.4 Learning to Decode - Blind Neural Belief Propagation Decoder . 126
5.4.1 System model 126
5.4.2 An Auto-Encoder Structure to Learn to Decode 126
5.4.3 Model Training and Results 128
Results 129
5.5 Learning to Code - Neural Belief Propagation Auto-Encoder for Linear Block Code Design 130
5.5.1 Learning to Code: Main Challenges 130
Differentiability of the Models 130
Scalability - the "Curse of Code Dimensionality" 130
5.5.2 Linear Block Code Neural Belief Propagation Auto-Encoder 130
Linear Block Code Neural Encoder 130

Gated Neural Belief Propagation Decoder 132
Proposed Auto-Encoder Architecture for Linear Block Code Design . 132
5.5.3 Data-sets, Training and Evaluation Procedures 134
5.5.4 Results 135
(8,4) Code: Illustration of the Proposed Concept 135
(31,16) Codes: Statistical Study of Model Training 137
(31,11) Codes: Auto-encoder Training Procedure and Ablation Study 139
(63,36) and (63,45) Codes: Scalability and Algorithmic Complexity . . 141
Comparison with LDPC and other State-Of-the-Art Codes 142
5.5.5 Discussion and Perspectives 145
5.5.6 Source Code 146
5.6 Conclusion of the Chapter 147

Foreword

Previous chapter described how differentiable graphical structures can be used to represent conventional signal processing algorithms thus enabling the application of learning algorithms and the use of dedicated high efficiency AI accelerator hardware platforms. In this chapter, we go one step further by studying how the use of learning procedure on such trainable signal processing structure can bring significant performance gains.

After the equalisation and (soft) demodulation blocks studied in the previous chapter, channel coding is one of the fundamental block missing in our NN-based communication chain. Moreover, it is an interesting function to study from a NN and ML standpoint, given the many challenges it raises in this respect and which will be detailed in the course of this chapter. At first, this chapter will, following the previously defined methodology, start by describing how one can express conventional decoding algorithm using a differentiable graphical model. Then, we will show how ML techniques can be used to improve the decoding performance. Finally, an end-to-end approach will be used for the joint design of a coding scheme and associated decoder.

5.1 Motivations

Efficient FEC schemes are a key enabler for IoT scenarios and/or *Ultra-Reliable Low-Latency Communications* (URLLC) in next generation communication networks. Such use-cases typically imply the use of short packets and low-complexity and/or fast (de)coding schemes, in line with latency, energy consumption, hardware cost and computational power constraints. Existing FEC codes, such as BCH, *Tail Biting Convolutional Code* (TB-CC), Turbo, Polar or LDPC codes, and their respective decoders, attempt to meet these needs, each with their own advantages and limitations. In late 2018, 3GPP agreed to use Polar codes for the control channels and LDPC for the data channels of the 5G-NR interface for *enhanced Mobile Broad-Band* (eMBB) applications [140]. On the one hand, for high throughput applications, large LDPC codes offer near capacity performances, efficient encoding and decoding with highly parallel decoders based on BP and variants, *e.g. Min-Sum* (MS) [141] or *Offset Min-Sum* (OMS) [142]. On the other hand, Polar codes are proven to achieve capacity on *Binary Symmetric Channel* (BSC) and offer a reasonable decoding complexity for shorter block lengths based on successive cancellation list decoders. While BP decoders offer high performance and efficient decoding for long and sparse codes, they tend to be less efficient for decoding smaller, usually denser, ones. Indeed, for shorter code lengths, the *Parity-Check* (PC) matrix of LDPC codes is, unfortunately, no more sparse. The "Low Density" property is lost and it becomes unavoidable to have short cycles in the PC matrix. Similarly, while successive cancellation list decoders offer high performance and controlled decoding complexity for short polar codes, they tend to become inapplicable when considering larger codes. Therefore, there is currently no efficient unified FEC code and decoder architecture for all code lengths and rate ranges. Such an architecture would be of great interest for next generation communication networks such as 6G networks. This chapter propose different approach to this problematic:

- Improving the performance of a decoder for an already existing coding scheme. This is the approach adopted in Section 5.3, where ML techniques are used to learn to decode a code using a trainable weighted decoding procedure.
- Improving a coding scheme with respect to a given decoder target. This is the method described in Section 5.5 where an end-to-end NN model is used to design a coding scheme *ex nihilo*. This approach is combined with a joint training of the decoder structure to further improve the FEC performance.

AI and ML techniques are deemed to play a major role in 6G networks [5, 143] and they have naturally been applied to the field of channel coding and particularly to the short block length regime. As described in Chapter 3, the advantages of such approaches are mainly two-folds:

- As described in Chapters 2 and 3, the mathematical framework of NN enables extremely efficient, low-power, generic and cost effective hardware implementations such as NPU[9].

Therefore, describing classic telecommunication and signal processing functionalities as NN architectures is a way to benefits from the advantages of these hardware technologies even though the raw algorithmic performance, such as the BER, might not necessarily be improved.

- Partly enabled by the use of NN, the use of training procedure and ML techniques is a way to improve the performance of the signal processing algorithms. The improvements are usually of two natures. On the one hand, by defining new solutions offering a finer processing, tailored to a communication scenario, *e.g.* correction of complex channel impairments, ML allows to reduce the modelling deficit and improve performance when compared to existing solutions. On the other hand, optimal algorithms - *e.g.* minimal Euclidean distance *Maximum Likelihood Decoding* (MLD) - are sometimes known but their algorithmic complexity can grow rapidly with the size of the problem until it eventually leads to computationally infeasible tasks within a realistic time frame. In such situations, ML allows to find lower complexity approximations and thus improve efficiency by reducing the algorithmic deficit.

Multiple contributions have succeeded in improving the decoding performance by applying a NN-based decoder to an existing coding scheme. One such example is the work on NBP, which seeks to implement a trainable weighted BP decoder to improve the decoding performance of short BCH codes [101]. Following this idea, similar approaches are employed in [103, 102, 144, 105]. Other contributions try to learn end-to-end communication schemes, including channel coding. As an example, an AE NN models is used in [115] to jointly design the transmitter and receiver of a simple communication scheme. However, while interesting, these end-to-end *black-box* approaches are, to the best of our knowledge, currently limited to, either small code sizes, *e.g.* (8,4) or (15,7) in [115], or rely on extremely large models, particularly complex to train notably because of the so called "*Curse of dimensionality*". This well-known phenomenon in ML is related to the fact that the solution space of a problem grows exponentially with its dimensionality. This makes the available training data sparse and thus non-statistically significant or the needed volume of training samples too important. Similarly, the "*Curse of (code-word) dimensionality*" appears in the context of PHY layer when one tries to train models with increased information block sizes. The number of code-words associated to information blocks of size k is of 2^k which can rapidly become prohibitively large when increasing k, making the learning of large codes a challenging problem [115, 5].

To alleviate this problem, one solution is to use structured approaches [145, 146], [1, p 576-577]. For example, one can exploit code structures such as convolutional codes [147] or construct specific NN-based decoder structure using RNN [148] or CNN [147] to circumvent this dimensionality problem. Still, existing approaches are, to the best of the our knowledge, limited to the design of convolutional [147, 148], Turbo [149] or Polar codes [107] and often exhibit a high number of model parameters and a reduced interpretability. Also, the comparison with traditional (de)coding schemes is sometimes complex. Indeed, the proposed solutions are often based on the learning of end-to-end communication schemes (usually based on AE), not limited to the (de)coding function but also allow the joint learning of (de)modulation, equalisation, etc. It then becomes difficult to distinguish the contribution of each of these functions to the final performance in view of a fair comparison with classical approaches. A different approach based on BP decoders uses a *Genetic Algorithm* (GA) to optimise LDPC codes from existing distributions with promising results [108].

5.2 An Introduction to Iterative Methods for the Decoding of Linear Block Codes

This brief introduction aims to provide the necessary knowledge on classic iterative techniques that are used for the decoding of LBC such as LDPC codes. The present document introduces as a reminder notions related to LBC before describing soft iterative decoding methods based on TG models and BP algorithms.

5.2.1 Fundamentals of Linear Block Codes

As described in Chapter 3, channel coding is a method used to improve reliability of a transmission over a noisy communication channel by adding redundancy in the transmitted signal so that potential transmission errors can be corrected by the receiver. LBC are a class of FEC for which the information to be transmitted is split into blocks $\boldsymbol{x}$ of fixed size k named *words*. Words are encoded using linear combinations to form *code-words* $\boldsymbol{c}$ of size n with $n > k$. In the case of binary codes, which are the only codes considered in this manuscript, we recall that, there are 2^k different information words and associated code-words, referred to as the *code-book*, forming a subset of $\mathbb{F}_2^n$. The minimal distance d_{min} of the code is the smallest Hamming distance between two code-words.

Let $\mathcal{C}(n,k)$ be a binary LBC of length n and rank k that will be referred to as an "(n,k) code" throughout the rest of this manuscript. Let $\boldsymbol{G} \in \mathbb{F}_2^{k,n}$ be the associated generator matrix of size k by n describing the encoding relations. For example, considering the following encoding relations of a *Hamming* (7,4) codes:

$$\begin{cases} c_1 = x_1 \\ c_2 = x_2 \\ c_3 = x_3 \\ c_4 = x_4 \\ c_5 = x_1 \oplus x_2 \oplus x_4 \\ c_6 = x_1 \oplus x_3 \oplus x_4 \\ c_7 = x_2 \oplus x_3 \oplus x_4 \end{cases} \tag{5.1}$$

These relations can be expressed by the following generator matrix (in standard form):

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{5.2}$$

Let $\boldsymbol{H} \in \mathbb{F}_2^{(n-k),n}$ be the corresponding PC matrix of size $(n-k)$ by n constructed such that $\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0} \in \mathbb{F}_2^{k,(n-k)}$. The role of the PC matrix is to describe the relations to be verified at the receiver for any valid code-words. In the previous Hamming (7,4) code example, a valid PC matrix, obtained using *Gauss-Jordan Elimination* method, could be:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{5.3}$$

Which can equivalently be described by the following system of equation stating that the three parity equations that sums to zero if there is no errors in the code-words:

$$\begin{cases} c_5 \oplus c_1 \oplus c_2 \oplus c_4 = 0 \\ c_6 \oplus c_1 \oplus c_3 \oplus c_4 = 0 \\ c_7 \oplus c_2 \oplus c_3 \oplus c_4 = 0 \end{cases} \tag{5.4}$$

The PC equations can be represented by the diagram of figure 5.1 showing the code covering and its ability to always correct one error and detect up to three errors.

The encoding process to obtain a code-word $\boldsymbol{c} \in \mathbb{F}_2^n$ from a word $\boldsymbol{x} \in \mathbb{F}_2^k$ can simply be defined as:

$$\boldsymbol{c} = \boldsymbol{x}\boldsymbol{G} \in \mathbb{F}_2^n \tag{5.5}$$

For example if we consider the information word $\boldsymbol{x} = \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}$, we can obtain the

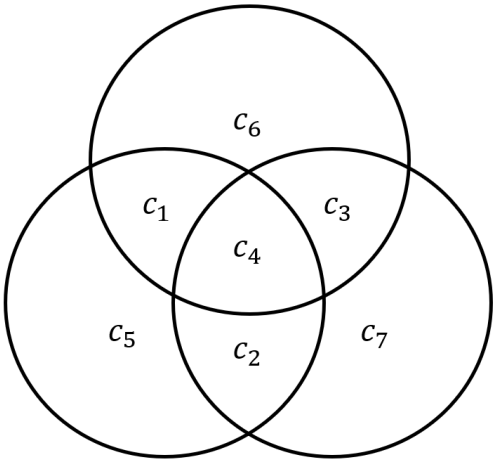


Figure 5.1: Representation of the Three Parity Check Equations of Hamming (7,4) Code
If there is no error in the code-word, the bits inside a given circle must sum to zero.

corresponding code-word as follow[1]:

$$\boldsymbol{c} = \boldsymbol{x}\boldsymbol{G} = \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{5.6}$$

The linear property of linear block codes ensures that any linear combination of code-words is also a code-word. Using linear combinations of the rows and columns permutations, a *standard* form of the generator matrix can be obtained [79]:

$$\boldsymbol{G}_{\text{std}} = \left(\boldsymbol{I}^{k,k} | \boldsymbol{R}^{k,(n-k)} \right) \tag{5.7}$$

where $\boldsymbol{I}^{k,k}$ and $\boldsymbol{R}^{k,(n-k)}$ are respectively an identity matrix and a redundancy matrix.

Using the standard form results in a simplified encoding process with code-words of the form:

$$\boldsymbol{c} = (\boldsymbol{x}|\boldsymbol{r}) \tag{5.8}$$

where $\boldsymbol{x}$ is the raw information word and $\boldsymbol{r}$ the computed redundancy.

When $\boldsymbol{G}$ is in the standard form the code is said to be systematic and the corresponding PC can simply be obtained as:

$$\boldsymbol{H}_{\text{std}} = \left(\left[\boldsymbol{R}^{k,(n-k)} \right]^T \middle| \boldsymbol{I}^{(n-k),(n-k)} \right) \tag{5.9}$$

On the decoder side and by construction of the PC matrix, the received code-word $\boldsymbol{y}$ is valid if the following relation is verified:

$$\boldsymbol{s} = \boldsymbol{y}\boldsymbol{H}^T = \boldsymbol{0} \tag{5.10}$$

where $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$ is called the *syndrome* of the received code-word. If the received code-word contains some errors, the value of the syndrome depends only on the value of the error vector $\boldsymbol{\epsilon}$:

$$\boldsymbol{s} = \boldsymbol{y}\boldsymbol{H}^T = (\boldsymbol{c} + \boldsymbol{\epsilon})\boldsymbol{H}^T = \boldsymbol{\epsilon}\boldsymbol{H}^T$$

where $\boldsymbol{y}$ is the received code-word and $\boldsymbol{\epsilon}$ is the random error vector.

The rate of the code $r = k/n$, represents the proportion of the coded bits that conveys useful information. FEC mechanisms imply an important trade-off between error correction capacity and information rate. In fact, a good way of measuring the global efficiency of a code is to evaluate the BER related to the SNR normalised per information bit usually denoted as E_b/N_0 where E_b denotes the average energy used to transmit one bit of information and N_0 is the variance of the AWGN noise.

[1] All operations described here follow modulo two arithmetic in Galois field $\mathbb{F}_2$

5.2.2 A Step Towards Soft Iterative Decoding

As described in Chapter 3, having more accurate - *e.g.* soft - information about the received data improves the performance of FEC mechanisms, but at the cost of increased decoding complexity (which is even more true as the code size increases). To reduce this complexity, iterative methods can be employed to correct one or a few errors at a time and thus progress until the entire code word is error-free.

Such a concept is illustrated with the example of a three state iterative decoder applied to the decoding of Hamming (7,4) code. Aside from the two states 0 and 1 the receiver can tag received data as *erased* when it is unsure of the received data. As one can see in Figure 5.2, the iterative decoding combined with more precise information on the value of the received symbols (3 different states: 0,1 or "erasure") allow to correct up to three erasures.

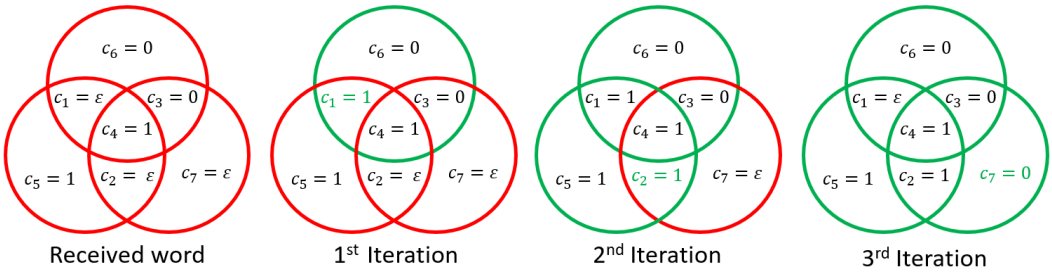


Figure 5.2: Iterative decoding of three erasures with Hamming (7,4) codes.

We can push this idea further using soft-decoding based on LLR instead of hard-decoding with a finite number of states. Instead of using hard-decoding to associate to each bit of information a value from a fixed set of possible values, we can consider soft-decoding where probabilities on the bits values are used instead. In the binary case, one can associate to each received samples y_i the two probabilities $\mathbb{P}\{\mathrm{y}_i = y_i|\mathrm{x}_i = 0\}$ and $\{\mathrm{y}_i = y_i|\mathrm{x}_i = 1\}$ which is often written into a more compact combination of both metric, the *Likelihood Ratio*:

$$\Lambda(y_i) = \frac{\mathbb{P}\{\mathrm{y}_i = y_i|\mathrm{x}_i = 0\}}{\mathbb{P}\{\mathrm{y}_i = y_i|\mathrm{x}_i = 1\}} \tag{5.11}$$

Which can be conveniently transposed into LLR:

$$\lambda(y_i) = \log\frac{\mathbb{P}\{\mathrm{y}_i = y_i|\mathrm{x}_i = 0\}}{\mathbb{P}\{\mathrm{y}_i = y_i|\mathrm{x}_i = 1\}} \tag{5.12}$$

We recall the LLR expression for BPSK symbols under AWGN channel demonstrated in Section 4.2.2:

$$\lambda(y) = \frac{2Ay}{\sigma^2} \tag{5.13}$$

where A is the amplitude of the BPSK symbols and σ^2 the noise variance.

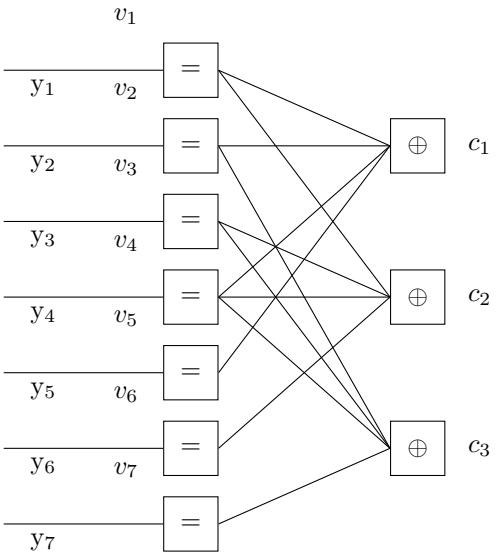
5.2.3 Factor Graph Representation of a Linear Block Code

Now that we have described how a code enforce relations between several variables and how these variables can be described in a probabilistic manner at the receiver side, a question is how to describe an efficient soft decoding process. Such a process would need to combine the information from multiple linked random variables, or said otherwise compute marginals over, potentially large, joint probability distributions. Such a class of problem might efficiently be treated using the theory of FG described in Chapter 2.

In fact, the relation between bits of a code can be represented by a bipartite graph called a *Tanner Graph* (TG). TG are a special case of the more general concept of FG and are used in several iterative decoding algorithms based on MPA such as the BP algorithm. For example, modern codes such as LDPC were designed to ensure good performance and high throughput thanks to low complexity decoding algorithms like BP based on such graphical models. On the

contrary, most of the traditional LBC (Hamming codes, BCH codes, etc.) were rather designed with algebraic properties in mind such as minimal Hamming distance and are usually defined by *High Density Parity-Check* (HDPC) matrices. Such properties do not necessarily guarantee good performance when using TG decoding which rather rely on graph properties such as a low density of connections, stopping sets, graph girth[2], etc [150]. In a way, modern coding techniques give a central place to the decoding algorithm, whereas earlier coding techniques put more emphasis on the MLD performance of the code [150].

Following up on the example of the Hamming (7,4) code, one can define the TG of the code as shown on Figure 5.3. In such a graph, the seven received variables (leaves) are distributed by equality constraint factors (commonly referred to as variable nodes) to each of the three *zero-sum* constraint factors (commonly referred to as control nodes) that bind the variables of the code together.



$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Figure 5.3: Tanner Graph of the Hamming (7,4) Code.

As introduced in Chapter 2, equality constraint factors are defined as:

$$g_{=}\left(x^{(1)}, \ldots, x^{(n)}\right) \triangleq \prod_{i=2}^{n} \delta(x^{(1)} - x^{(i)}) \tag{5.14}$$

where $x^{(1)}, \ldots, x^{(n)}$ are all the variables connected to the factor.

A new type of factor, specific to the FEC use-case, are the zero-sum factors which states that all connected variables should sum (in the corresponding finite field) to zero:

$$g_{\oplus}\left(x_1, \ldots, x_n\right) \triangleq \delta\left(\bigoplus_{i=1}^{n} x_i\right) \tag{5.15}$$

5.2.4 Applying Message Passing Algorithm to the Factor Graph of a Code

Now that the structure and nodes of the FG of a code have been described we will detail how one can apply the standard sum-product rule to such a structure to perform inference. We recall

[2]The girth of a code denotes the length of the shortest cycle in its bipartite Tanner graph.

the (discrete) sum-product rule described in Chapter 2:

$$\boxed{\mu_{g\to \mathrm{x}_i}(x_i) = \sum_{\backsim x_i} g(x_1, \ldots, x_i, \ldots, x_k) \prod_{\substack{j=1 \\ j\neq i}}^{k} \mu_{\mathrm{x_j}\to g}(x_j)} \tag{5.16}$$

And the final marginalisation rule when the two messages of an edge are available:

$$\boxed{f(x_i) = \mu_{g_1\to \mathrm{x}_i}(x_i) \times \mu_{g_2\to \mathrm{x}_i}(x_i)} \tag{5.17}$$

We now define the corresponding sum-product rule for the two types of nodes encountered in a Tanner graph:

Sum-Product Rule at a Variable Node - Equality Constraint

In the case of equality node, the factor function is defined following Eq. (5.14). The Sum-Product rule defining the outgoing message is thus reduced to:

$$\mu_{g\to \mathrm{x}_i}(x_i) = \underbrace{\sum_{\backsim x_i} \overbrace{\begin{cases} 0 & \text{if any} \quad x_j \neq x_i \\ 1 & \text{if} \quad x_2 = \ldots = x_i = \ldots = x_n \end{cases}}^{\delta(x_i - x_2)...\delta(x_i - x_n)}}_{=1} \prod_{j\neq i} \mu_{\mathrm{x_j}\to g}(x_j) = \prod_{j\neq i} \mu_{\mathrm{x_j}\to g}(\mathrm{x}_j = x_i) \tag{5.18}$$

Indeed the nested sum over all variable domains except variable x_i is valid iff all variables connected to the factor are equal to x_i.

Recall that we consider binary variables. As such, the marginalisation process over the factor graph consists either in computing the PMF of the variables, *i.e.* $p_{x_i}(\mathrm{x_i} = 0)$ and $p_{x_i}(\mathrm{x_i} = 1)$, or, in a more compact form its LLR, *i.e.* $\log \frac{p_{\mathrm{x}_i}(\mathrm{x_i}=0)}{p_{\mathrm{x}_i}(\mathrm{x_i}=1)}$. The exchanged messages in the FG are thus either of the form of vectors of size 2 containing the two probabilities, or more simply scalar LLR. If we consider messages based on probabilities, the Eq (5.18) becomes[3]:

$$\boxed{\mu_{g\to \mathrm{x}_i}(x_i) = \begin{pmatrix} p_{g\to \mathrm{x}_i}(\mathrm{x}_i = 0) \\ p_{g\to \mathrm{x}_i}(\mathrm{x}_i = 1) \end{pmatrix} = \alpha \prod_{j\neq i} \begin{pmatrix} p_{\mathrm{x_j}\to g}(\mathrm{x}_j = 0) \\ p_{\mathrm{x_j}\to g}(\mathrm{x}_j = 1) \end{pmatrix}} \tag{5.19}$$

If LLR are considered instead of probability the outgoing message from equality constraint node g becomes:

$$\boxed{\mu_{g\to \mathrm{x}_i}(x_i) = \lambda_{g\to \mathrm{x}_i}(x_i) = \log\left(\frac{p_{g\to \mathrm{x}_i}(\mathrm{x}_i = 0)}{p_{g\to \mathrm{x}_i}(\mathrm{x}_i = 1)}\right) = \log\left(\prod_{j\neq i} \frac{p_{\mathrm{x_j}\to g}(\mathrm{x}_i = 0)}{p_{\mathrm{x_j}\to g}(\mathrm{x}_i = 1)}\right) = \sum_{j\neq i} \lambda_{\mathrm{x_j}\to g}(x_j)} \tag{5.20}$$

Note that we adopt, within the FG formalism, the following notation for PMF:

$$p_{g\to \mathrm{x}_i}(x_i)$$

This slightly different notation uses an arrow to express the origin/destination of some probablistic information in the graph. For example, $p_{g\to \mathrm{x}_i}(x_i = 0) = \alpha \prod_{j\neq i} p_{\mathrm{x_j}\to g}(x_j = 0)$ denote the probability that the variable $\mathrm{x_i}$ equals 0, given the extrinsic information summarised by node g.

[3] An optional scaling factor α is added so that the probabilities always sum to one.

Sum-Product Rule at a Check Node - Zero-Sum Constraint

In the case of zero-sum node the factor function is defined by Eq. (5.15). The Sum-Product rule is therefore given by:

$$\mu_{g\to \mathrm{x}_i}(x_i) = \sum_{\sim x_i} \delta(x_1 \oplus ... \oplus x_i \oplus ... \oplus x_n) \prod_{j\neq i} \mu_{\mathrm{x}_j \to g}(x_j) \tag{5.21}$$

Let x_i be random binary variables and the messages $\mu_{x_j\to g}(x_j)$ of the form of binary PMF (*i.e.* size two vectors). Let y be a random binary variable defined as the (modulo-2) sum of the n variable x_i:

$$\mathrm{y} = \bigoplus_{i=1}^{n} \mathrm{x}_i \tag{5.22}$$

One can show that[4]:

$$\begin{cases} p_{\mathrm{y}}(\mathrm{y}=0) - p_{\mathrm{y}}(\mathrm{y}=1) = \prod_{i=1}^{n} \left[p_{\mathrm{x}_i}(\mathrm{x}_i=0) - p_{\mathrm{x}_i}(\mathrm{x}_i=1)\right] \\ p_{\mathrm{y}}(\mathrm{y}=0) + p_{\mathrm{y}}(\mathrm{y}=1) = \prod_{i=1}^{n} \left[p_{\mathrm{x}_i}(\mathrm{x}_i=0) + p_{\mathrm{x}_i}(\mathrm{x}_i=1)\right] = 1 \end{cases} \tag{5.23}$$

Which can be simplified as:

$$\begin{cases} p_{\mathrm{y}}(\mathrm{y}=0) = \frac{1}{2} + \frac{1}{2} \prod_{i=1}^{n} \left[p_{\mathrm{x}_i}(\mathrm{x}_i=0) - p_{\mathrm{x}_i}(\mathrm{x}_i=1)\right] \\ p_{\mathrm{y}}(\mathrm{y}=1) = \frac{1}{2} - \frac{1}{2} \prod_{i=1}^{n} \left[p_{\mathrm{x}_i}(\mathrm{x}_i=0) - p_{\mathrm{x}_i}(\mathrm{x}_i=1)\right] \end{cases} \tag{5.24}$$

Equation (5.21) can thus be rewritten as:

$$\mu_{g\to \mathrm{x}_i}(x_i) = \begin{pmatrix} p_{g\to \mathrm{x}_i}(0) \\ p_{g\to \mathrm{x}_i}(1) \end{pmatrix} = \alpha \begin{pmatrix} \overbrace{\sum_{\sim x_i} \delta(x_1 \oplus ... \oplus x_i = 0 \oplus ... \oplus x_n) \prod_{j\neq i} \mu_{\mathrm{x}_j\to g}(x_j)}^{p_{\mathrm{y}}(\mathrm{y}=0) \quad \text{with} \quad \mathrm{y}=\bigoplus_{i\neq j} \mathrm{x}_i} \\ \underbrace{\sum_{\sim x_i} \delta(x_1 \oplus ... \oplus x_i = 1 \oplus ... \oplus x_n) \prod_{j\neq i} \mu_{\mathrm{x}_j\to g}(x_j)}_{p_{\mathrm{y}}(\mathrm{y}=1) \quad \text{with} \quad \mathrm{y}=\bigoplus_{i\neq j} \mathrm{x}_i} \end{pmatrix} \tag{5.25}$$

$$\boxed{\mu_{g\to \mathrm{x}_i}(x_i) = \alpha \begin{pmatrix} \frac{1}{2} + \frac{1}{2} \prod_{j\neq i} \left[p_{\mathrm{x}_j\to g}(0) - p_{\mathrm{x}_j\to g}(1)\right] \\ \frac{1}{2} - \frac{1}{2} \prod_{j\neq i} \left[p_{\mathrm{x}_j\to g}(0) - p_{\mathrm{x}_j\to g}(1)\right] \end{pmatrix}} \tag{5.26}$$

Which is the expression of the sum-product rule at a zero-sum factor node, when considering random binary variables and messages taking the form of probabilities. Lets now see how this same rule can be expressed in terms of LLR:

Let

$$\Delta_{g\to x_i}(x_i) \triangleq \frac{p_{g\to \mathrm{x}_i}(0) - p_{g\to \mathrm{x}_i}(1)}{p_{g\to \mathrm{x}_i}(0) + p_{g\to \mathrm{x}_i}(1)} \tag{5.27}$$

From Eq. (5.23) one can write:

$$\Delta_{g\to \mathrm{x}_i}(x_i) = \frac{\prod_{j\neq i} \left[p_{\mathrm{x}_j\to g}(0) - p_{\mathrm{x}_j\to g}(1)\right]}{\prod_{j\neq i} \left[p_{\mathrm{x}_j\to g}(0) + p_{\mathrm{x}_j\to g}(1)\right]} = \prod_{j\neq i} \Delta_{\mathrm{x}_j\to g}(x_j) \tag{5.28}$$

[4]Proof in Appendices A.I

Let $\Delta_{\mathrm{x}_j \to g}(x_j)$ be expressed in terms of likelihood ratios:

$$\Delta_{\mathrm{x}_j \to g}(x_j) = \frac{\frac{p_{\mathrm{x}_j \to g}(0)}{p_{\mathrm{x}_j \to g}(1)} - 1}{\frac{p_{\mathrm{x}_j \to g}(0)}{p_{\mathrm{x}_j \to g}(1)} + 1} = \frac{\Lambda_{\mathrm{x}_j \to g}(x_j) - 1}{\Lambda_{\mathrm{x}_j \to g}(x_j) + 1} \tag{5.29}$$

By definition of the LLR, we have:

$$\lambda_{\mathrm{x}_j \to g}(x_j) \triangleq \log\left(\frac{p_{\mathrm{x}_j \to g}(0)}{p_{\mathrm{x}_j \to g}(1)}\right) = \log\left(\Lambda_{\mathrm{x}_j \to g}(x_j)\right) \tag{5.30}$$

As a consequence:

$$\Delta_{\mathrm{x}_j \to g}(x_j) = \frac{e^{\lambda_{\mathrm{x}_j \to g}(x_j)} - 1}{e^{\lambda_{\mathrm{x}_j \to g}(x_j)} + 1} = \frac{e^{\lambda_{\mathrm{x}_j \to g}(x_j)/2} - e^{-\lambda_{\mathrm{x}_j \to g}(x_j)/2}}{e^{\lambda_{\mathrm{x}_j \to g}(x_j)/2} + e^{-\lambda_{\mathrm{x}_j \to g}(x_j)/2}} = \tanh\left(\frac{\lambda_{\mathrm{x}_j \to g}(x_j)}{2}\right) \tag{5.31}$$

Combining Eq. (5.28) and (5.31) leads to:

$$\tanh\left(\frac{\lambda_{g \to \mathrm{x}_i}(x_i)}{2}\right) = \prod_{j \neq i} \tanh\left(\frac{\lambda_{\mathrm{x}_j \to g}(x_j)}{2}\right) \tag{5.32}$$

As a consequence the outgoing message from a zero-sum factor node g toward the edge x_i, knowing the LLR messages from incoming edges x_j is given by:

$$\boxed{\lambda_{g \to \mathrm{x}_i}(x_i) = 2\tanh^{-1}\left(\prod_{j \neq i} \tanh\left(\frac{\lambda_{\mathrm{x}_j \to g}(x_j)}{2}\right)\right)} \tag{5.33}$$

Table 5.1 summarise the two conventional sum-product rules in a TG for BP decoding of binary codes (with their probabilities and LLR forms).

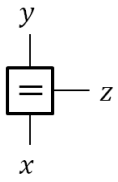
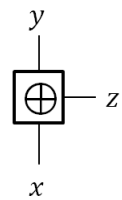
	In terms of probabilities: $\mu_{g \to \mathrm{z}}(z) = \begin{pmatrix} p_{g \to \mathrm{z}}(0) \\ p_{g \to \mathrm{z}}(1) \end{pmatrix} = \alpha \begin{pmatrix} p_{\mathrm{x} \to g}(0) \times p_{\mathrm{y} \to g}(0) \\ p_{\mathrm{x} \to g}(1) \times p_{\mathrm{y} \to g}(1) \end{pmatrix}$ In terms of LLR: $\mu_{g \to \mathrm{z}}(z) = \lambda_{g \to \mathrm{z}}(z) = \lambda_{\mathrm{x} \to g}(x) + \lambda_{\mathrm{y} \to g}(y)$
	In terms of probabilities: $\mu_{g \to \mathrm{z}}(z) = \begin{pmatrix} p_{g \to \mathrm{z}}(0) \\ p_{g \to \mathrm{z}}(1) \end{pmatrix} = \alpha \begin{pmatrix} p_{\mathrm{x} \to g}(0)p_{\mathrm{y} \to g}(0) + p_{\mathrm{x} \to g}(1)p_{\mathrm{y} \to g}(1) \\ p_{\mathrm{x} \to g}(0)p_{\mathrm{y} \to g}(1) + p_{\mathrm{x} \to g}(1)p_{\mathrm{y} \to g}(0) \end{pmatrix}$ In terms of LLR: $\mu_{g \to \mathrm{z}}(z) = \lambda_{g \to \mathrm{z}}(z) = 2\tanh^{-1}\left[\tanh\left(\frac{\lambda_{\mathrm{x} \to g}(x)}{2}\right) \times \tanh\left(\frac{\lambda_{\mathrm{y} \to g}(y)}{2}\right)\right]$

Table 5.1: Sum-Product Rules for Equality and Zero-Sum Factors of a Tanner Graph

Belief Propagation Decoding of a Linear Block Codes

We described in previous sections the Sum-Product rule for different types of nodes involved in the TG representation of a code. In this section, we describe the iterative BP[5] decoding algorithm which uses the sum-product rule to compute efficiently all the marginals of the TG of a code. We recall the basic steps of such a message passing algorithm on a generic graph:

- Input variables are provided to the half-edge (leaves) of the graph.
- For each nodes where all the needed input messages are available, new messages are computed according to the sum-product rule (Eq. (5.16)).
- As new messages are computed and propagated, one can progress in the graph and successively compute new messages. In cycle-free graphs, only one pass forward and one pass backward is needed and each messages are computed only once.
- Finally, for each edge, the marginal function can be computed as the product of the two opposite messages (Eq. (5.17)).

In the context of communication systems, messages exchanged at the decoder between the nodes of the TG are usually related to the LLR of transmitted bits. The sum-product update rule can be applied iteratively to all the nodes of a code graph using equations (5.34), (5.35) and (5.36) (see previous sections for the detailed derivations).

Message from the variable node i toward the check node j is computed using:

$$\mu_{v_i \to c_j} = \lambda_i + \sum_{l \neq j} \mu_{c_l \to v_i} \tag{5.34}$$

where λ_i is the *a priori* LLR received by variable node i and $\mu_{c_l \to v_i}$ are the other messages received by variable node i from neighbouring check nodes l.

Similarly, message from the check node j toward the variable node i is defined as:

$$\mu_{c_j \to v_i} = 2 \tanh^{-1} \left[\prod_{l \neq i} \tanh \left(\frac{\mu_{v_l \to c_j}}{2} \right) \right] \tag{5.35}$$

where $\mu_{v_l \to c_j}$ are the messages received by check node j from neighbouring variable nodes l.

Finally, the *a posteriori* LLR can be computed using the marginalisation rule at the variable node i as:

$$\tilde{\lambda}_i = \lambda_i + \sum_{l} \mu_{c_l \to v_i} \tag{5.36}$$

Note that the marginalisation step uses here a sum instead of a product because of the LLR nature of the exchanged message (more specifically because of the use of a log function)

As explained in Chapter 2, while BP is generally exact for tree structures (meaning that it always converges to the true posterior in one pass) it is only approximate for graphs with cycles. The presence of such loops in the inference graph can induce undesirable feedback effects. The TG of codes are generally not cycle-free and the BP inference not exact, hence the necessity to apply iterative decoding by passing messages back and forth between variable and check nodes, using equations (5.34) and (5.35), before hopefully converging to a satisfying solution. In practice, the BP has proven to be effective in the decoding of high girth (the shortest cycle of a TG) codes, such as large LDPC, offering both high performance and reasonable complexity. For shorter and higher density codes, the presence of short cycles is usually detrimental to the decoding performance [151].

[5] "Belief Propagation" and "Sum-Product" decoding are often used interchangeably to refer to the same decoding process based on the sum-product algorithm. In this document, we will mostly use "Belief Propagation".

5.2.5 Degree Distribution Notations

Following notations from *Richardson and Urbanke* [150], a convenient way to describe LBC *ensembles* and particularly LDPC ensembles is to use degree distribution pairs (Λ, P). Let $\mathcal{C}(n,k)$ be a LBC of size (n,k). Let Λ_i be the number of variable nodes with degree i and P_j be the number of check nodes with degree j. It is worth noting that the total number of variable nodes is $n = \sum_i \Lambda_i$, the total number of check nodes is $(n-k) = \sum_j P_j$ and the total number of ones in the matrix, *i.e.* edges of the TG, is $e = \sum_i i\Lambda_i = \sum_j jP_j$. From these notations, one can define the variable and check *degree distribution from node perspective* polynomials:

$$\Lambda(x) = \sum_i \Lambda_i x^i \qquad P(x) = \sum_j P_j x^j \tag{5.37}$$

As previously, the total number of variable nodes and check nodes can be noted as:

$$\Lambda(1) = \sum_i \Lambda_i = n \qquad P(1) = \sum_j P_j = (n-k) \tag{5.38}$$

And the *normalised degree distributions from node perspective* are written as:

$$L(x) = \frac{\Lambda(x)}{\Lambda(1)} = \sum_i \frac{\Lambda_i x^i}{\Lambda(1)} = \sum_i L_i x^i \qquad R(x) = \frac{P(x)}{P(1)} = \sum_j \frac{P_j x^j}{P(1)} = \sum_j R_j x^j \tag{5.39}$$

In other words, L_i (R_j respectively) is the probability that a variable node (check node respectively) chosen uniformly at random among the n nodes ($(n-k)$ nodes respectively) has a degree i (j respectively).

The overall code matrix density can be derived as:

$$\delta = \frac{e}{n(n-k)} = \frac{\sum_i i\Lambda_i}{\Lambda(1)P(1)} = \frac{\sum_j jP_j}{\Lambda(1)P(1)} \tag{5.40}$$

5.3 Belief Propagation Decoder as an Efficient Recurrent Neural Network for Short Codes

In this section, we describe how the BP propagation algorithm can be expressed using a RNN model. The proposed model allows to efficiently execute a trainable and weighted version of the conventional BP algorithm that can improve the decoding performance of short codes.

5.3.1 Description of the Neural Belief Propagation Algorithm and its Variants

As described in previous section, the FG of a code not being usually cycle-free, it is necessary to apply the BP iterative decoding by passing messages back and forth between variable and check nodes, using equations (5.34) and (5.35), before hopefully converging to a satisfying solution. While the performance of such decoder is close to optimal when considering high girth codes such as large LDPC, the presence of short cycles in smaller codes can induce feedback effects, detrimental to the decoding performance. NBP decoders were recently introduced by *Nachmani et al.* as a way to improve the decoding performance of BP iterative algorithm for short to medium length LBC [101].

Neural Belief Propagation Decoder

The main idea behind NBP algorithm [101] is to represent the BP algorithm using a NN and to learn how to weight BP equations to reduce the negative influence of the short cycles on the decoding performance. In the form described in [101], the NBP modifies equations (5.34) and (5.36) by adding four sets of trainable weights ($\boldsymbol{\omega}^{(\lambda)}, \boldsymbol{\omega}^{(\mu)}, \boldsymbol{\omega}^{(\tilde{\lambda})}$ and $\boldsymbol{\omega}^{(\tilde{\mu})}$):

$$\mu_{v_i \to c_j} = \omega_{i,j}^{(\lambda)} \lambda_i + \sum_{l \neq j} \omega_{i,j,l}^{(\mu)} \mu_{c_l \to v_i} \tag{5.41}$$

$$\widetilde{\lambda_i} = \omega_i^{(\tilde{\lambda})} \lambda_i + \sum_{l} \omega_{i,l}^{(\tilde{\mu})} \mu_{c_l \to v_i} \tag{5.42}$$

The interested reader can refer to the original publication for the detailed derivations of NBP equations [101]. With the growing number of IoT use-cases and associated needs of low-complexity decoding and performing short codes, this type of decoder is of interest and several publications discussed improvements of NBP using pruning [102], weights sharing [144] or active sampling [105].

Parallel Neural Belief Propagation Decoders

Parallel variants of NBP have also been proposed in the original paper (mRRD-RNN) [101] and further improved in a second publication (perm-RNN) [104] thus combining the concepts of *modified Random Redundant Decoder* (mRRD) and NBP decoders. The latter drastically improves the performance of the decoder and bring it closer to that of MLD, at the cost of much higher complexity. Indeed, depending on the received error vector different valid decoding graphs can lead to different decoding performance. In these decoders, each of the parallel decoding branches is a BP decoder (or variant) to which a different permutation of the received code-word is supplied. To ensure the correct operation of this approach, the permutations are carefully chosen and must belong to the automorphism group of the code. The main drawbacks associated to these approaches are the additional complexity and the mandatory knowledge of the *automorphism* group of the code. One interesting variant of this approach [106] proposes to use a NN to predict which of the permutation will probably lead to the best decoding results and therefore reduce the decoding complexity by executing only the corresponding branch of the parallel decoder. Such parallel decoders will not be further detailled in the present work.

5.3.2 Neural Belief Propagation as an Efficient Gated Recurrent Neural Network

In this section, a generic NN architecture is proposed to execute an efficient version of NBP algorithm applied to the decoding of any short to medium sized LBC. Unlike the original NBP algorithm, which is configured to decode predefined codes from the literature, we expect the proposed architecture to be generic and thus able to learn to decode without any prior knowledge. At first, sum-product rules will be described as efficient matrix operations for generic LBC. Then the proposed operations will be embedded in a custom RNN cell architecture. Mechanisms, such as weight sharing and gating, enabling an efficient decoding scheme to be learned will be described. A generic factor graph, described in Fig. 5.4, will be used throughout this section to illustrate the proposed architecture[6]

In the proposed approach, and similarly to the NBP, edges weights are used to improve the decoding performance in the presence of short cycles. Still, to reduce the complexity, the NBP equations were simplified to:

$$\mu_{v_i \to c_j} = \lambda_i + \sum_{l \neq j} \omega_{i,l}^{(\mu)} \mu_{c_l \to v_i} \tag{5.43}$$

$$\widetilde{\lambda_i} = \lambda_i + \sum_{l} \omega_{i,l}^{(\tilde{\mu})} \mu_{c_l \to v_i} \tag{5.44}$$

This form is more compact than standard NBP as there are no weights for the inputs LLR and the message weights of equation (5.43) do not depend on the destination check nodes anymore.

[6]For the sake of simplicity, such a FG cannot represent a valid code $\mathcal{C}(n,k)$ since the number of check nodes $n_{\text{check}} = n - k$ is equal to the number of variable nodes $n_{\text{var}} = n$, so that k is necessarily equal to 0. Nevertheless, the proposed structure can easily be extended to any other number of variable nodes and check nodes and thus represent valid codes.

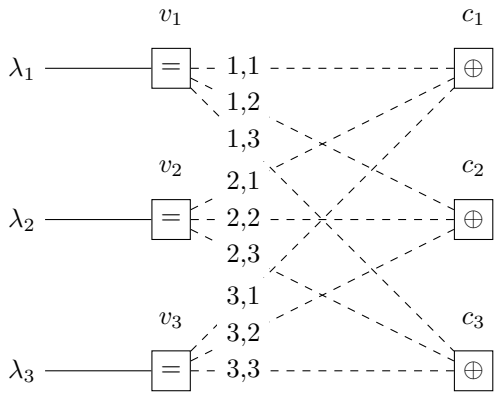


Figure 5.4: Generic 3×3 Factor Graph
λ_i are the received LLR, v_i variable nodes and c_j check nodes. The dashed edges represent the generic nature of the factor graph, where, depending on the considered decoding scheme, the edges may or may not be actually present. Note that the edges, and the corresponding weights that will be used in the following sections, are always indexed from the variable node i to the control node j they connect.

These modifications reduce the complexity, potentially at the cost of a slight reduction in model expressiveness, although no significant performance degradation was observed with respect to NBP.

As in the NBP, the message equation from the control nodes to the variable nodes remains unchanged from the conventional BP algorithm:

$$\mu_{c_j \to v_i} = 2 \tanh^{-1} \left[\prod_{l \neq i} \tanh \left(\frac{\mu_{v_l \to c_j}}{2} \right) \right] \tag{5.45}$$

Variable to Check Nodes Computation

At each NBP iteration the decoder starts by updating messages from variable to check nodes, $\mu_{v_i \to c_j}$, following the weighted Eq. (5.43). This computation can be implemented using a simple matrix multiplication as shown in Figure 5.5. A diagonal matrix is formed from the concatenation of the messages received from check nodes[7] and the input LLR. The resulting matrix is then multiplied by a weight matrix. With a good initialization of the parameters, this layer can rigorously perform the update rule for messages from variable to check nodes, but it has several drawbacks:

- High complexity in terms of number of parameters and computations.
- Need of a sparse weights matrix to actually perform BP algorithm.
- Loss of explanability after a training. There are no guarantees that the performed algorithm is still a BP or NBP (without further refinements, all parameters of the weight matrix, even zeros, being trainable).
- Low generalization capabilities due to loosely structured architecture [145, 152] leading to probable unscalability to bigger codes.

An efficient, scalable and fully differentiable computational graph for the variable to check nodes update rule is proposed in Figure 5.6. This graph drastically reduces the number of parameters and operations as shown in Table 5.2 and ensures the efficient execution of a lower complexity NBP. In a sense, the proposed structured computation can relate to the philosophy behind the efficient implementation of convolutional layers based on general matrix multiplication [153]. In these approaches, the inputs and kernels of the convolutional layer are first reshaped in a

[7]The input messages received from check nodes $\mu_{c_j \to v_i}$ are initialised to 0 at the first iteration, which correspond to an equally likely LLR prior.

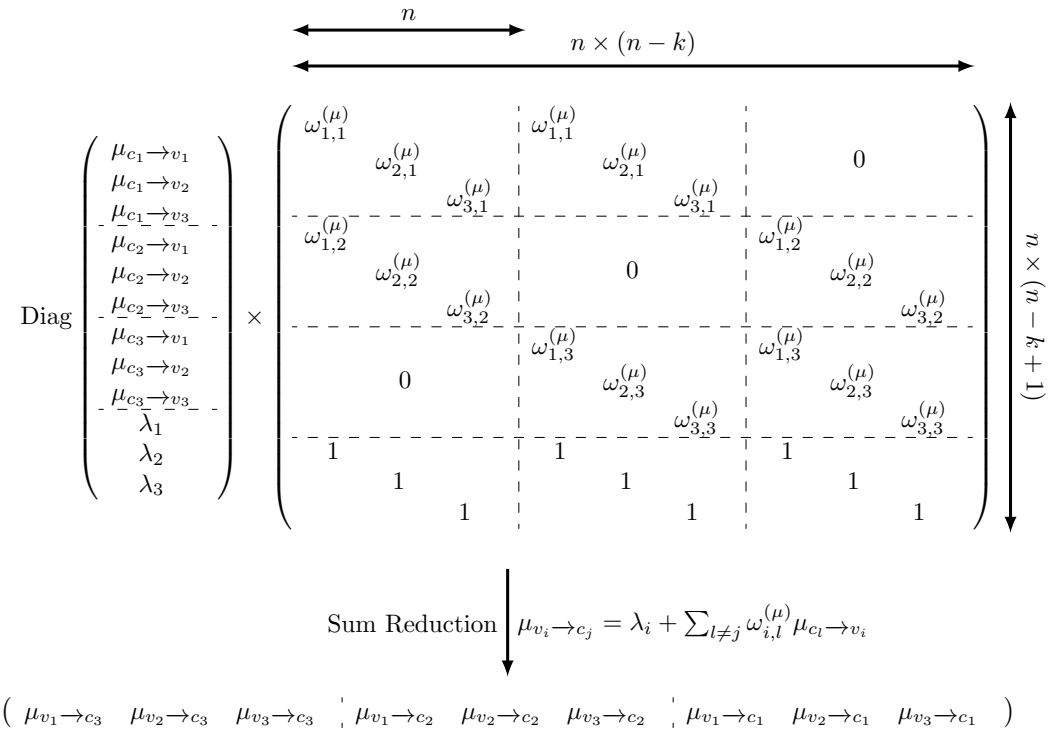


Figure 5.5: Naive Implementation of Eq. (5.43) using a Matrix Multiplication
$Diag(\boldsymbol{v})$ denotes the diagonal matrix whose diagonal elements are constructed from the elements of the vector $\boldsymbol{v}$. n is the number of variable nodes and $(n-k)$ is the number of check nodes.

particular way, before performing a convolution calculation using compact matrix multiplication instead of the sparse multiplication (see section 2.2.2) that would result from a naive approach. Once again, we emphasise on the generic nature of the proposed description. Obviously, a realistic (de)coding scheme would not involve all of the possible connections of the FG, such that certain of the $\omega_{i,j}^{(\mu)}$ would be equal to zero (in the case of a LDPC code, the connection topology would even be a sparse topology such that most $\omega_{i,j}^{(\mu)}$ would in fact be equal to zero). In the present work, we distinguish the contribution of the graph structure - *i.e.* what connections should be present in the decoding graph - from that of the NBP weights - *i.e.* how these connections should be weighted so as to improve the decoding performance. We thus note $\boldsymbol{\omega}^{(\mu)} = \sigma(\boldsymbol{w}_G) \odot \boldsymbol{w}_{\boldsymbol{\Sigma}}$ where $\sigma(\boldsymbol{w}_G)$ is a gating mechanism which represent the decoder structure and is used to prune/select the relevant connections[8], $\odot$ denote the point-wise multiplication, or *Hadamard* product, while $\boldsymbol{w}_{\boldsymbol{\Sigma}}$ is a weighting mechanism used to improve the decoding performance similarly to what is used in the original NBP algorithm from *Nachmani et al.*. Both sets of parameters are assumed to be trainable if one seeks to learn to decode without prior knowledge of a coding scheme. Yet, the gating weights can also be set manually from the parity check matrix of code - this architecture then become similar to the original NBP approach - and the NBP weights be set to 1 - the algorithm then represent a conventional BP.

	Naive (Fig 5.5)	Improved (Fig 5.6)
# multiplications	$\mathcal{O}(n^2 \times (n-k)^2)$	$\mathcal{O}(n \times (n-k))$
# additions	$\mathcal{O}(n^2 \times (n-k)^2)$	$\mathcal{O}(n \times (n-k)^2)$
Biggest tensor allocation	$\mathcal{O}(n^2 \times (n-k)^2)$	$\mathcal{O}(n \times (n-k)^2)$

Table 5.2: Complexity - Variable Nodes to Check Nodes

[8]The $\sigma(x)$ function refers, as is often the case in ML, to a function with sigmoidal behaviour used to enforce the binary nature of the gating mechanism. Different functions will be used in the course of this work, so that they will be formally defined at a later stage.

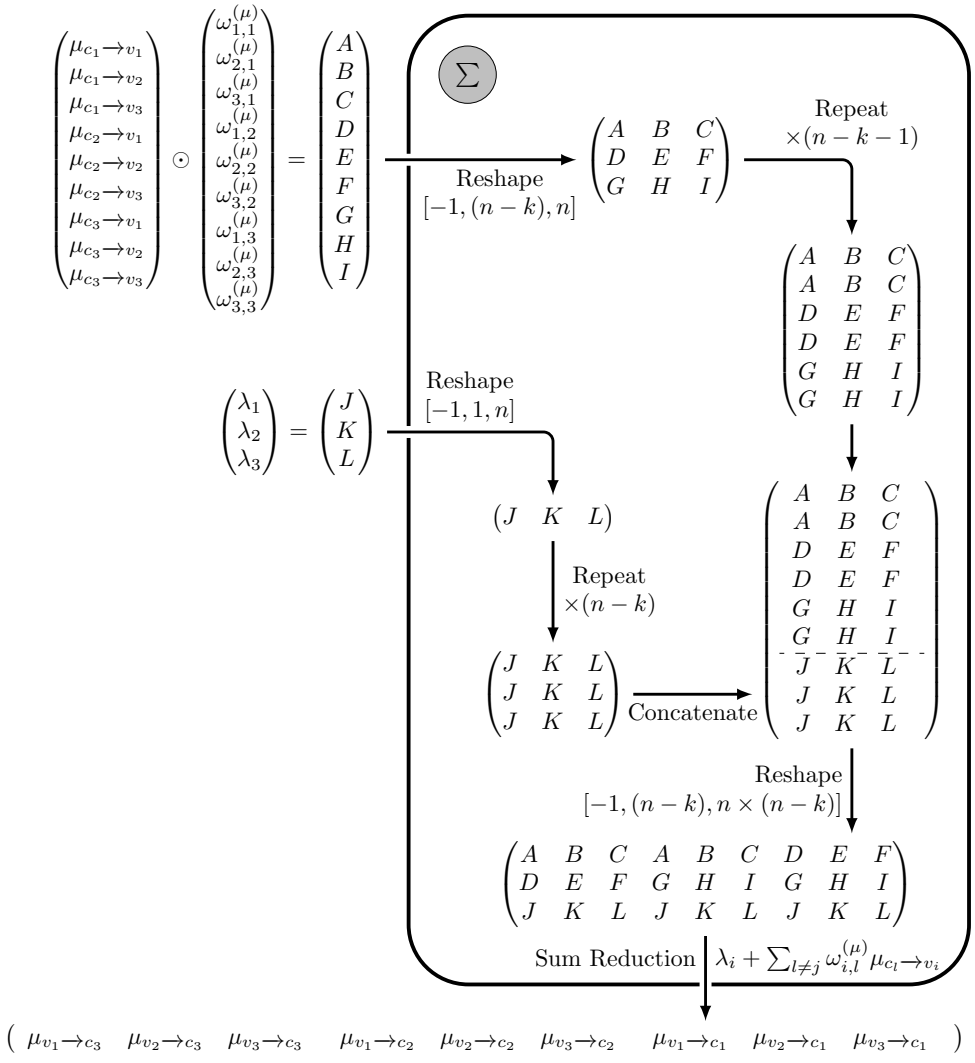


Figure 5.6: Efficient Computational Graph for Eq. (5.43)
Using several reshaping operations, the sparse computation of the naive implementation described in Figure 5.5 is transformed into a compact and structured calculation that precisely represents a weighted BP algorithm. The -1 terms in the various reshaping operations denote the fact that these operation are broadcasted over an arbitrary batch-size. $\odot$ denotes the point-wise product, or *Hadamard* product.

Check to Variable Nodes Computation

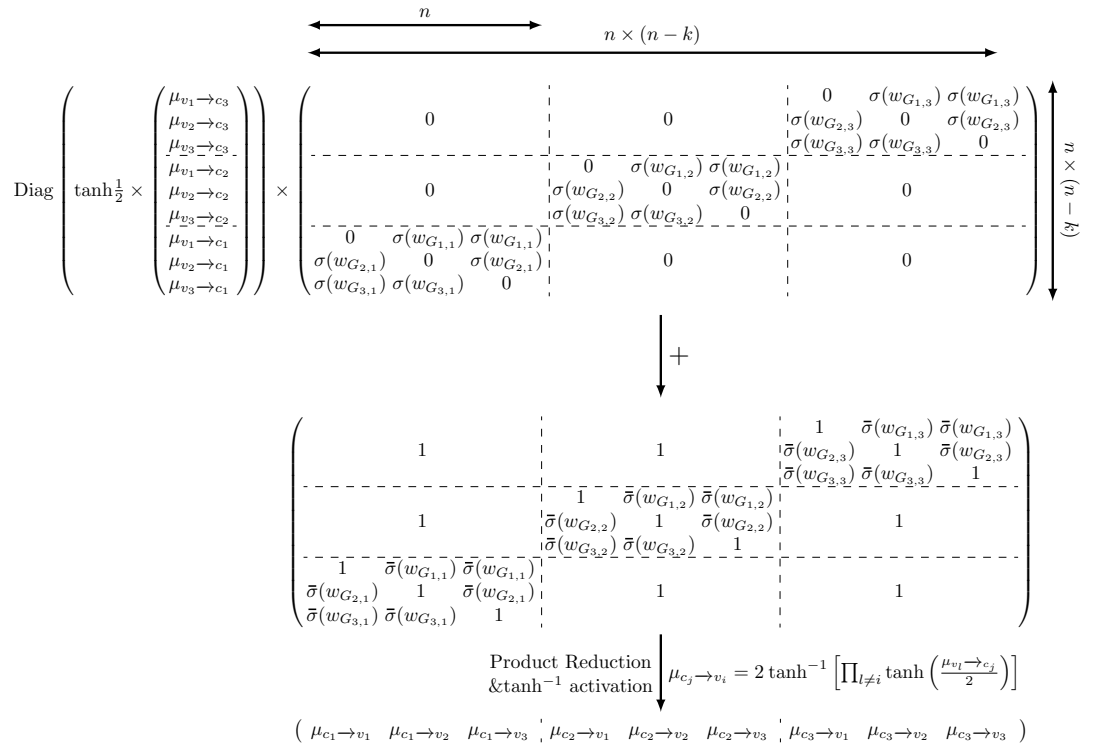


Figure 5.7: Naive Implementation of Eq. (5.45)
As previously described, $\sigma(\boldsymbol{w_G})$ denote the (ideally binary) gating weights. They are used to select the relevant messages, depending on the decoding graph structure. $\bar{\sigma}(\boldsymbol{w_G})$ = 1 - $\sigma(\boldsymbol{w_G})$ denotes the (soft) binary inverse of $\sigma(\boldsymbol{w_G})$.

The second step of the NBP decoding process consists in updating messages from check to variable nodes $\mu_{c_j \to v_i}$ based on Eq. (5.45). It can be implemented by using a naive architecture based on a few differentiable operations as described in Fig. 5.7. The multiplicative step selects the relevant inputs based on the shared gating mechanism $\sigma(\boldsymbol{w}_G)$. The additive step ensures the neutral element of the product is added to non-selected inputs, before the upcoming product reduction (so that the latter does not always give a result equal to 0). The following $\tanh^{-1}$ activation function having exploding gradient when approaching -1 or $+1$ makes gradient based training through such a functional block difficult. To overcome this issue one can use a Taylor expansion of the function as proposed in [154], or simply clipping. A judicious parameterization of this architecture implements the update rule for messages from check to variable nodes. But, once again, it is inefficient and can lose interpretability after a training phase. An efficient computational graph for this update rule is proposed in Fig. 5.8. Based on standard matrix operations, it reduces the complexity as shown in Table 5.3[9].

[9]The algorithmic complexity of tanh and $\tanh^{-1}$ operators is not included in the table because of the presence of these operations in both architectures.

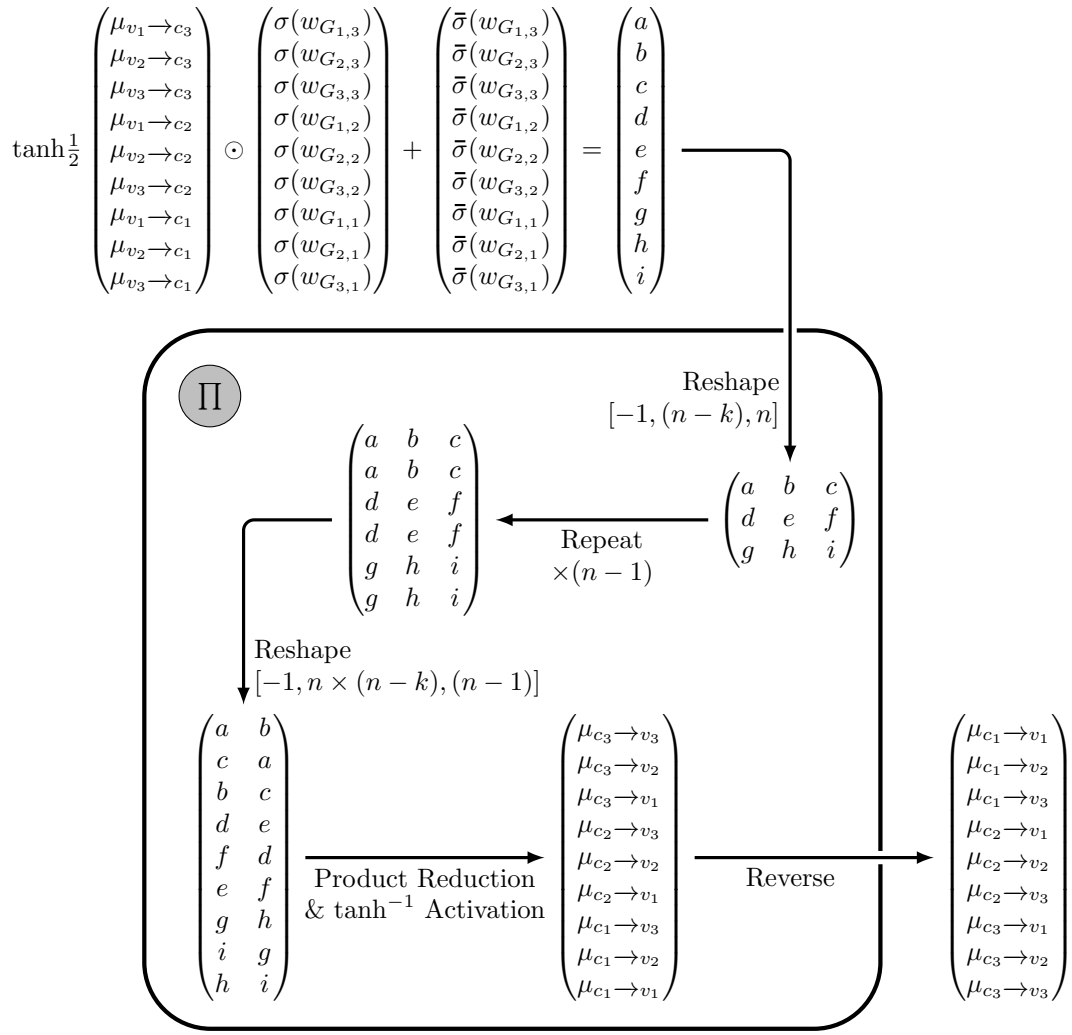


Figure 5.8: Efficient Computational Graph for Eq. (5.45)
It is important to note that given the order of the input messages provided by previous operation (see Section 5.3.2), it is necessary to re-arrange the parameters vectors too.

	Naive (Fig 5.7)	Improved (Fig 5.8)
# multiplications	$\mathcal{O}(n^2 \times (n-k)^2)$	$\mathcal{O}(n^2 \times (n-k))$
# additions	$\mathcal{O}(n^2 \times (n-k)^2)$	$\mathcal{O}(n \times (n-k))$
Biggest tensor allocation	$\mathcal{O}(n^2 \times (n-k)^2)$	$\mathcal{O}(n^2 \times (n-k))$

Table 5.3: Complexity - Check Nodes to Variable Nodes

Output Marginalisation

The final step of the NBP decoding consists in computing the *a posteriori* LLR $\tilde{\lambda}_i$ using Eq. (5.44). Similarly to Section 5.3.2, one can express this computation as a sparse matrix multiplication as shown in Fig. 5.9. A more efficient computational graph is proposed on Fig. 5.10. The complexity of the proposed model is reduced compared to the naive architecture as described in Table 5.4. Again, and similarly to the case of the variable to check node computation, some of the $\omega_{i,j}^{(\tilde{\mu})}$ are expected to be equal to 0 depending on the considered graph of the code. We thus decompose the weights as $\boldsymbol{\omega}^{(\tilde{\mu})} = \sigma(\boldsymbol{w}_G) \odot \boldsymbol{w}_M$ where $\sigma(\boldsymbol{w}_G)$ is the same gating mechanism while $\boldsymbol{w}_M$ is a trainable weighting mechanism specific to this marginalisation operation.

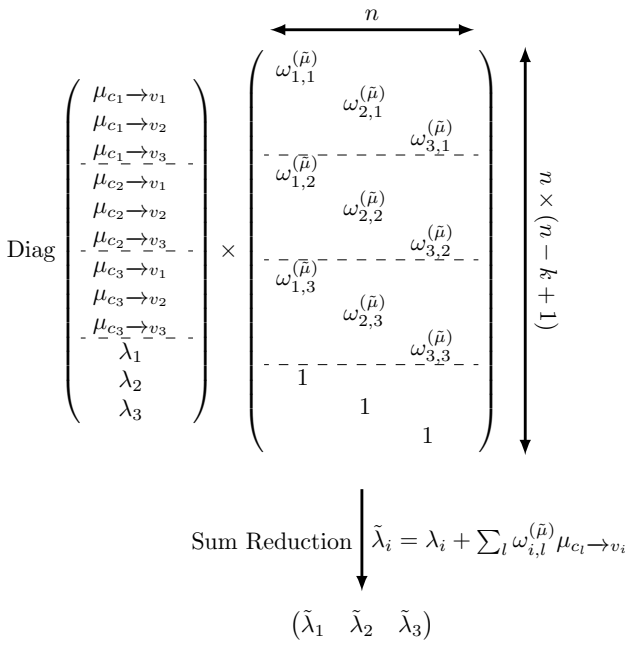


Figure 5.9: Naive Implementation of Eq. (5.44) using a Matrix Multiplication
Note that the $\omega^{(\tilde{\mu})}$ weights are different from the $\omega^{(\tilde{\mu})}$ used for the variable to check nodes messages computation.

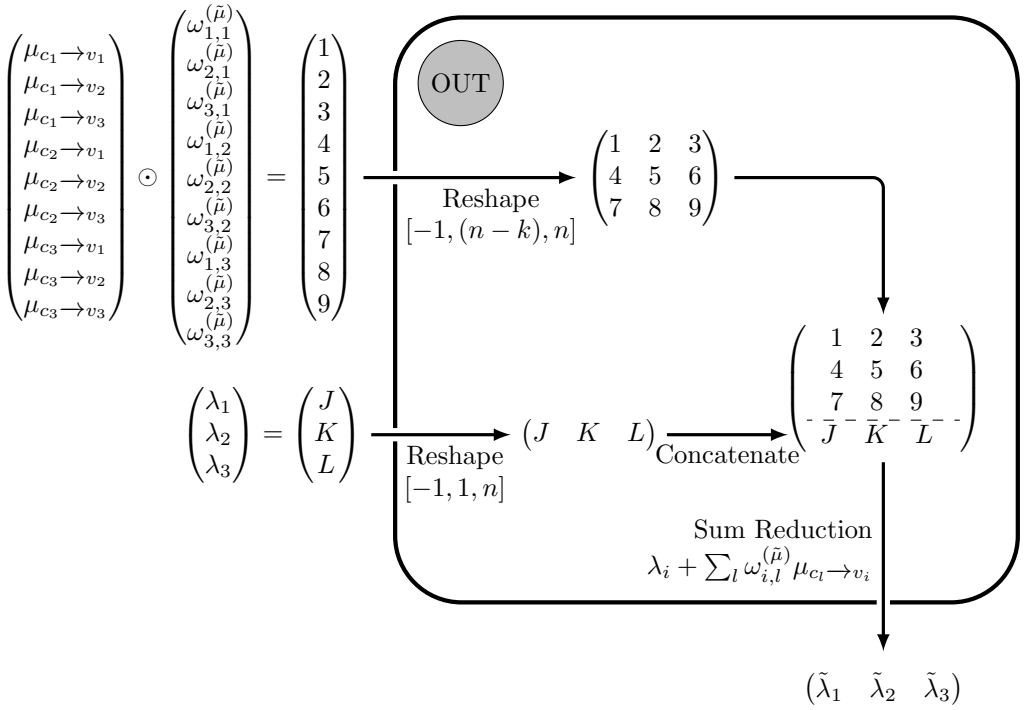


Figure 5.10: Efficient Computational Graph for Eq. (5.44)

	Naive (Fig 5.9)	Improved (Fig 5.10)
# multiplications	$\mathcal{O}(n^2 \times (n-k))$	$\mathcal{O}(n \times (n-k))$
# additions	$\mathcal{O}(n^2 \times (n-k))$	$\mathcal{O}(n \times (n-k))$
Biggest tensor allocation	$\mathcal{O}(n^2 \times (n-k))$	$\mathcal{O}(n \times (n-k))$

Table 5.4: Complexity - Output Marginalisation Operation

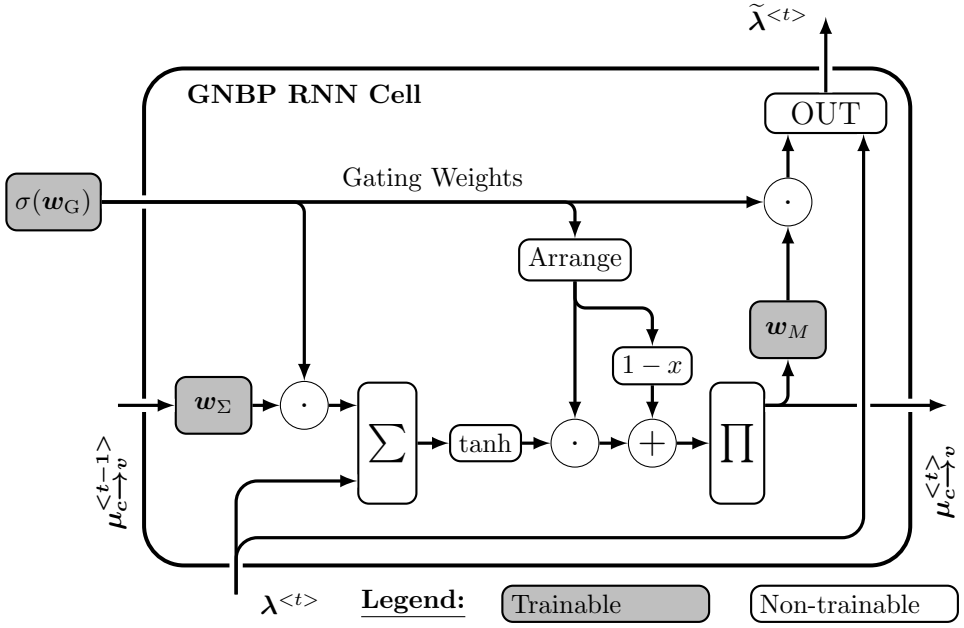


Figure 5.11: Gated Neural Belief Propagation Recurrent Neural network Cell
The proposed cell is a generic representation of one iteration of BP. The RNN allows to sequentially execute several iterations of the decoder while sharing the decoding parameters. $\boldsymbol{\lambda}^{<t>}$ and $\widetilde{\boldsymbol{\lambda}}^{<t>}$ denote the *a priori* and *a posteriori* LLR vector respectively provided and computed at decoding iteration t. While the *a priori* vector is usually the same at each iteration, the *a posteriori* vector resulting from the decoding may vary from one iteration to another. $\boldsymbol{w}_\Sigma$ and $\boldsymbol{w}_M$ are the NBP weights used to improve the decoding performance. $\sigma(\boldsymbol{w}_G)$ are the gating weights used to define the decoding graph structure. The three parameters vectors can be trained and are shared from one iteration to another under the RNN property. $\sigma(\boldsymbol{w}_G)$ weights are represented outside the cell as they might be provided by an external model in some of the further described implementations of Sections 5.4 and 5.5. *N.B.*: $\odot$ and $\oplus$ denote element-wise multiplication and addition.

Recurrent Neural Network Cell

BP decoding being an iterative algorithm, the aforementioned operations must be repeated several times to reach good performance. RNN can be used to execute such iterative decoding in a NN framework [101]. Using the previously described operations, a structured RNN cell, that will be referred to as *Gated Neural Belief Propagation* (GNBP), tailored to perform low-complexity NBP decoding of LBC is proposed in Fig. 5.11. The RNN cell (see Chapter 2 for more details on RNN structures), inspired by gated cells such as LSTM [43] or GRU [44], is built around two types of trainable weights:

- **Gating weights:** $\boldsymbol{w}_G$ represent the topology of the FG, *i.e.* the PC matrix of the code, and are used to select messages accordingly during the different steps of the decoding process. To represent such binary selection behaviour a sigmoidal activation function σ is applied to these weights.
- **NBP weights:** The $\boldsymbol{w}_\Sigma$ and $\boldsymbol{w}_M$ weights are used to improve the performance of the decoding scheme similarly to the NBP mechanism described in [101]. They should be real valued and centred around 1 to ensure weighting of the messages and not selection.

All the weights are shared between different RNN iterations (following known properties of RNN). Gating weights are also distributed inside any given iteration between the different operations of the BP algorithm; see Fig. 5.11. The structured NN architecture ensures the learned decoding algorithm is similar to a NBP but allows the learning of the code's factor graph's topology. It can be noted that the chosen architecture only weights the messages at certain steps. From conducted experiments, using different weighting mechanisms before $\boldsymbol{\Sigma}$ and **OUT** operations - *i.e.* $\boldsymbol{w}_\Sigma$ and $\boldsymbol{w}_M$ - seems performing. In most of the trials, using a weighting mechanism before the $\boldsymbol{\Pi}$ operation was detrimental to the performance of the model. This might

partially be explained by the numerical sensitivity of the product reduction and arctanh function used in $\mathbf{\Pi}$ block. The weighting of the input LLR (inside the RNN cell) did not prove efficient either.

5.4 Learning to Decode - Blind Neural Belief Propagation Decoder

In the previous section, an efficient and generic description of NBP decoding algorithm was proposed in the form of a fully differentiable RNN architecture. In the present section we look at how ML procedure can be applied to such a structure to learn to decode LBC. At first, we study how the proposed GNBP RNN decoder is able to learn an improved decoding scheme without any prior knowledge of the coding scheme used at the emitter. Indeed, the original NBP publication [101] assumes a perfect knowledge of the code by the receiver to define the NN architecture. In the present work, the proposed NN architecture is used to perform blind NBP decoding of LBC without prior knowledge of the coding scheme used. Furthermore, while several publications discussed improvements of NBP using pruning [102, 155], weights sharing [144] or active sampling [105], none of them, to the best of our knowledge, studied the learning of NBP for LBC without prior knowledge of the code.

5.4.1 System model

A simple communication scheme is considered for the rest of this chapter (see Figure 5.12). Inputs words are encoded based on a generator matrix $\boldsymbol{G}$. A BPSK modulation is applied and the symbols are sent through an AWGN channel with real noise power $N_0/2$. LLR of the received samples are decoded based on a PC matrix $\boldsymbol{H}$.

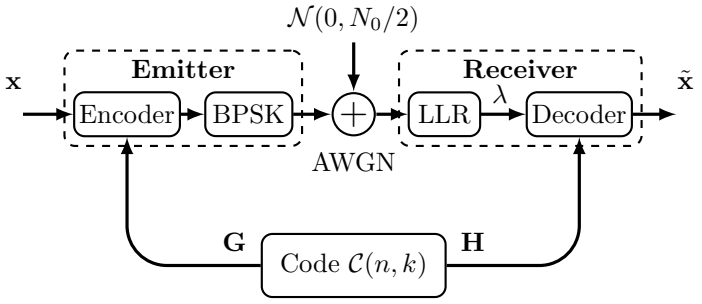


Figure 5.12: System model of the considered communication scheme.

5.4.2 An Auto-Encoder Structure to Learn to Decode

An AE structure based on the previously described GNBP RNN cell is defined (see Figure 5.13). In an AE learning scheme, the inputs are identical to the associated labels (hence the *unsupervised* or *semi-supervised* terms often used to refer to such learning structures). The proposed AE model is defined as an end-to-end Tensorflow graph where input words of size k are provided to a LBC encoder which applies a systematic coding scheme defined by its generator matrix $\boldsymbol{G}$. The coded words of size n are modulated using a BPSK modulation and sent over an AWGN channel. Upon reception, the received samples are soft demodulated to provide a vector of LLR $\boldsymbol{\lambda}$ to the decoder. In order to ensure the consistency of LLR amplitude spans across all training samples of a batch, LLRs are normalised by their mean absolute value per codeword. This normalisation should not affect the decoding capabilities of the system as the high versus low confidence LLR ordering remains unchanged. The normalized LLR are broadcasted and weighted on a per-iteration basis and provided to the GNBP RNN decoder as follows:

$$\boldsymbol{\lambda}^{<t>} = w_{\text{in}}^{<t>} \frac{n\boldsymbol{\lambda}}{\sum_{j=1}^{n} |\lambda_j|} \quad \forall t \in \{1, \dots, n_{\text{iter.}}\} \tag{5.46}$$

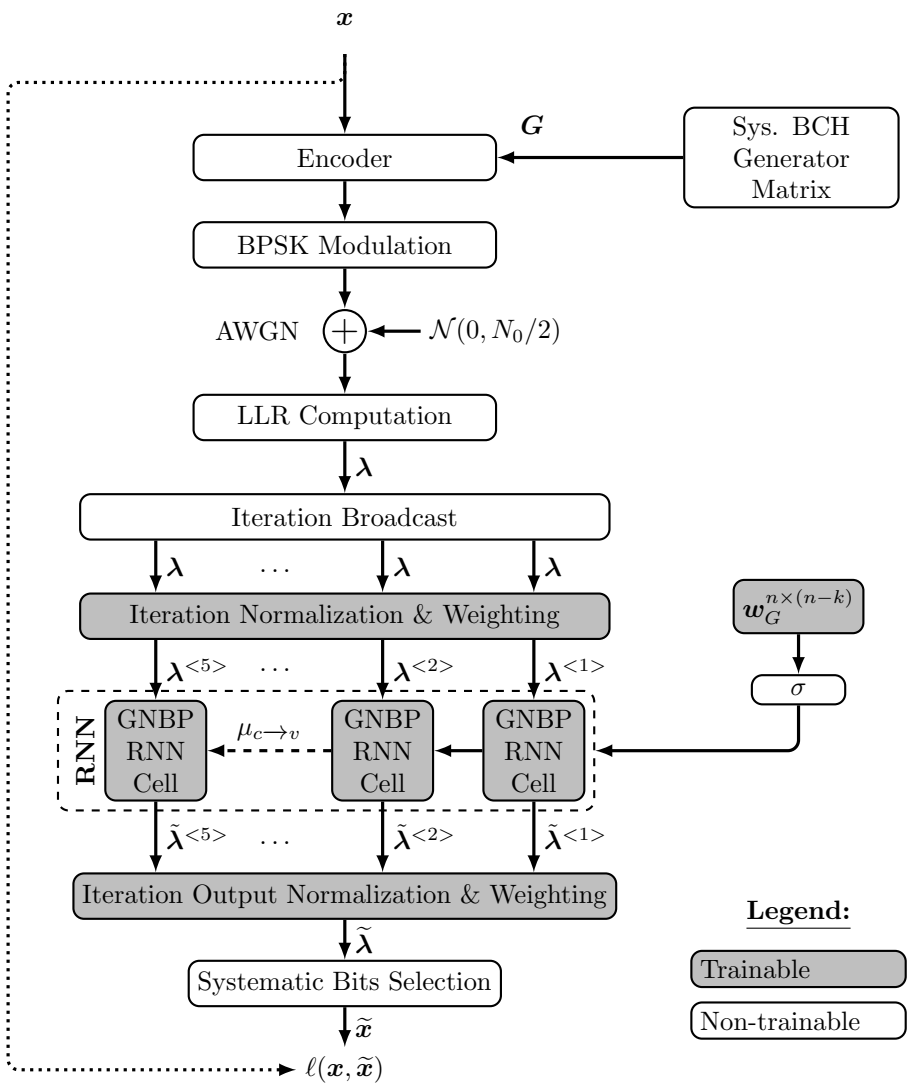


Figure 5.13: Proposed Auto-Encoder Architecture
GNBP RNN, described in Section 5.3, is used to learn to decode without prior knowledge of the code used at the emitter (with $n_{\text{iter}} = 5$ BP iterations). Note that while all the blocks of the transmission chain are defined as Tensorflow functions, no training of the encoder part is considered in this section.

In this section, the gating mechanism of the GNBP cell is based on a parametric "refined gate function" applied to the trainable code weights $\boldsymbol{w}_G$ [156], defined as:

$$\begin{cases} \boldsymbol{r} = \text{sigmoid}(\boldsymbol{w}_G \odot \boldsymbol{w}_r) \\ \boldsymbol{f} = \text{sigmoid}(\boldsymbol{w}_G \odot \boldsymbol{w}_f) \\ \sigma(\boldsymbol{w}_G) = (2\boldsymbol{r} - 1) \odot [\boldsymbol{f} \odot (\boldsymbol{f} - 1)] + \boldsymbol{f} \end{cases} \tag{5.47}$$

where $\boldsymbol{w}_r$ and $\boldsymbol{w}_f$ are two sets of parameter of the same size as the original gating weights $\boldsymbol{w}_G$.

This function closely acts as a sigmoid function but is supposed to allow better gradient back-propagation in the saturation regime of the sigmoid thus improving the learn-ability of the gates (at the cost of additional trainable parameters).

Instead of extracting only the result of the last BP iteration, the outputs of all iterations are re-combined using a weighted sum. Thus, the RNN layer is used in a "many-to-many" configuration. This approach is inspired by [157, 144] and might allow an easier back-propagation of the gradient by injecting it back at each BP iterations:

$$\tilde{\boldsymbol{\lambda}} = \sum_{t=1}^{n_{\text{iter.}}} w_{\text{out}}^{<t>} \frac{n\tilde{\boldsymbol{\lambda}}^{<t>}}{\sum_{j=1}^{n} |\tilde{\lambda}_j^{<t>}|} \tag{5.48}$$

The systematic bits are selected as the first k LLR of the decoded code-words and are fed to a sigmoid function to assign them binary-like values. In an AE learning scheme, the inputs are identical to the associated labels. Hence, the loss function is computed as the BCE between the input and decoded binary words:

$$\ell(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \text{BCE}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \text{BCE}\left(\boldsymbol{x}, \sigma(-\tilde{\boldsymbol{\lambda}})[0:k]\right) \tag{5.49}$$

Unless otherwise specified, the RNN decoder is configured to execute the equivalent of 5 BP iterations. The total number of trainable parameters of the proposed architecture is defined as follows:

$$n_{\text{param.}} = \underbrace{n(n-k)}_{\boldsymbol{w}_\Sigma} + \underbrace{n(n-k)}_{\boldsymbol{w}_M} + \underbrace{n(n-k)}_{\boldsymbol{w}_G} + \underbrace{2n(n-k)}_{\text{Refine Gate } \sigma} + \underbrace{2n_{\text{iter.}}}_{\boldsymbol{w}_{\text{in}} \,\&\, \boldsymbol{w}_{\text{out}}} = 5n(n-k) + 2n_{\text{iter.}} \tag{5.50}$$

The use of shared parameters and a structured architecture allow for a controlled number of trainable parameters, improved explainability and possibly better generalisation capabilities for larger codes [145, 152]. Furthermore, for a fixed code-rate, the number of parameters of the AE evolves in $\mathcal{O}(n^2)$ ensuring the scalability of the proposed architecture on that matter, at least in the short-to-medium block size regime of interest to the present study.

5.4.3 Model Training and Results

Weights	# Trainable Parameters	BCH(15,7) decoder	BCH(15,11) decoder
PRE-PROCESSING STAGE			
$\boldsymbol{w}_{\text{in}}$	$n_{\text{iter.}}$	5	5
GNBP RNN			
$\boldsymbol{w}_G$	$n \times (n-k)$	120	60
Refine Gate σ	$2 \times n \times (n-k)$	240	120
$\boldsymbol{w}_\Sigma$	$n \times (n-k)$	120	60
$\boldsymbol{w}_M$	$n \times (n-k)$	120	60
POST-PROCESSING STAGE			
$\boldsymbol{w}_{\text{out}}$	$n_{\text{iter.}}$	5	5
TOTAL	$2 \times n_{\text{iter.}} + 5 \times n \times (n-k)$	610	310

Table 5.5: Number of Trainable Parameters of the Proposed Decoder

Information words $\boldsymbol{x}$ are encoded using systematic versions of either BCH (15,11) or BCH (15,7) codes described in [158]. For both codes, the overall complexity of the model is described in Table 5.5. The AE framework enables a simple training process as the loss function can be computed as the BCE between initial information words $\boldsymbol{x}$ and decoded words $\widetilde{\boldsymbol{x}}$. Many hyperparameter configurations were tested, and the one that yielded the best results is described thereafter. For both codes, training is performed using randomly chosen information bits, divided in words of size k. The number of words used for training corresponds to 10 times the total number of possible words, 2^k. The SNR used during training phase is fixed at 4 dB. RMSProp [159] optimiser is used with a triangular cyclic learning rate scheduler [160] oscillating between learning rates of 10^{-2} and 10^{-1}. The model is trained during 250 Epochs. Except $\boldsymbol{w}_G$ weights that are initialized using Glorot uniform initializer [161], all the other weights are initialised to 1. A shifted L_2 regularisation mechanism is applied to penalise learned NBP weights that lie too far from 1.

Results

The model is evaluated on a dataset composed of randomly chosen words. To obtain reliable results, instead of fixing the number of testing words, the number of errors to reach has been fixed. The performance of the best model among 50 trainings, denoted as "Blind GNBP", is compared with Maximum Likelihood (ML) [158] and standard BP decoding baselines for both codes as depicted in Fig. 5.14. All the performance curves are displayed in *Block Error Rate* (BLER) versus E_b/N_0. Proposed model is able to learn to decode both codes, without prior knowledge, at least at the level of performance of a standard BP decoder, and even outperforms it in the case of BCH (15,11) code thanks to NBP approach.

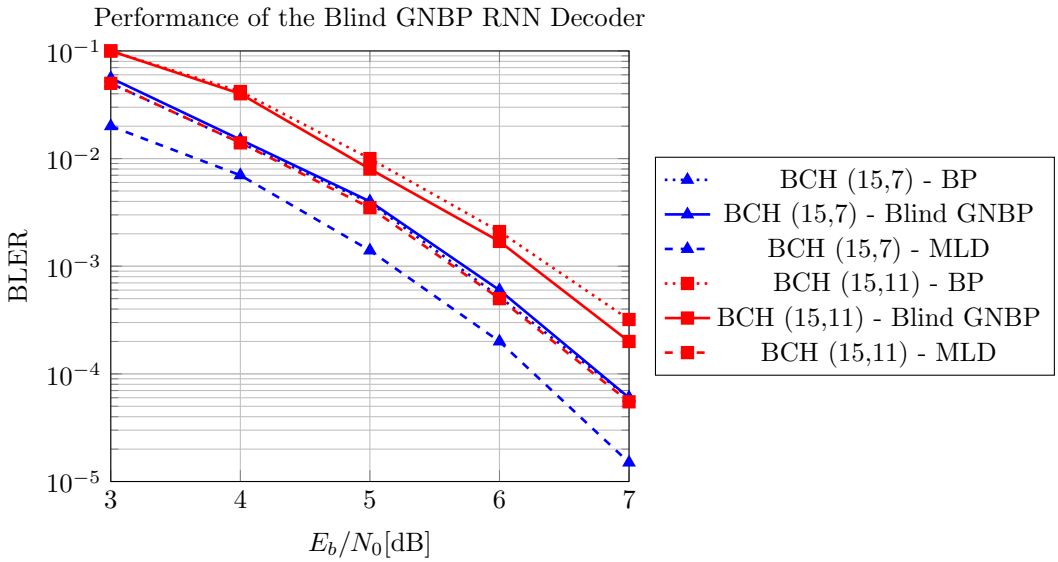


Figure 5.14: Block Error Rate of the Blind Gated Neural Belief Propagation Decoder
The proposed model (solid lines) is compared with standard BP (dotted lines) and Maximum Likelihood (dashed lines) decoders for BCH (15,11) (red - square markers) and BCH (15,7) (blue - triangle markers) codes.

This efficient GNBP architecture thus enables a blind approach where no prior knowledge of the used coding scheme is needed at the receiver side and leverages NBP technique to achieve potential performance gains over standard BP decoders. The computational graph has been designed to be low-complexity, generic and scalable to bigger codes. Now that the faisability of learning to decode an arbitrary LBC coding scheme has been demonstrated, a fully trainable end-to-end architecture is described in the next section, to jointly discover an efficient coding and decoding scheme, and particularly while considering longer and thus more challenging code sizes of up to 128 bits.

5.5 Learning to Code - Neural Belief Propagation Auto-Encoder for Linear Block Code Design

This section investigates the joint learning of short to mid block-length coding schemes and associated BP like decoders using ML techniques. An interpretable AE architecture is proposed, ensuring scalability to block sizes currently challenging for ML-based linear block code design approaches.

5.5.1 Learning to Code: Main Challenges

Differentiability of the Models

Gradient based ML techniques require the differentiability of the models to be trained. Proposed AE models in the domain of PHY layer can present several non differentiability issues related to the channel, the digital modulation, and when considering channel coding, the extensive use of finite field (*e.g.* $\mathbb{F}_2$) and associated modulo arithmetic based on *eXclusive-OR* (XOR). The trainability of the model can also be impacted by many other optimisation issues such as non-convexity, non-smoothness and non-linear separability of the loss function. Differentiable structures and approximations must be defined to circumvent such issues, thus allowing the back-propagation of gradients during the model training.

Scalability - the "Curse of Code Dimensionality"

A key factor for the adoption of AI/ML-based (de)coding methods is the scalability of the proposed models and their associated training schemes. As the number of possible words in an (n, k) code is 2^k, even for relatively small codes it becomes impossible to use exhaustive training data-sets that include all words. In the previous section, reasonably small codes were considered so that a naive approach to the learning of a decoding scheme was possible. In this section, codes with sizes up to 128 bits will be considered and it will no longer be possible to consider such an approach. It is therefore necessary to find structured architectures and learning procedures that allow the training on a subset of the possible data-words while ensuring the generalisation to the remaining words during execution of the coding scheme in real operation conditions.

5.5.2 Linear Block Code Neural Belief Propagation Auto-Encoder

This section focuses on the joint design of short to medium sized LBC matrices and associated weighted BP decoding structures using ML techniques. The idea is to guide the design of code matrices based on the decoder structure rather than optimising the decoding of a potentially flawed code with respect to the decoder.

Linear Block Code Neural Encoder

Unlike the previous section where the parameters of the coding scheme were learned only from the decoder's side, in this section we seek to learn both a coding and a decoding scheme, hence the need for a trainable and differentiable LBC encoder, hereafter referred to as a neural LBC encoder. The objective of this block is to encode binary words of size k into coded representation of size n based on a set of trainable parameters representing the code's generator matrix $\boldsymbol{G}$. The latter will be provided, in the present section, by an external "bridge" model, as will be introduced in Figure 5.16. The proposed encoder block describes linear combination in $\mathbb{F}_2$ by representing the XOR function as a differentiable product of bipolar symbols (Figure 5.15):

- Binary words of size k are converted into bipolar form (*i.e.* $\{0; 1\}$ are mapped to $\{+1; -1\}$).
- Inputs are broadcasted and multiplied by trainable external binary weights, *i.e.* the code's generator matrix $\boldsymbol{G}$, thus selecting the bits participating in each of the n encoding equations.
- For the variables that were not selected the neutral element of the product $(+1)$ is added.

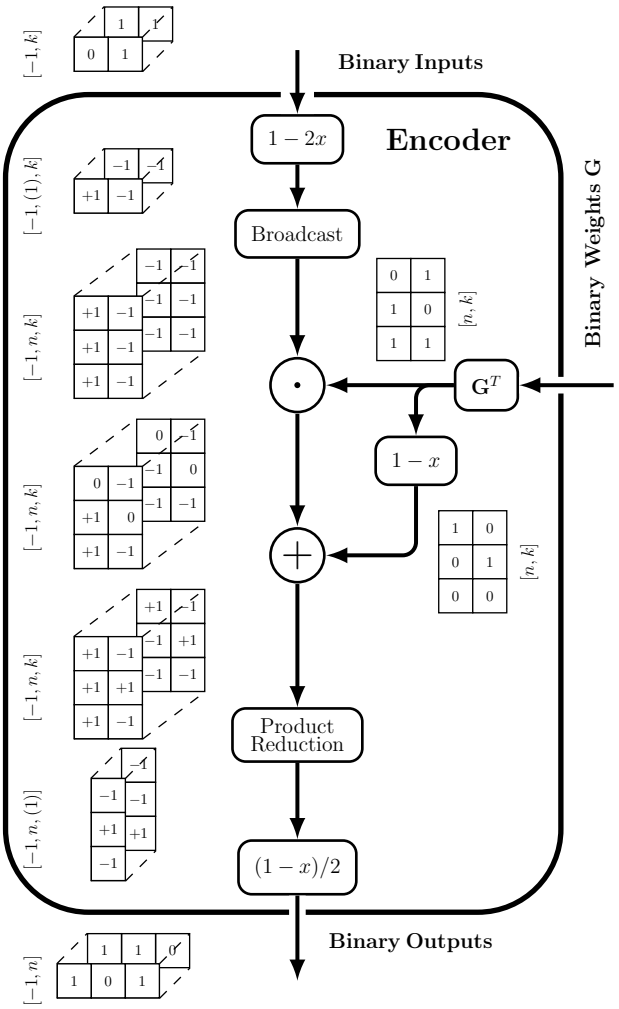


Figure 5.15: Proposed Linear Block Code Neural Encoder Model
Toy example of a (n=3, k=2) encoding. The proposed block works only for binary inputs and weights $\boldsymbol{G}$. To represent the $\mathbb{F}_2$ XOR as a differentiable operation, a product reduction of binary inputs in their bipolar form is used. *N.B.*: The last conversion step from bipolar to binary notation is not necessary when the up-coming modulation is a BPSK.

- Finally, a product reduction is performed for each of the n encoding equations.
- *(Optional)* The results can be converted back to binary form. This is not interesting in the present case were the bipolar words can be directly transmitted as BPSK symbols.

The proposed architecture works if the weights $\boldsymbol{G}$ and inputs are indeed binary. Since the encoder inputs are also those of the AE model, it is not difficult to guarantee this condition for the latter. However, this is more difficult in the case of the trainable weights fed to the encoder which require the loss function to remain differentiable with respect to them. Several solutions could be thought of, such as sigmoidal activation applied to the weights (such as the "refine gate mechanism" that was used in previous section for blind decoding), regularisation methods, sampling techniques [162], discrete functions trained using REINFORCE algorithm [163], Straight-Through Estimator [164] or a stochastic binariser [165] as used in [107]. In this work, a differentiable approximation of the step function, inspired by the idea of differentiable bypass [166], is used. The idea is to apply a non-differentiable step function on a forward pass of the NN and approximate the gradient by a differentiable approximation of the step, a sigmoid in the present case, during the backward pass. This functional block will be referred to as a *Differentiable Step Function* (DSF):

$$f(x) = \text{step}(x) \qquad \frac{df(x)}{dx} = \frac{d\sigma(x)}{dx} = \sigma(x)\sigma(1-x) \tag{5.51}$$

where $\sigma(x)$ denotes the sigmoid function. The DSF is applied to the weights $\boldsymbol{G}$ before being used by the encoder as will be further detailed in Section 5.5.2.

Gated Neural Belief Propagation Decoder

The decoder is based on the GNBP RNN Cell introduced in Section 5.3, although with two minor modifications:

- The trainable binary gating weights defining the decoding TG architecture are now provided by an external "bridge model". These weights represent the PC matrix of the code, $\boldsymbol{H}$.
- The refine gate mechanism that was used in Section 5.4 to ensure saturated sigmoid functions is now replaced by the DSF which is applied directly inside the aforementioned bridge model.

Proposed Auto-Encoder Architecture for Linear Block Code Design

The proposed AE architecture for the joint design of a LBC and associated GNBP decoder is very similar to that of previous section which was used to learn to decode LBC, except that we now want to learn not only to decode but also to code. The AE consists of the proposed encoder and decoder models at the transmitter and receiver sides (Figure 5.16). The encoder weights and corresponding decoder gating weights that represent the code matrices are provided by a third "bridge model" to ensure that they are continuously matched. The use of such a weight sharing procedure makes the training easier and thus reduces training time. A simple way to implement this procedure is to restrict the learned code to generator and parity-check matrices in standard forms. It enables a direct and differentiable conversion between $\boldsymbol{G}$ and $\boldsymbol{H}$, as described in Section 5.2.1. The standard form offers the advantage of a reduced number of trainable code parameters. In addition, it reduces the complexity of the encoding and possibly avoids costly decoding in the absence of errors, *e.g.* by computing a *Cyclic Redundancy Check* (CRC) of the complete data-frame before checking the syndromes of it's constituent code-words and eventually initiating decoding procedures. However, the standard form can be detrimental to the performance of a BP-based decoder as it generally leads to denser PC matrices. Input words are encoded, according to the trainable matrix $\boldsymbol{G}$ provided by the bridge model, using the encoder introduced in Section 5.5.2. The resulting code-words are modulated using a BPSK and sent over an AWGN channel. On reception, the LLR of the code-words are computed and normalised

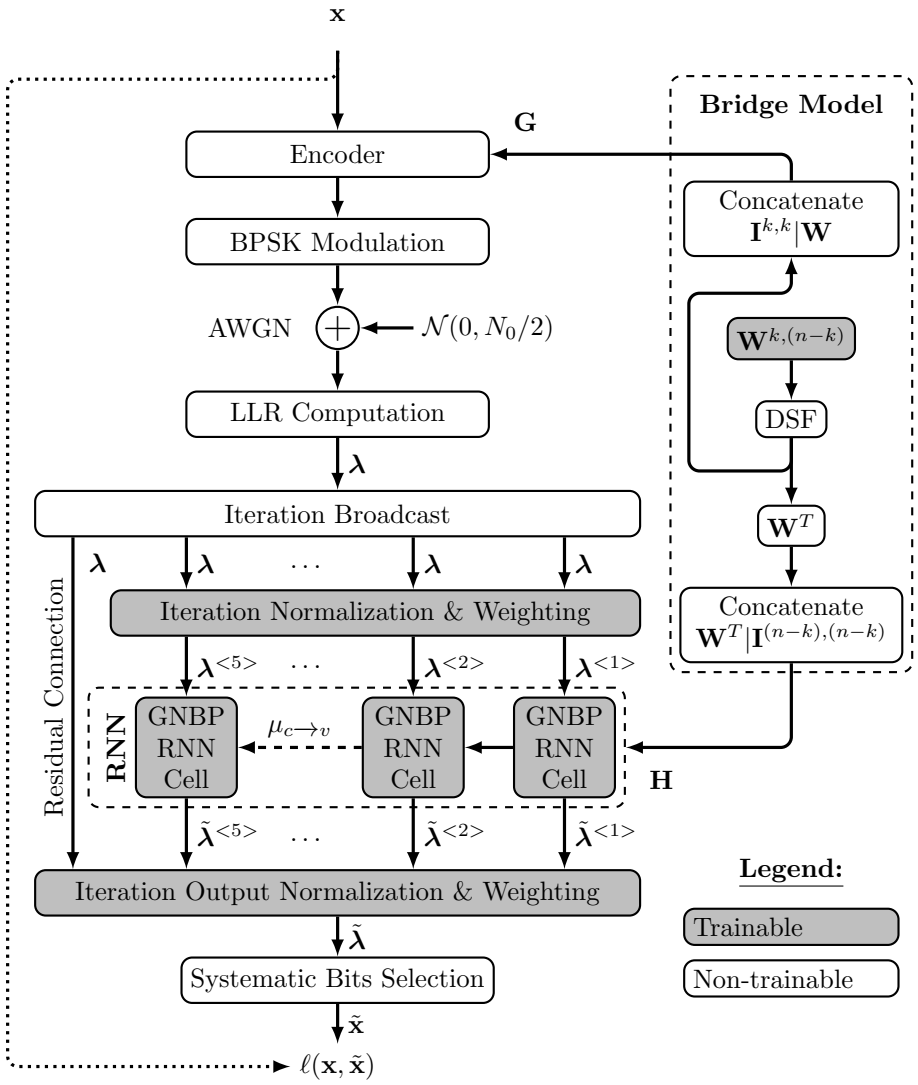


Figure 5.16: Proposed Auto-Encoder Architecture for Linear Block Code Design
Contrarily to previous section, the code used at the emitter is now part of the learning process. The binary encoder's generator and decoder's PC matrices are now provided by a third model that will be referred to as "Bridge Model". Once again, $n_{\text{iter}} = 5$ BP iterations. To facilitate gradient back-propagation toward the emitter, a residual connection is added to bypass the GNBP RNN decoder. *N.B.:* the $\sigma(\boldsymbol{w}_G)$ notation previously used to denote the gating weights of the RNN cell are now denoted by $\boldsymbol{H}$, the equivalent PC matrix.

n	8	31		63		128
k	4	16	11	45	36	64
$n_{\text{param.}}$	91	1181	1471	3089	4385	20491

Table 5.6: Auto-Encoder Number of Parameters

by their per-codeword mean absolute value. The normalised LLR are broadcasted and weighted on a per-iteration basis and provided to the GNBP RNN decoder as follows:

$$\boldsymbol{\lambda}^{<t>} = w_{\text{in}}^{<t>} \frac{n\boldsymbol{\lambda}}{\sum_{j=1}^{n} |\lambda_j|} \quad \forall t \in \{1, \dots, n_{\text{iter.}}\} \tag{5.52}$$

After GNBP decoding based on the trainable matrix $\boldsymbol{H}$ provided by the bridge model, the iterations results and the residual connection are normalised and weighted for recombination as follows:

$$\tilde{\boldsymbol{\lambda}} = w_{\text{residual}} \frac{n\boldsymbol{\lambda}}{\sum_{j=1}^{n} |\lambda_j|} + \sum_{t=1}^{n_{\text{iter.}}} w_{\text{out}}^{<t>} \frac{n\tilde{\boldsymbol{\lambda}}^{<t>}}{\sum_{j=1}^{n} |\tilde{\lambda}_j^{<t>}|} \tag{5.53}$$

As in previous section, the systematic bits are extracted as the first k LLR of the decoded code-words and the loss function is computed as the BCE between the input and decoded binary words (after sigmoid activation):

$$\ell(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \text{BCE}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \text{BCE}\left(\boldsymbol{x}, \sigma(-\tilde{\boldsymbol{\lambda}})[0:k]\right) \tag{5.54}$$

Unless otherwise specified, the RNN decoder is configured to execute the equivalent of 5 BP iterations. The use of a residual connection bypassing the decoder and the iterations results re-combination mechanism are design choices deemed to facilitate gradient back-propagation during training. The total number of trainable parameters of the proposed architecture is defined as follows:

$$n_{\text{param.}} = \underbrace{k(n-k)}_{\text{Systematic code}} + \underbrace{2n(n-k) + 2n_{\text{iter.}} + 1}_{\text{GNBP decoder}} \tag{5.55}$$

Again, the use of shared parameters and a structured architecture allow for a controlled number of trainable parameters (Tab. 5.6). Similarly to the previous AE, the number of parameters of the AE evolves in $\mathcal{O}(n^2)$ (for a fixed code-rate) ensuring the scalability of the proposed architecture. Thanks to the definition of matched $\boldsymbol{G}$ and $\boldsymbol{H}$ matrices in standard form, the bridge model ensures an efficient training phase. The interpretable structure of the model allows the learned code to be extracted and used with other LBC decoding schemes, possibly within legacy, non-AI based systems. The enhanced weighted decoding procedure based on the RNN GNBP cell is jointly trained with the coding scheme. In conclusion, the proposed differentiable AE architecture allows the efficient training of performing LBC suitable for a BP-based decoding.

5.5.3 Data-sets, Training and Evaluation Procedures

In a naive approach, learning a code of size (n, k) would require to use a training data-set including all the 2^k words. This number grows exponentially with the code size and can become prohibitively large. In the case of LBC and under symmetric assumption on the channel and the decoder (implying in the case of NN based decoders the use of symmetric activations, absence of biased units, etc.), it is common to train or evaluate a decoder only with the zero code-word while guaranteeing performance on the complete code [150]. Nevertheless, the training of the encoder is also considered in this work, thus requiring the use of different words. The proposed encoder ensures that all code-words are linear combinations of the basis of the vector subspace of the code. Hence, the model can be trained using only the basis vectors of the words thus reducing the training data-set size from an exhaustive data-set of 2^k words to only k words.

It should also be noted that since the XOR function used in the encoder is a non-linearly separable operator, optimising the weights upstream this function can be challenging for a high input cardinality. Indeed, the non-linear separability of the operator prevents from identifying

the individual contributions of each input in the final results as for a linearly separable operator such as the sum. Still, one way to ensure that the individual contributions are identifiable in the final results is to ensure that there is only one non-zero input contributing to the XOR in each dataset entry *e.g.* by considering only standard basis vectors as inputs, hence the adopted data-set. Similarly, the decoder uses product operators, particularly challenging with respect to the optimisation of the model for similar reasons. Following the same intuition as before, one should carefully choose the channel noise level ensuring a controlled amount of bit flips per code-word. Gaussian noise sampled by the NN model with a fixed E_b/N_0 of 4 dB has shown empirically to provide good results. The model is trained with RMSProp optimiser [159] using batch-size of 64 words and 25 steps per epochs until an early stopping criterion is met (200 epochs without improvement) or the maximum number of epochs, set to 1 000, is reached (a significant margin is used as it has been experimentally observed that the AE usually converges to its best level of performance within the first 100 epochs). The LR is initially set at 10^{-1} and follows a decaying schedule on plateau: when the model does not improve for 50 epochs the LR is reduced by 20%. The validation data-set is randomly sampled from the 2^k possible words in batches of size 64 which are used to monitor the model progress on the real encoding/decoding task, *i.e.* transmitting and recovering any of the possible words with as few errors as possible. A BCE loss function, particularly suited for binary error evaluation, is used. The model weights representing the systematic part of the code, $\boldsymbol{W}$, are initialised uniformly at random on the $[-0.01; +0.01]$ range. The GNBP decoder weights are initialised at '1'. After training, the model is evaluated on randomly sampled words from the 2^k possible words with E_b/N_0 ranging from 0 dB to 6 dB. To ensure a reliable BER evaluation at each E_b/N_0 level, new words are sampled and provided to the model until the 95% confidence interval, computed using Agresti-Coul method [167], on the estimated model BER performance is within the estimated BER +/-5% interval, *i.e.* there is a 95% probability that the true value of the BER lies in-between +/- 5% of the estimated BER[10].

5.5.4 Results

The following sections describe the results obtained with the proposed model in various experiments. At first, (8,4) codes will be used to illustrate the approach and to formulate hypotheses as to why it is likely to produce successful results. A second experiment will study in depth the training of the AE on a (31,16) code size and compare its performance with those of a standard BCH code in standard and non-standard forms. The observed performance gain will be investigated in a short ablation study on a (31,11) code size. Scalability of the approach will then be evaluated on (63,36) and (63,45) code sizes and compared to some existing results, notably with the NBP approach [101]. In addition, a comparison of the complexity of the different decoders will be proposed. Finally, the approach will be tested against other state-of-the-art codes with sizes up to 128 bits, including LDPC codes better suited for BP decoding than BCH codes.

(8,4) Code: Illustration of the Proposed Concept

The model is first tested on a (8,4) code size and compared with a classic hand-crafted code: Extended Hamming (8,4) (Figure 5.17). The performance of the AE model (—■—) is compared to those of MLD[11] (- - -) and BP (-●-) decoders applied to the Hamming code. MLD is also applied, after training, on the code matrix designed by the AE (——). This allows to evaluate the proposed code's raw algebraic performance independently of the decoder's target.

As demonstrated by the MLD performance curves, the learned code is not as good as the Hamming code in terms of algebraic properties. Nonetheless, the proposed AE outperforms, by a significant margin, a standard BP decoder applied to Hamming code.

Figure 5.18 represents the PC matrices as well as the corresponding TG for both Hamming code and the best AE designed code out of 5 trials. Several interesting algebraic properties are to

[10] In this section, results are presented, unless otherwise specified, in BER versus E_b/N_0 [dB] where $N_0/2$ is the variance of the (real) AWGN noise. When the proposed model is compared with other algorithms such as standard BP or NBP algorithms, 5 decoding iterations are considered, unless expressly stated otherwise.

[11] Unless otherwise specified, the MLD decoders considered in this work are based on an exhaustive minimal distance algorithm.

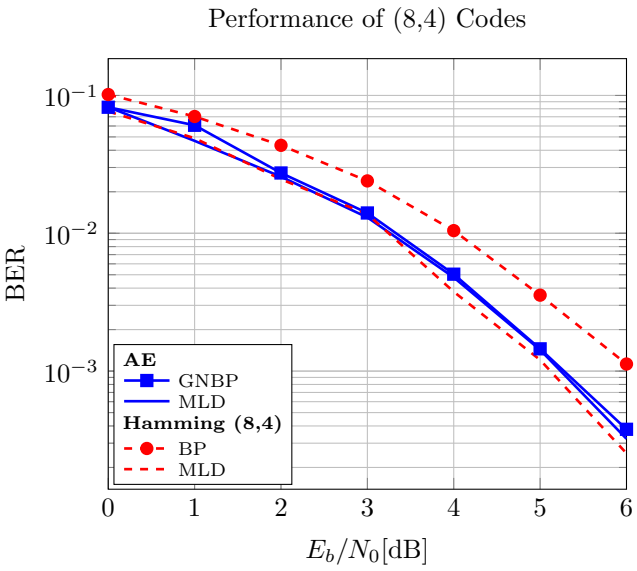


Figure 5.17: Performance Comparison of the Auto-Encoder with that of a Hamming (8,4) Code
The extended Hamming code not being designed for BP decoder performs poorly with it, although its MLD performance are good. On the contrary, a code designed for a BP-like decoder offers higher performance.

be noted when comparing both codes. As one can see the learned coding scheme is an irregular coding scheme. The minimum distance of the learned code is equal to 3 while the minimum distance of the Hamming (8,4) code is equal to 4. This explains why the proposed coding scheme is less efficient in terms of MLD performance. Still, the average Hamming distance is of 4.26 for both codes. As one can see, while the density of the PC matrix is reduced, the girth of the TG is augmented in the case of the learned code. Indeed, the Hamming code presents a density of $\delta = 16/32 = 0.500$ and a girth of 4 with 6 associated cycles, whereas the learned code has a density $\delta = 13/32 = 0.406$ and a girth of 6 with 3 associated cycles. These properties are key in the performance of a BP decoder and can explain the performance gain of the proposed code when associated to a GNBP decoder. These results are a good illustration of the proposed concept: instead of trying to find a code with good properties such as minimum distance, the model proposes a code that improves the performance with respect to the targeted BP-like decoder. In the case of such a short code, it is worth noting that even a naive MLD is not complex and is eventually more efficient than 5 iterations of a BP-like approach. However, this

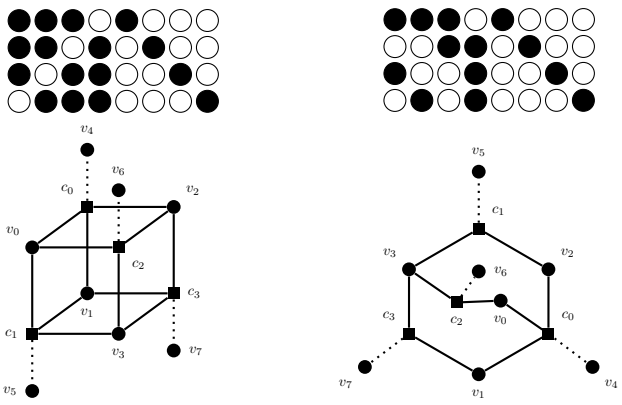


Figure 5.18: (8,4) PC Matrices and their Associated Tanner Graph Representation
(a) Left: Extended Hamming (b) Right: Auto-Encoder. - A non bipartite representation has been used to easily exhibit the cycles present in both graphs. A compact notation of the TG is used where black circle represent variable node whereas black square represents check nodes.

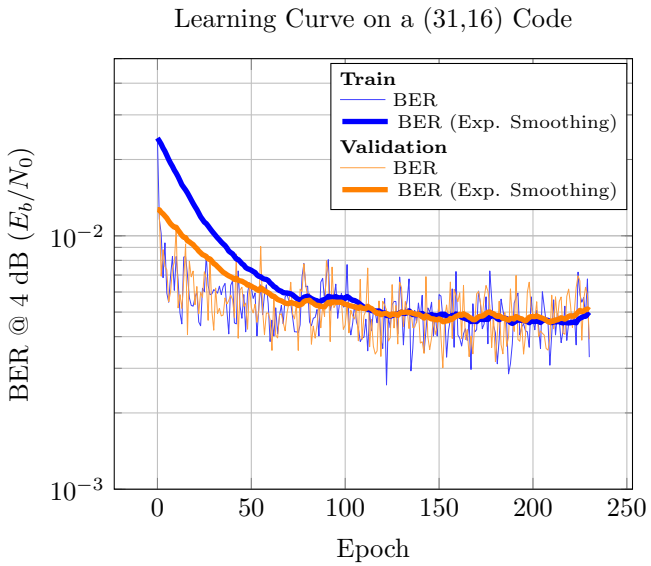


Figure 5.19: BER Evolution During a Training of the Auto-Encoder Model
The noisy curves can partly be explained by the relatively small batch sizes considered here. The best validation BER of 3×10^{-3}, attained around the 150th epoch, is consistent with the performance of the AE displayed on Figure 5.20.

statement quickly becomes untrue as the code size increases.

(31,16) Codes: Statistical Study of Model Training

Model Performance During Training: The proposed model's BER metric is measured during training to assess the existence of a learning process on a (31,16) code. Figure 5.19 shows the BER at the selected training E_b/N_o of 4 dB evaluated at each epoch for the training (——) and validation (——) data-sets. Although noisy, these curves show a clear decrease in BER during training as supported by the corresponding smoothed metrics (—— / ——). During training the BER is roughly divided by a factor 10, from $\approx 2.5 \times 10^{-2}$ to a minimum of $\approx 2.5 \times 10^{-3}$.

Training Repeatability: The design of an AI/ML-based error correction scheme can usually be performed offline and only the best learned coding scheme is selected for implementation in the final communication system. Nevertheless, it is interesting to study the performance consistency of the learned codes as an empirical assessment of the model training reliability. Furthermore, in future work, such model could be integrated into end-to-end communication systems relying on the joint learning of equalisation, (de)modulation and (de)coding, etc. Such scenario would require an efficient and reliable online learning process. The min-max interval of BER performance of the proposed AE (—■—) across 50 independent trainings (Figure 5.20) demonstrates the model's ability to learn a larger code size of (31,16) and shows a reliable and repeatable training process with relatively small variations between runs. Indeed, Figure 5.20 shows approximately 0.5 dB between the worst and best model in the asymptotic regime. The repeatability of model training being demonstrated, the following results of this chapter will present only the best training out of 5 trials as an estimate of the achievable performance of the AE and thus reduce the effective simulation time. Additionally, Figure 5.20 shows the performance of a non-systematic BCH code (31,16) from the literature [158], evaluated on MLD (-·-·-) and BP (-●··) decoders. A systematic version of the BCH code obtained by the *Gauss-Jordan* elimination method is also evaluated with a BP decoder (-●-). Finally, the code learned by the AE is extracted and evaluated on these same MLD (——) and BP (—●—) decoders. Similarly to what has been observed previously with the extended Hamming code, the learned (31,16) code has slightly worse MLD performance than the BCH code (yet almost equivalent in the case of the best model) but is significantly more competitive when using the BP or

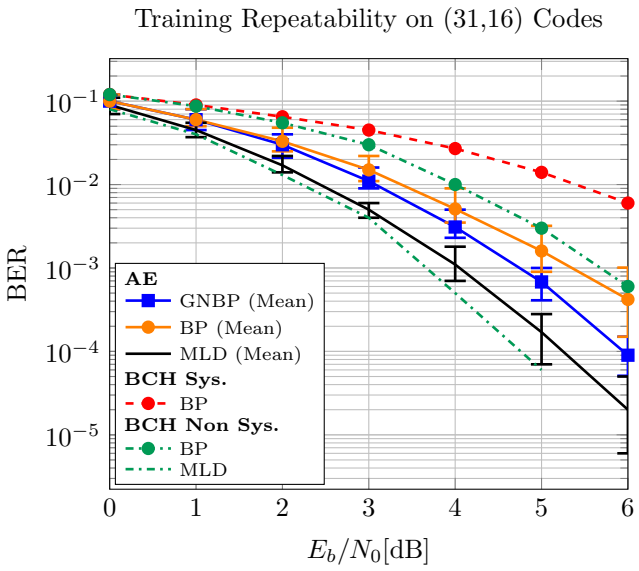


Figure 5.20: Repeatability of the Auto-Encoder Training
At each E_b/N_0 level, the error bars describe the min-max interval of the BER performance across the 50 independent trainings. Performance of systematic and non-systematic BCH codes, decoded using standard BP, are provided for comparison.

GNBP decoders compared to the standard BP decoder applied to both systematic and non-systematic BCH codes. It can also be noted that a systematic form is generally detrimental to the performance of a BP decoder. Indeed, it artificially increases the density of the PC matrix in its redundant part which in turn can reduce the girth of the code.

Correlation Between Decoders Types and their Performances: Figure 5.20 showed the span of the BER performance of the 50 learned codes with different decoders. One question is whether good code performance with one decoder type is generally associated with good performance with the other decoders. To answer this question, the performance correlations of a given learned code between GNBP and BP (Figure 5.21) or MLD decoders (Figure 5.22) and between BP and MLD decoders (Figure 5.23) are provided. For each decoder and E_b/N_0, the BER performance of the 50 learned codes are normalised on the $[-1; +1]$ range and aggregated on a single plot. As one can see, the performance of the learned code evaluated on BP decoder are correlated with their performance on the GNBP decoder. This was foreseeable since the GNBP decoder is derived from the BP decoder. On the contrary, the performance of the learned code evaluated on the GNBP or BP decoders are not correlated with those of the corresponding MLD decoder. The latter observation was also to be expected as the system is not trained in view of a MLD but rather (GN)BP decoding. Furthermore, a good code from a BP decoding perspective is not necessarily a good code from a MLD perspective (as was illustrated in Section 5.5.4).

Degree Distributions of Learned Codes: In an attempt to explain the observed performance from a code design perspective, and even though only small codes are considered here, the normalised variable and check node degree distributions are computed for each of the 50 learned PC matrices and averaged over all the trainings (Figure 5.24 and 5.25). These distributions (· ■ ·) are compared with those at model initialisation (· ● ·). In addition to demonstrating consistent results across trainings and showing that the learning procedure does indeed have an impact on the choice of specific, non-random structures, the learned matrices exhibit unexpected check node degree distributions. The learning procedure seems to encourage overall lower density of the matrices, which was something to be expected given the targeted BP decoder structure. On the other hand, it also seems to promote the definition of a few high degree check nodes. This behavior is not yet explained but is of interest for future studies.

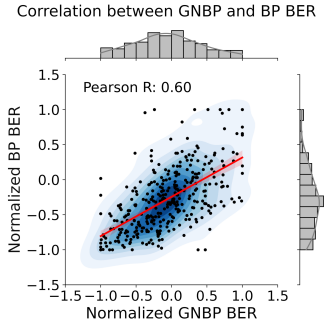


Figure 5.21: Correlogram of the Performance of the 50 Learned Codes Evaluated on GNBP and BP Decoders.

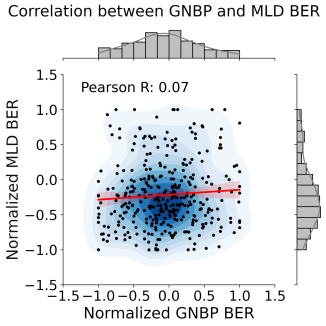


Figure 5.22: Correlogram of the Performance of the 50 Learned Codes Evaluated on GNBP and MLD Decoders.

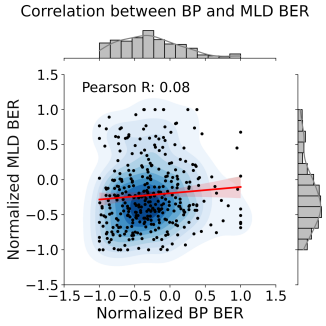


Figure 5.23: Correlogram of the Performance of the 50 Learned Codes Evaluated on BP and MLD Decoders.

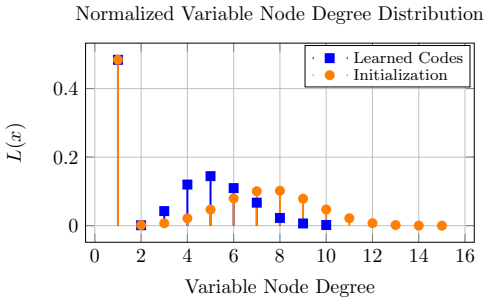


Figure 5.24: Variable Node Degree Distribution Before and After Training
The distribution is averaged over the 50 runs. Note that the peak observed for a degree of 1 is explained by the standard form of the considered codes.

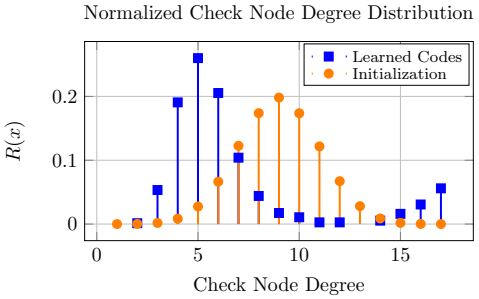


Figure 5.25: Check Node Degree Distribution Before and After Training
The distribution is averaged over the 50 runs. Unexpectedly, some check nodes take on high degrees, leading to all-1 (or almost all-1) rows in the redundant part of the PC matrix.

(31,11) Codes: Auto-encoder Training Procedure and Ablation Study

Several questions arise from observations of previous section. What is the origin of the AE performance gain? Is it due mainly to the use of a weighted decoding procedure such as GNBP, the design of a more efficient coding scheme or a combination of both? Is it preferable to first train the coding scheme and then adapt the GNBP decoder to proposed code or learn jointly both codes and decoders as it was done in previous sections? To try to answer these questions, the AE is applied to a (31,11) code and compared to a BCH code from [158]. The influence of the different elements of the AE and their training schedules is studied to better understand the origin of the performance gain. Different schemes are compared (Figure 5.26):

- The full AE model as described in 5.5.2 to study the influence of both the learned code and the GNBP decoding scheme (—■—).
- The code learned by the aforementioned AE model evaluated on a standard BP decoder to study the influence of the sole learned code (—●—).
- A slightly different AE trained with a differentiable standard BP decoder *i.e.* without any GNBP decoder weights, inputs LLR weightings, residual connection or weighted outputs sum (only the result of the last iteration is selected as in standard BP decoders). This scheme is referred as "AE BP" on the Figure 5.26 (··●··).
- The code learned by this AE BP model used to train a GNBP decoder to study the impact of the training schedule (··■··).

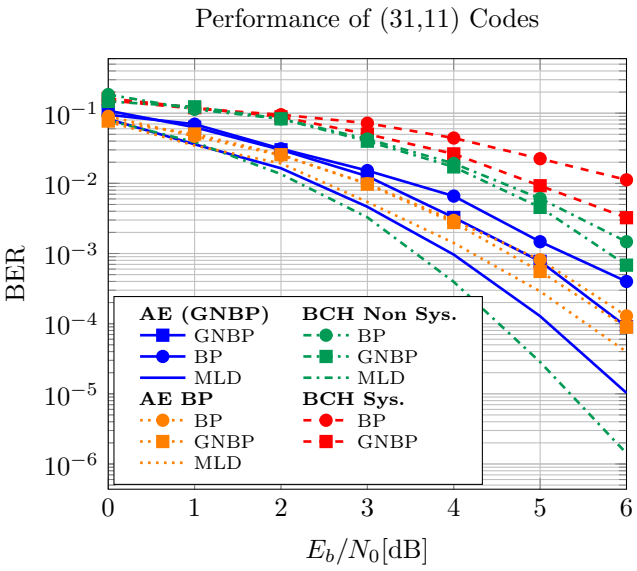


Figure 5.26: Study of the Joint Encoder/Decoder Training Procedure
Evaluation of the relative contributions of the code and decoder design to the final performance.

- The MLD performance of the codes learned by both AE (GNBP) (——) and AE BP (·····) models.

- A reference BCH (31,11) code [158] in both systematic and non-systematic forms evaluated with MLD (-·-·-), standard BP (-●·· / -●-) and trainable GNBP decoders (-■·· / -■-).

Even though a different code rate is considered, similar statements can be made on the performance of the (31,11) code to that of the (31,16) code: the MLD performance of the BCH (31,11) code is better than that of the learned coding schemes. Yet, all the proposed coding scheme outperforms by a significant margin the standard BP decoder applied to both systematic and non-systematic BCH (31,11) codes. It appears that the design of an effective coding scheme and the use of a weighted GNBP decoding procedure can both contribute to the final performance of the AE. However, the training schedule has an impact on their relative contributions to the final performance (Figure 5.26). Indeed, after learning both the code and the GNBP decoder weights, regardless of the training schedule of the different blocks involved, the AE achieves similar performance. Yet, the code learned directly with a BP decoder achieves better performance than the code learned with a GNBP decoder and subsequently evaluated on a BP decoder. Therefore, if the final goal is to run a GNBP decoder, it is more interesting to train the model directly with such a decoder rather than first training the code on a BP decoder and then learn the GNBP weights, which would take twice as much training time for a similar outcome.

On the contrary, if the aim is to use a standard BP decoder, it is more interesting to train the model with such a decoder from the beginning, as this may lead to better overall results compared to a code learned on a GNBP decoder and then evaluated on a BP decoder. It is also worth noting that the performance of the AE model trained and evaluated only on a BP decoder is close to that of the AE model using a GNBP decoder. This suggests that when the coding scheme is well constructed and suffers from few defects with respect to a BP decoder, *e.g.* a small number of short cycles, the benefit of a weighted decoding procedure such as GNBP might be reduced and a simpler BP decoder could be sufficient. Nevertheless, as shown by the GNBP performance curve of the BCH code, the use of a weighted decoder can bring a significant performance gain for codes that exhibit defects with respect to the targeted BP decoding procedure, *e.g.* BCH codes, confirming results from previous works [101]. However, the observed performance gain is not up to the level of the AE joint code and decoder design.

Code	(n, k)	$(63, 36)$	$(63, 45)$
# Code-words	2^k	$\approx 10^{10}$	$\approx 10^{13}$
# Basis code-words	k	36	45
# Different sys. PC matrices	$2^{k(n-k)}$	$\approx 10^{292}$	$\approx 10^{243}$

Table 5.7: "The Curse of Code Dimensionality"

(63,36) and (63,45) Codes: Scalability and Algorithmic Complexity

Scalability to (63,36) and (63,45) Codes: To illustrate the flexibility and scalability of the proposed approach, the AE model is evaluated on higher sizes (63,36) and (63,45) codes. With such sizes the code-books include too many code-words to be included in an exhaustive data-set (Tab. 5.7). Additionally, the trainable part of the systematic matrices include many configurations of '0' and '1'.

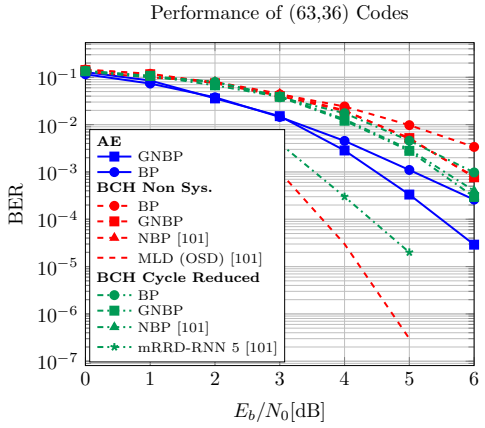


Figure 5.27: Performance Comparison Between (63,36) BCH and AE-Based Codes with Various Decoders.
NBP, mRRD-RNN and MLD (OSD) performances are from *Nachmani et al.* [101]. BP results are consistent with those from [101]. The performance of GNBP decoder demonstrate the equivalence with NBP decoders.

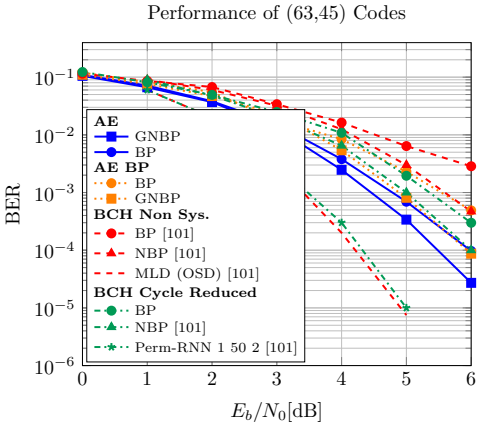


Figure 5.28: Performance Comparison Between (63,45) BCH and AE-Based Codes with Various Decoders.
All results, except the AE results and the cycle reduced BP, are from *Nachmani et al.* [101]. Unlike the case of (63,36) BCH, the BP and NBP results from [101] were not successfully reproduced for the non-systematic code.

The performance of the AE (—■—) is compared to that of the BP (-·●·· / -●-), NBP (-·▲·· / -▲-) and MLD (- - -) - evaluated using *Ordered Statistic Decoding* (OSD) [168] - decoders applied to the (63,36) and the (63,45) BCH codes as described in the original *Nachmani et al.* publication [101]. AE BP scheme results, evaluated with both BP (··●··) and GNBP (··■··), are also provided for the $(63, 45)$ code. In contrast to Section 5.5.4, this scheme is here not on par with the AE GNBP. This difference is currently unexplained and would require further studies, which have not yet been conducted at the time of the writing of this manuscript, to determine its cause. The performance of *mRRD-RNN* (-·✱··) and *Perm-RNN* (-·✱··) parallel BP decoders from [101, 104] are also provided as examples of close to MLD decoders, although at a much higher complexity (Figure 5.27 and 5.28). In addition, the GNBP decoder is applied on BCH (63,36) codes (-·■·· / -■-), demonstrating similar performance to that of the NBP. For both sizes, the AE outperforms BP and NBP decoders with different BCH PC matrices forms. Even the code learned by the AE evaluated with a BP decoder (—●—) outperforms GNBP decoder applied on the BCH codes, at least in the considered E_b/N_0 regime. Again, these curves show that the design of a code tailored to the decoder provides substantial performance gain at no significant complexity cost.

Complexity Study: To further illustrate the advantageous performance to complexity ratio of the proposed approach, the number of decoding operations is computed for the different codes

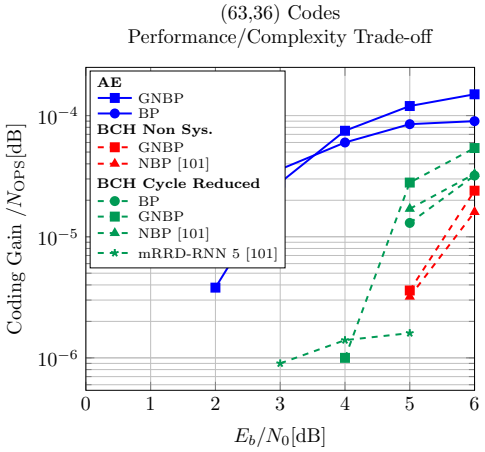


Figure 5.29: Performance to Complexity Ratios of the Different (63,36) Codes and Decoders
This ratio is highlighted by the normalised coding gain per decoding operations. Higher is better.

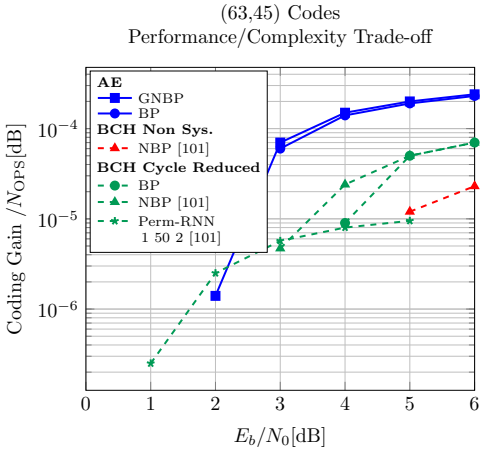


Figure 5.30: Performance to Complexity Ratios of the Different (63,45) Codes and Decoders
This ratio is highlighted by the normalised coding gain per decoding operations. Higher is better.

and associated decoders[12], as summarised in Tab. 5.8. The coding gain (dB) with respect to an uncoded BPSK is then computed for each code and decoder and normalised by the number of operations[13]. The AE is able to design codes that offer a good performance to complexity trade-off based on GNBP or BP decoding (Figure 5.29 and 5.30). Final complexity is also dependent of specific hardware implementations which is not taken into account in the present analysis. Regardless of this point, one should also consider that, during training, the AE requires a higher number of operations as the code's matrices are not yet fixed.

Lower complexity BP-like decoders from the literature such as MS, OMS or *Neural Offset Min-Sum* (NOMS) [103] are not included to the comparison, but there is no strong evidence suggesting that such approaches could not be applied to the proposed AE and further improve the performance to complexity ratio. The densities δ of the learned codes are lower than those of the BCH codes (Tab. 5.8). This was to be expected as BP decoders are supposed to perform well on low density codes, *e.g.* LDPC codes. A regularisation technique could be used in future work to further enhance this interesting property that can both reduce the decoding complexity and potentially improve the performance. Still, this would be a strong *inductive bias* towards the already known solution that BP decoders perform well on low density codes and could potentially conceal innovative code designs that has not yet been thought of and that could be revealed by an AI/ML system.

Comparison with LDPC and other State-Of-the-Art Codes

Comparison with Short (64,k) LDPC Codes - Impact of the Rate: To ensure a fairer study of the proposed system, and since BP decoders were originally engineered to decode such codes, a comparison with LDPC codes of size 64 and with various rates is made (Figure 5.31). The different regular and irregular LDPC codes are generated by the *Progressive Edge Growth* (PEG) algorithm [169] and decoded using BP (-●·· / ··●··) and GNBP (-■··) decoders.

Said codes are compared with the proposed AE model trained with a GNBP decoder and evaluated with both BP (—●—) and GNBP (—■—) decoders. When considering a GNBP decoder,

[12]See Appendices B for detailed derivations. As described by *Nachmani et al.*, *Perm-RNN (1,50,2)* denotes a decoder with one branch, 50 permutations per branch and two NBP iterations between two consecutive permutations. Although it is not a parallel decoder, it requires a total number of up to 100 NBP iterations. Similarly, *mRRD-RNN (5)* denotes a 5 branch parallel decoder each constituted of 30 blocks of two NBP iterations leading to a total number of up to 300 NBP iterations

[13]More precisely, the coding gain is normalised based on the number of multiplications. Such operations are usually considered to be more costly than additions. Moreover, the number of additions is unchanged between BP, GNBP and NBP decoders.

n	k	Code	δ	Decoder	# Operations	
					# SUM	#MULT.
63	36	AE (systematic)	0.18	BP - 5 iter.	11,775	21,475
				GNBP - 5 iter.	11,775	24,525
		BCH (non systematic)	0.29	BP - 5 iter.	23,620	38,880
				GNBP - 5 iter.	23,620	43,740
				NBP - 5 iter.	23,620	65,245
		BCH (cycle reduced)	0.24	BP - 5 iter.	16,490	27,840
				GNBP - 5 iter.	16,490	31,980
				NBP - 5 iter.	16,490	46,715
				mRRD-RNN(5)	989,400	2,802,900
	45	AE (systematic)	0.19	BP - 5 iter.	4,910	11,290
				GNBP - 5 iter.	4,910	13,450
		BCH (non systematic)	0.38	BP - 5 iter.	17,500	47,520
				GNBP - 5 iter.	17,500	51,840
				NBP - 5 iter.	17,500	67,495
		BCH (cycle reduced)	0.28	BP - 5 iter.	8,680	24,720
				GNBP - 5 iter	8,680	27,840
				NBP - 5 iter	8,680	35,275
				perm-RNN(1,50,2)	173,600	494,400

Table 5.8: Estimated Number of Operations for Each Code and Decoder

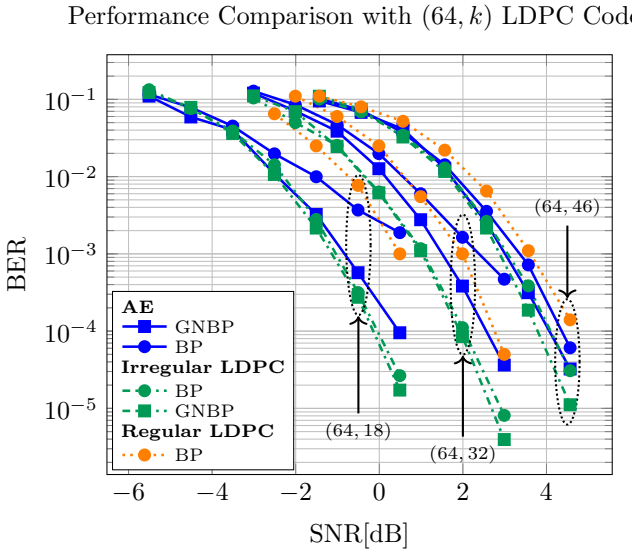


Figure 5.31: Performance Comparison Between Various (64,k) LDPC Codes and The Proposed AE

Different decoder targets are considered. For better readability, and in contrast to the other graphs of this section, the SNR [dB] is used for the x-axis.

and despite the standard form constraint, the AE shows decent performances compared to the LDPC codes. AE performs significantly better than regular LDPC codes and is not far from the irregular codes (half a dB difference in the worst case). For the $\frac{46}{64}$ code rate, the GNBP AE even reaches the performance of the irregular LDPC code under BP decoding and is close to the performance of the same code under GNBP decoding. Under BP decoding, the AE code is not as good as before, especially for the lower code-rates, where the systematic part of the PC matrix becomes prominent.

n	k	Code	δ	Degree Distributions
64	18	AE	0.10	$\Lambda(x) = 46x^1 + x^{10} + 2x^{12} + 4x^{13} + 2x^{14} + 5x^{15} + 2x^{16} + 2x^{17}$ $P(x) = 4x^2 + 7x^3 + 11x^4 + 8x^5 + x^6 + 4x^7 + 2x^9 + x^{10} + 2x^{11} + x^{13} + x^{16} + x^{18} + 3x^{19}$
		Irregular LDPC	0.056	$\Lambda(x) = 45x^2 + x^3 + 18x^4$ $P(x) = 19x^3 + 27x^4$
		Regular LDPC	0.087	$\Lambda(x) = 64x^4$ $P(x) = 20x^5.0 + 26x^6.0$
	32	AE	0.16	$\Lambda(x) = 32x^1 + x^6 + 3x^7 + 7x^8 + 10x^9 + 3x^{10} + 3x^{11} + 3x^{12} + 2x^{13}$ $P(x) = x^4 + 5x^5 + 5x^6 + 4x^7 + 2x^8 + 2x^9 + 5x^{10} + x^{12} + 2x^{13} + x^{15} + x^{18} + x^{26} + 2x^{31}$
		Irregular LDPC	0.079	$\Lambda(x) = 31x^2 + 33x^3$ $P(x) = 31x^5 + x^6$
		Regular LDPC	0.13	$\Lambda(x) = 64x^4$ $P(x) = 32x^8$
	46	AE	0.18	$\Lambda(x) = 19x^1 + 3x^2 + 14x^3 + 9x^4 + 9x^5 + 7x^6 + 2x^7 + x^8$ $P(x) = 2x^8 + x^9 + 5x^{10} + x^{11} + 2x^{12} + 2x^{13} + 2x^{14} + x^{15} + x^{16} + x^{17}$
		Irregular LDPC	0.15	$\Lambda(x) = 17x^2 + 47x^3$ $P(x) = 5x^9 + 13x^{10}$
		Regular LDPC	0.22	$\Lambda(x) = 64x^4$ $P(x) = 14x^{14} + 4x^{15}$

Table 5.9: Degree Distribution of the Codes

The Tab. 5.9 provides the degree distributions of the codes. Similarly to what was observed in Section 5.5.4, it can be seen that the learned codes have degree distributions with significantly higher polynomial degrees than the regular and irregular LDPC codes (Tab. 5.9). This can probably be partly explained by the systematic constraint but could also be a code design strategy. Surprisingly, these much higher degree distributions lead to decent performance on the GNBP decoder. Moreover, it can be noted that the densities of the learned matrices are generally higher than those of the LDPC codes, which is to be expected with a systematic matrix.

Comparison with State-of-the-Art (128,64) Codes - Impact of the Number of Iterations: Finally, the AE is compared at the challenging (128,64) size to binary codes and theoretical bounds from [170] (Figure 5.32). The model is trained and evaluated with a number of decoding iterations ranging from 3 to 10 to study the impact of the latter on the performance. When the number of iterations at evaluation is different from the one used during training, the GNBP decoder is re-trained while keeping the previously learned code fixed. AR3A (-o-) and CCSDS (-o-) LDPC codes [170] display better performance than the best AE GNBP (···□···) but their decoding complexity is also higher with up to 200 decoding iterations. For a fairer comparison the CCSDS code [158] is decoded with 3 (-•-), 5 (-●-) and 10 (-o-) BP iterations, thus reducing the performance gap to half a dB, similar to the one observed between the AE and the LDPC codes of Section 5.5.4. As expected, the performance of the standard and the AE codes improves with the number of iterations. The number of training iterations of the AE also seems to have an impact on the final performance. Figure 5.32 shows that it seems better to train the AE with 3 iterations and subsequently evaluate it with 10 (···□···) instead of training it with 10 iterations from the start (··□··). Another ML-based approach relying on a GA is proposed as an additional comparison, although it uses 20 iterations (-⊗·) [108]. The latter performs well both against standard LDPC and the AE. However, the GA approach is fundamentally different from the one adopted in this section because it attempts to optimise LDPC codes based on existing distributions while the AE learns *ex nihilo*. Although their decoding complexity is not easily comparable to that of the AE, the performance of a Polar code (-▲-·) and a TB-CC (-⬟···) with their respective decoders are also provided [170]. Finally, the performance of an extended BCH is provided, showing poor BP performance (-o··) even though its MLD performance (-·-··) is close to the optimum. As before, the AE outperforms the BCH with a significantly reduced

complexity. These results demonstrate that a systematic LBC of significant size can be learned by a ML procedure, with performance relatively close to that of LDPC codes and a controlled complexity.

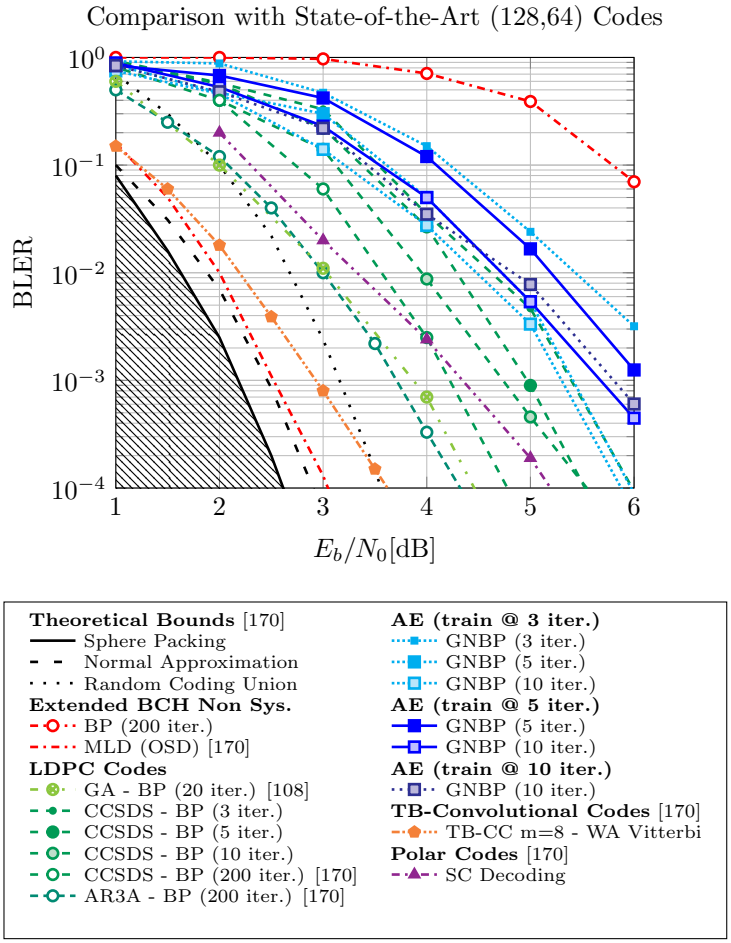


Figure 5.32: Comparison of Different (128,64) Codes and their Respective Decoders from [170, 108]
For the BP decoders, a study of the impact of the number of decoding iterations is conducted, both for the proposed schemes and the conventional ones. For comparison with [170, 108], BLER is used on the y-axis instead of BER.

5.5.5 Discussion and Perspectives

The proposed AE architecture is a performing, low complexity, AI-based approach for the joint design of small-to-medium size LBC and associated weighted decoders. Instead of improving the decoding procedure of codes that otherwise suffers from defects with respect to the targeted decoder, the proposed model supports the joint design of a LBC tailored to a BP-like decoding allowing significant performance gain. The main advantages of the approach are:

- Code agnostic: The proposed approach does not require the knowledge of any LBC scheme from the literature, which allows to build codes of arbitrary size and rate.
- Repeatable: The proposed architecture offers a repeatable training procedure. The study showed results within a 0.5 dB margin over 50 runs for a (31,16) code in Section 5.5.4.
- Adaptable: The AE model can be trained with various decoders, *e.g* BP or GNBP. While GNBP has been shown to improve decoding performance on some codes, this study pointed out in Section 5.5.4 that the AE trained with a BP decoder can have performance close to

that of the AE trained with the GNBP. Although it has not been confirmed in Section 5.5.4, a well designed code might thus not necessarily need a weighted decoding procedure.

- Interpretable and Low complexity: The tightly structured architecture ensures a controlled number of parameters, tractable training procedure and high interpretability compared to black-box ML-based approaches. After learning, the code matrices can be extracted and eventually used in a standard LBC decoder ensuring backward compatibility with legacy, non-AI based communication systems with a satisfactory level of performance. This interesting property also allows the study of the learned matrices in order to better understand the quality of the learned codes or to use them as initialization for other learning algorithms.

- Scalable and Differentiable: As explained in Section 5.5.4, the structured design of both encoder and decoder is a key enabler for the scalability of the approach. The model is able to jointly learn a code and associated decoder only using the k basis vectors of the information space, while generalising well to the many other unseen words even on relativity important code size. Moreover, the different structures presented throughout the chapter ensure the differentiability of the complete architecture.

- Efficient: The study of Section 5.5.4 showed improvement in terms of performance to complexity ratios when compared to other approaches, *e.g.* NBP or high performance but high complexity parallel decoders applied to (63,45) and (63,36) BCH codes. As shown in Section 5.5.4, the performance of the learned codes with size up to $(128, 64)$ are close to those of LDPC codes, which are reference codes for BP decoding procedure. This emphasises the ability of an AI-based system to design competitive LBC.

Still, several perspectives of future works can be outlined from the current proposition:

- The current architecture does not exploit the blind property of the decoder as described in Section 5.4. Instead, a bridge model is used to keep the encoder and decoder matched during training, which makes distributed online learning procedures complex in addition to limiting the codes to standard forms only. The latter usually leads to denser PC matrices which can be a limitation in terms of performance and decoding complexity when considering BP decoders. As such, it would be interesting to further study a non systematic AE architecture.

- The GNBP RNN cell uses a low complexity static decoding strategy which repeat a certain number of time the BP operations under the same set of shared parameters. The decoding performance could be improved by dynamically computing the decoding graph at each decoding iterations of the RNN, based on the inputs and states of the decoder as it is often the case in standard RNN-based approaches. One could learn codes permutations to be applied during the decoding procedure, or inhibit certain parts of the graph, which has been shown to improve significantly decoding performance in parallel decoders [171, 104, 106].

- A (GN)BP decoder was considered in this work but lower complexity decoders from the literature *e.g.* MS, OMS or NOMS, could probably be used within the proposed system to further reduce the complexity of the system while maintaining competitive performance.

- As shown in Section 5.5.4, the learned codes have unexpected degree distributions with high degrees. Further studies are needed to understand the performance of these distributions.

5.5.6 Source Code

All the material necessary to reproduce the experiments listed in this section is available in open access at:

`https://github.com/Orange-OpenSource/GNBP`

5.6 Conclusion of the Chapter

In this chapter, we have examined the issue of applying ML techniques to improve the performance of channel coding mechanisms. The latter impose difficult constraints on the learning procedure due to the intensive use of finite field arithmetic. In a first part, and based on the theory of FG described in Chapter 2, we detailed an iterative decoding method, known as BP, which is notably implemented in 5G networks for the decoding of LDPC codes of data channels. In a second part, following the approach of Chapter 4, we described a structured RNN model capable of executing a weighted and low-complexity version of the BP algorithm. In a third part, we examined how the proposed structure was able to learn to decode a given coding scheme without any prior knowledge. The NN model succeeded in learning to decode BCH codes with performance at least equal to that of a conventional BP decoder. Finally, the approach was taken a step further by defining a complete auto-encoder structure allowing the joint learning of a code and the associated weighted RNN BP decoder. An extensive performance study of this model has been provided, showing performance comparable to that of SOTA LDPC codes with sizes up to 128 bits, which corresponds to the packet size typically encountered in IoT scenarios.

CHAPTER 6

CONCLUSION

6.1 Reminder on the Context of this Work

The future 6G cellular networks will need to support the rapid expansion of the IoT services and use-cases both in terms of traffic and QoS. These scenarios encompass very strong constraints such as high spectral and energy efficiency, reduced hardware cost, complexity and subsequent compute power, etc. All these constraints call for an efficient and low complexity PHY layer design. While AI and ML techniques are expected to be largely integrated in future networks to address the ever-growing complexity of the latter, they are particularly promising for the improvement of PHY layer algorithms, including that of IoT systems.

6.2 Results

In this context, this work looked at how NN and ML could benefits to the constrained PHY layer of such cellular IoT networks. In particular, the proposed study focused on several fundamental base-band signal processing blocks, relying either on binary signals or sampled continuous signal, such as channel coding, demodulation and channel equalisation.

Following the proposed approach and methodology, the first chapter of contributions, Chapter 4, has explained how classical signal processing algorithms can be described in the NN formalism. One incentive behind this approach is to allow these algorithms to be run on hardware accelerators for AI and thus benefit from their interesting properties in terms of efficiency. This type of generic hardware has benefited from years of development, particularly in the areas of CV and NLP, and can now be considered mature. It is already embedded in consumer devices, such as smartphones (albeit for application tasks only), and is therefore widely available off the shelf at a very competitive cost. Therefore, by expressing signal processing algorithms in a compatible form, communication systems could benefit from these hardware targets resulting in increased efficiency. Assuming some changes in the overall system architecture, a mutualisation of AI computing resources could even be envisioned, as the hardware used by the RAN could be exploited to process application tasks at the edge when it is not used by the communication tasks. In the proposed study, equivalent low-complexity NN models have been proposed for single and multi-path linear equalisation tasks. An ML procedure corresponding to the MMSE solution has been applied to parameterise such a NN model, but without seeking a performance gain over the conventional solution. The presence of a learning bias was demonstrated and formally justified. Following a similar approach, a minimal NN model has been proposed for the low-complexity demodulation of M-QAM modulations of arbitrary order. A solution for both soft and hard demodulation was proposed. Finally, the proposed models were integrated into a prototype low-complexity NN-based PHY layer which was tested under real operating conditions using SDR cards and a channel emulator, proving the performance equivalence to the

corresponding conventional solution. The results described in this chapter have been published in a conference paper [12].

The second contribution chapter, Chapter 5, took the previous proposal a step further by using ML procedures to achieve better performance than classical PHY layer algorithms. Indeed, a second advantage of expressing signal processing algorithms using NN structures is that it allows the use of efficient learning procedures, such as GD and BPROP. The chapter starts by describing how a conventional LBC decoding algorithm, namely the BP algorithm, can be expressed using an interpretable and differentiable structure. Then, an ML algorithm is applied to adapt the proposed structure to the decoding of a coding scheme from the literature and thus improve the performance over the conventional solution. Among other things, and contrary to the prior literature, one of the objectives of the proposal is to make the learning of the decoding task possible without any knowledge of the coding scheme used by the transmitter. These first findings led to the publication of a second conference paper [13]. Since one of the performance limitations of the previous approach is due to the code used at the transmitter, an end-to-end approach to the joint design of a coding scheme and the associated decoder is adopted in a second part of the chapter. Based on the LBC decoder structure introduced earlier, a trainable end-to-end AE architecture is proposed. The described structure and learning methods guarantee scalability to block sizes currently challenging for ML-based linear block code design approaches. Codes of up to 128 bits in size were successfully learned, demonstrating performance comparable to SOTA LDPC codes in the short to medium length regime and thus the applicability of ML and NN techniques for the realisation of such tasks. The results of this second part have been published in a journal article [14].

This work contributed to the Hexa-X project, in particular to the WP4 on "AI-Driven Communication and Computation Co-Design" as described in [6, 17, 18].

6.3 Perspectives

The use of AI, in its modern sense, at the PHY layer of communication systems is still in its infancy. As more and more studies show its relevance for almost all parts of the radio interface, new perspectives are opening up, leading to innovative approaches that could result in unexpected paradigm shifts.

On the one hand, the use of neural network-based algorithms opens up new perspectives in terms of hardware architecture that could be used for signal processing in future networks. The shift from expensive, specialised hardware to generic, energy and cost efficient AI accelerators could be a very interesting change. Furthermore, as AI is now used in more and more domains, it could be conceivable in a MEC scenario to delegate AI-based application tasks to these new network hardware resources, when available.

On the other hand, these approaches open up entirely new possibilities in terms of algorithm performance and complexity thanks to advanced ML procedures. In particular, it is now possible to consider the transition from a modular system of specialised blocks to the optimisation of the overall system in an end-to-end manner. Although the joint optimisation of certain subsystems has already been studied, for example in the framework of JSCC, its practical implementation was often very complex. AI now makes it possible to consider such joint optimisation, although it presents many challenges. The latter could even be considered at a larger scale, for example in scenarios such as semantic communications, where the whole communication chain is optimised according to the overall communication objectives.

More generally, as AI research has produced practical solutions that are now widely adopted in all segments of the industry, AI is now often seen as a tool. In such a context, the "conventional" works on many of the issues of the PHY layer, e.g. MIMO, ECC, etc., is still ongoing, but it is now almost certain that they will involve this new tool at some point.

APPENDICES

A Proofs

A.I Sum of Random Binary Variables

Let x_i be a binary random variable that can take the value 0 or 1 with probabilities $p_{\mathrm{x}_i}(x_i = 0)$ and $p_{\mathrm{x}_i}(x_i = 1)$, respectively. Let y be a second binary random variable defined as a modulo sum of n variable x_i:

$$\mathrm{y} = \bigoplus_{i=1}^{n} \mathrm{x}_i \tag{A.I.1}$$

Let also w_i and t_i be random variables given by $\mathrm{w}_i = 1 - 2\mathrm{x}_i$ and $\mathrm{t} = \prod_{i=1}^{n} \mathrm{w}_i$.

Obviously, w_i equals $+1$ or -1 with probabilities $p_{\mathrm{x}_i}(x_i = 0)$ and $p_{\mathrm{x}_i}(x_i = 1)$, respectively. As a consequence:

$$\mathbb{E}\{\mathrm{w}_i\} = p_{\mathrm{x}_i}(x_i = 0) - p_{\mathrm{x}_i}(x_i = 1) \tag{A.I.2}$$

By hypothesis of independence:

$$\mathbb{E}\{\mathrm{t}\} = \prod_{i=1}^{n} \mathbb{E}\{\mathrm{w}_i\} = \prod_{i=1}^{n} [p_{\mathrm{x}_i}(x_i = 0) - p_{\mathrm{x}_i}(x_i = 1)] \tag{A.I.3}$$

t is equal to $+1$ or -1 if an even or odd number of $\mathrm{w}_i = -1$ (or equivalently $\mathrm{x}_i = 1$).
Similarly, y equal 0 or 1 if an even or odd number of $\mathrm{x}_i = 1$.
As a consequence:

$$\begin{cases} p_{\mathrm{t}}(t = +1) = p_{\mathrm{y}}(y = 0) \\ p_{\mathrm{t}}(t = -1) = p_{\mathrm{y}}(y = 1) \end{cases} \tag{A.I.4}$$

So:

$$\mathbb{E}\{\mathrm{t}\} = p_{\mathrm{y}}(y = 0) - p_{\mathrm{y}}(y = 1) = \prod_{i=1}^{n} [p_{\mathrm{x}_i}(x_i = 0) - p_{\mathrm{x}_i}(x_i = 1)] \tag{A.I.5}$$

Finally,

$$\boxed{p_{\mathrm{y}}(y = 0) - p_{\mathrm{y}}(y = 1) = \prod_{i=1}^{n} [p_{\mathrm{x}_i}(x_i = 0) - p_{\mathrm{x}_i}(x_i = 1)]} \tag{A.I.6}$$

B Decoding Complexity

Let $\mathcal{C}(n,k)$ be an irregular LBC of size (n,k) defined by its degree distributions from node perspective $\Lambda(x)$ and $P(x)$. The number of operations involved in each step of the decoding is entirely defined by these two polynomials[1].

B.I Standard Belief Propagation Decoders

The objective is to evaluate the algorithmic complexity of one full iteration of standard BP decoding over the irregular code $\mathcal{C}(n,k)$. The equations for variable to check node messages, check to variable nodes messages and marginalisation computation are recalled:

$$\mu_{v_i \to c_j} = \lambda_i + \sum_{k \neq j} \mu_{c_k \to v_i} \tag{B.I.1}$$

$$\mu_{c_j \to v_i} = 2 \operatorname{artanh} \left[\prod_{k \neq i} \tanh \left(\frac{\mu_{v_k \to c_j}}{2} \right) \right] \tag{B.I.2}$$

$$\tilde{\lambda}_i = \lambda_i + \sum_{k} \mu_{c_k \to v_i} \tag{B.I.3}$$

Equation (B.I.1) involves Λ_i variable nodes of degree i from which are computed i variable to check node messages, each based on i variable (when taking into account the a priori LLR λ) and therefore requiring $i-1$ additions (Figure B.I.1). No multiplication are involved.

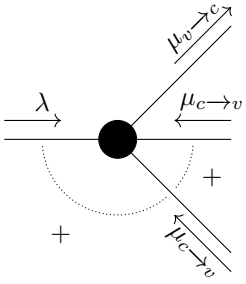


Figure B.I.1: Equation (B.I.1) - Variable to Check - Example of Degree 3 Variable Node.

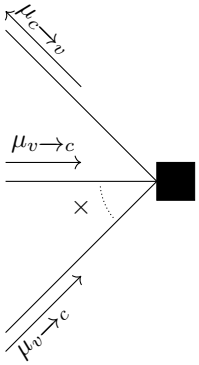


Figure B.I.2: Equation (B.I.2) - Check to Variable - Example of Degree 3 Check Node.

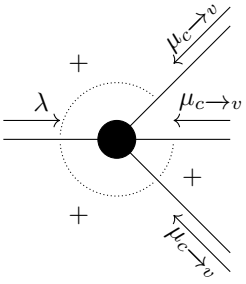


Figure B.I.3: Equation (B.I.3) - Marginalisation - Example of Degree 3 Variable Node.

Hence, the number of additions and multiplications required for the computation of all the variable to check nodes messages are:

$$N_{\Sigma_{v \to c}} = \sum_i i(i-1)\Lambda_i = \sum_i i^2 \Lambda_i - e \qquad N_{\Pi_{v \to c}} = 0 \tag{B.I.4}$$

Similarly, equation (B.I.2) involves P_j check nodes of degree j from which are computed j check to variable node messages, each based on $j-1$ variables and therefore requiring $j-2$ multiplications (Figure B.I.2). No additions are involved.

Therefore, the number of additions and multiplications required for the computation of all the check to variables nodes messages are[2]:

[1] The reader should keep in mind that the following derivations assume that the implementations of the decoding algorithms exploit the sparsity of the code (which is not necessarily the case in the present work).

[2] For simplification, the functions $f_1(x) = 2\text{artanh}(x)$ and $f_2(x) = \tanh(x/2)$ are considered as tabular functions stored in LUT that don't modify the algorithmic complexity.

$$N_{\Sigma_c \to v} = 0 \qquad N_{\Pi_c \to v} = \sum_j j(j-2)P_j = \sum_j j^2 P_j - 2e \tag{B.I.5}$$

Finally, equation (B.I.3) involves Λ_i variable nodes of degree i where a marginalisation involving $i+1$ variables is computed (Figure B.I.3).

As a result, the number of additions and multiplications required for the final marginalisation are:

$$N_{\Sigma_v \to \lambda} = \sum_i i\Lambda_i = e \qquad N_{\Pi_v \to \lambda} = 0 \tag{B.I.6}$$

Hence, the total number of operations required to perform one complete iteration of BP algorithm over an irregular code is:

$$N_{\Sigma_{BP}} = \sum_i i^2 \Lambda_i \qquad N_{\Pi_{BP}} = \sum_j j^2 P_j - 2e \tag{B.I.7}$$

B.II Neural Belief Propagation Decoders

NBP algorithm as described in [101] improves the performances of standard BP by adding trainable multiplicative weights into the BP algorithm. The NBP modifies equations (B.I.1) and (B.I.3) as following:

$$\mu_{v_i \to c_j} = \omega_{i,j}^{(\lambda)} \lambda_i + \sum_{k \neq j} \omega_{i,j,k}^{(\mu)} \mu_{c_k \to v_i} \tag{B.II.8}$$

$$\tilde{\lambda}_i = \omega_i^{(\tilde{\lambda})} \lambda_i + \sum_k \omega_{i,k}^{(\tilde{\mu})} \mu_{c_k \to v_i} \tag{B.II.9}$$

Since the number of operations required to compute messages from check to variable nodes is unchanged from standard BP), the number of operations required to execute equations (B.II.8) and (B.II.9) are:

$$N_{\Sigma_v \to c} = \sum_i i^2 \Lambda_i - e \qquad N_{\Pi_v \to c} = \sum_i i^2 \Lambda_i \tag{B.II.10}$$

And the complexity of the NBP marginalisation equation (B.II.9) is:

$$N_{\Sigma_v \to \lambda} = \sum_i i\Lambda_i = e \qquad \Pi_{v \to \lambda} = \sum_i (i+1)\Lambda_i = e + n \tag{B.II.11}$$

Hence, the total number of operations required to execute one iteration of NBP over an irregular code is:

$$N_{\Sigma_{NBP}} = \sum_i i^2 \Lambda_i \qquad N_{\Pi_{NBP}} = \sum_i i^2 \Lambda_i + \sum_j j^2 P_j + n - e \tag{B.II.12}$$

B.III Gated Neural Belief Propagation Decoders

The GNBP decoder slightly reduces the decoding complexity compared to a NBP as the weighting mechanism is simplified[3] The equations of the GNBP decoder are recalled:

$$\mu_{v_i \to c_j} = \lambda_i + \sum_{k \neq j} \omega_{i,k}^{(\mu)} \mu_{c_k \to v_i} \tag{B.III.13}$$

$$\tilde{\lambda}_i = \lambda_i + \sum_k \omega_{i,k}^{(\tilde{\mu})} \mu_{c_k \to v_i} \tag{B.III.14}$$

[3]During training the GNBP decoder must be able to represent all possible codes of size (n, k). As a result, and as already descirbed in Section 5.5.4, this is equivalent to consider that the code has higher complexity degree distributions of the forms $\Lambda(x) = kx^{(n-k)} + (n-k)x$ and $P(x) = (n-k)x^{(k+1)}$. After training, a tailored decoder can be devised for the learned code matrices.

It follows that the number of operations required to execute equations (B.III.13) and (B.III.14) for a fixed (*i.e* after training) irregular code are:

$$N_{\Sigma_{v\to c}} = \sum_i i^2\Lambda_i - e \qquad N_{\Pi_{v\to c}} = \sum_i i\Lambda_i = e \tag{B.III.15}$$

$$N_{\Sigma_{v\to \lambda}} = \sum_i i\Lambda_i = e \qquad N_{\Pi_{v\to \lambda}} = \sum_i i\Lambda_i = e \tag{B.III.16}$$

As a result, the total number of operations required to perform one complete iteration of GNBP algorithm over an irregular code is:

$$N_{\Sigma_{\text{GNBP}}} = \sum_i i^2\Lambda_i \qquad N_{\Pi_{\text{GNBP}}} = \sum_j j^2 P_j \tag{B.III.17}$$

B.IV Parallel Belief Propagation Decoders

Perm-RNN or mRRD decoders as proposed in [101],[171] and [104] rely on the idea of executing in parallel several BP decoders (or NBP,GNBP, etc.) using carefully chosen permutations of the automorphism group of the code. If one of the M parallel decoders reaches a valid code-word, the decoding process can be stopped. As a result, an upper bound for the complexity of such family of decoder is the number of parallel decoders times the number of iterations per decoder branch times the number of operation associated to one iteration of the considered decoder, *e.g.* BP or NBP as detailed in previous sections.

C BER and BLER Performance

To allow further comparisons, a complete summary of both BER and BLER versus E_b/N_o [dB] performance of all the models presented in Chapter 5, Section 5.5 are provided. The results are also available in open access along with the experiment source code at: `https://github.com/Orange-OpenSource/GNBP`

Table C..1: BER and BLER Performance

n	k	Code	Decoder	BER @ E_b/N_0 [dB]							BLER @ E_b/N_0 [dB]						
				0	1	2	3	4	5	6	0	1	2	3	4	5	6
8	4	AE (sys.)	BP - 5 iter.	8.10^{-2}	6.10^{-2}	3.10^{-2}	1.10^{-2}	5.10^{-3}	2.10^{-3}	4.10^{-4}	2.10^{-1}	1.10^{-1}	7.10^{-2}	3.10^{-2}	1.10^{-2}	5.10^{-3}	1.10^{-3}
			GNBP - 5 iter.	8.10^{-2}	6.10^{-2}	3.10^{-2}	1.10^{-2}	5.10^{-3}	1.10^{-3}	4.10^{-4}	2.10^{-1}	1.10^{-1}	7.10^{-2}	4.10^{-2}	1.10^{-2}	5.10^{-3}	1.10^{-3}
			MLD	8.10^{-2}	5.10^{-2}	3.10^{-2}	1.10^{-2}	5.10^{-3}	1.10^{-3}	3.10^{-4}	2.10^{-1}	1.10^{-1}	6.10^{-2}	3.10^{-2}	1.10^{-2}	4.10^{-3}	1.10^{-3}
		Hamming (sys.)	BP - 5 iter.	1.10^{-1}	7.10^{-2}	4.10^{-2}	2.10^{-2}	1.10^{-2}	4.10^{-3}	1.10^{-3}	2.10^{-1}	1.10^{-1}	1.10^{-1}	5.10^{-2}	2.10^{-2}	1.10^{-2}	3.10^{-3}
			GNBP - 5 iter.	1.10^{-1}	7.10^{-2}	4.10^{-2}	2.10^{-2}	9.10^{-3}	3.10^{-3}	8.10^{-4}	2.10^{-1}	1.10^{-1}	1.10^{-1}	4.10^{-2}	2.10^{-2}	6.10^{-3}	2.10^{-3}
			MLD	8.10^{-2}	5.10^{-2}	3.10^{-2}	1.10^{-2}	4.10^{-3}	1.10^{-3}	3.10^{-4}	2.10^{-1}	1.10^{-1}	5.10^{-2}	3.10^{-2}	8.10^{-3}	2.10^{-3}	5.10^{-4}
31	11	AE (sys.)	BP - 5 iter.	1.10^{-1}	7.10^{-2}	3.10^{-2}	2.10^{-2}	7.10^{-3}	1.10^{-3}	4.10^{-4}	4.10^{-1}	3.10^{-1}	1.10^{-1}	8.10^{-2}	3.10^{-2}	8.10^{-3}	2.10^{-3}
			GNBP - 5 iter.	1.10^{-1}	7.10^{-2}	3.10^{-2}	1.10^{-2}	3.10^{-3}	8.10^{-4}	4.10^{-4}	4.10^{-1}	3.10^{-1}	1.10^{-1}	6.10^{-2}	2.10^{-2}	3.10^{-3}	4.10^{-4}
			MLD	8.10^{-2}	4.10^{-2}	2.10^{-2}	5.10^{-3}	1.10^{-3}	1.10^{-4}	1.10^{-5}	3.10^{-1}	1.10^{-1}	6.10^{-2}	2.10^{-2}	4.10^{-3}	6.10^{-4}	6.10^{-5}
		AE BP (sys.)	BP - 5 iter.	9.10^{-2}	5.10^{-2}	3.10^{-2}	1.10^{-2}	3.10^{-3}	8.10^{-4}	1.10^{-4}	4.10^{-1}	2.10^{-1}	1.10^{-1}	6.10^{-2}	2.10^{-2}	5.10^{-3}	9.10^{-4}
			GNBP - 5 iter.	9.10^{-2}	5.10^{-2}	3.10^{-2}	1.10^{-2}	3.10^{-3}	6.10^{-4}	9.10^{-5}	4.10^{-1}	2.10^{-1}	1.10^{-1}	6.10^{-2}	2.10^{-2}	4.10^{-3}	7.10^{-4}
			MLD	8.10^{-2}	4.10^{-2}	2.10^{-2}	5.10^{-3}	1.10^{-3}	3.10^{-4}	4.10^{-5}	3.10^{-1}	2.10^{-1}	8.10^{-2}	3.10^{-2}	8.10^{-3}	2.10^{-3}	3.10^{-4}
		BCH (non sys.)	BP - 5 iter.	2.10^{-1}	1.10^{-1}	8.10^{-2}	4.10^{-2}	2.10^{-2}	6.10^{-3}	2.10^{-3}	8.10^{-1}	6.10^{-1}	4.10^{-1}	2.10^{-1}	1.10^{-1}	3.10^{-2}	1.10^{-2}
			GNBP - 5 iter.	2.10^{-1}	1.10^{-1}	8.10^{-2}	4.10^{-2}	2.10^{-2}	5.10^{-3}	7.10^{-4}	8.10^{-1}	6.10^{-1}	4.10^{-1}	2.10^{-1}	1.10^{-1}	3.10^{-2}	5.10^{-3}
		BCH (sys.)	BP - 5 iter.	2.10^{-1}	1.10^{-1}	8.10^{-2}	7.10^{-2}	4.10^{-2}	2.10^{-2}	1.10^{-2}	8.10^{-1}	6.10^{-1}	4.10^{-1}	3.10^{-1}	2.10^{-1}	8.10^{-2}	3.10^{-2}
			GNBP - 5 iter.	2.10^{-1}	1.10^{-1}	8.10^{-2}	5.10^{-2}	3.10^{-2}	9.10^{-3}	3.10^{-3}	8.10^{-1}	6.10^{-1}	4.10^{-1}	2.10^{-1}	1.10^{-1}	4.10^{-2}	1.10^{-2}
		BCH (all forms)	MLD	8.10^{-2}	4.10^{-2}	1.10^{-2}	3.10^{-3}	4.10^{-4}	3.10^{-5}	2.10^{-6}	2.10^{-1}	1.10^{-1}	4.10^{-2}	9.10^{-3}	1.10^{-3}	8.10^{-5}	4.10^{-6}
	16	AE (sys.)	BP - 5 iter.	8.10^{-2}	5.10^{-2}	2.10^{-2}	1.10^{-2}	3.10^{-3}	8.10^{-4}	1.10^{-4}	5.10^{-1}	3.10^{-1}	2.10^{-1}	7.10^{-2}	3.10^{-2}	6.10^{-3}	1.10^{-3}
			GNBP - 5 iter.	8.10^{-2}	5.10^{-2}	2.10^{-2}	9.10^{-3}	2.10^{-3}	4.10^{-4}	5.10^{-5}	5.10^{-1}	3.10^{-1}	2.10^{-1}	6.10^{-2}	2.10^{-2}	3.10^{-3}	3.10^{-4}
			MLD	7.10^{-2}	4.10^{-2}	1.10^{-2}	4.10^{-3}	7.10^{-4}	7.10^{-5}	6.10^{-6}	3.10^{-1}	2.10^{-1}	7.10^{-2}	2.10^{-2}	4.10^{-3}	4.10^{-4}	5.10^{-5}
		BCH (non sys.)	BP - 5 iter.	1.10^{-1}	9.10^{-2}	5.10^{-2}	3.10^{-2}	1.10^{-2}	3.10^{-3}	6.10^{-4}	7.10^{-1}	5.10^{-1}	4.10^{-1}	2.10^{-1}	6.10^{-2}	2.10^{-2}	4.10^{-3}
			GNBP - 5 iter.	1.10^{-1}	8.10^{-2}	5.10^{-2}	3.10^{-2}	8.10^{-3}	2.10^{-3}	3.10^{-4}	7.10^{-1}	5.10^{-1}	4.10^{-1}	2.10^{-1}	6.10^{-2}	1.10^{-2}	2.10^{-3}
		BCH (sys.)	BP - 5 iter.	1.10^{-1}	9.10^{-2}	6.10^{-2}	4.10^{-2}	3.10^{-2}	1.10^{-2}	6.10^{-3}	7.10^{-1}	5.10^{-1}	4.10^{-1}	2.10^{-1}	1.10^{-1}	5.10^{-2}	3.10^{-2}
			GNBP - 5 iter.	1.10^{-1}	9.10^{-2}	5.10^{-2}	3.10^{-2}	1.10^{-2}	4.10^{-3}	8.10^{-4}	7.10^{-1}	5.10^{-1}	4.10^{-1}	2.10^{-1}	9.10^{-2}	3.10^{-2}	6.10^{-3}
		BCH (all forms)	MLD	7.10^{-2}	4.10^{-2}	1.10^{-2}	4.10^{-3}	5.10^{-4}	6.10^{-5}	2.10^{-6}	3.10^{-1}	2.10^{-1}	7.10^{-2}	1.10^{-2}	2.10^{-3}	2.10^{-4}	8.10^{-6}
63	36	AE (sys.)	BP - 5 iter.	1.10^{-1}	7.10^{-2}	4.10^{-2}	2.10^{-2}	4.10^{-3}	1.10^{-3}	2.10^{-4}	8.10^{-1}	6.10^{-1}	3.10^{-1}	1.10^{-1}	4.10^{-2}	1.10^{-2}	2.10^{-3}
			GNBP - 5 iter.	1.10^{-1}	8.10^{-2}	4.10^{-2}	2.10^{-2}	3.10^{-3}	3.10^{-4}	3.10^{-5}	8.10^{-1}	7.10^{-1}	3.10^{-1}	1.10^{-1}	3.10^{-2}	4.10^{-3}	4.10^{-4}
		BCH (non sys.)	BP - 5 iter.	1.10^{-1}	1.10^{-1}	8.10^{-2}	4.10^{-2}	2.10^{-2}	1.10^{-2}	3.10^{-3}	1.10^{0}	1.10^{0}	8.10^{-1}	5.10^{-1}	3.10^{-1}	1.10^{-1}	4.10^{-2}
			GNBP - 5 iter.	1.10^{-1}	1.10^{-1}	8.10^{-2}	4.10^{-2}	2.10^{-2}	5.10^{-3}	8.10^{-4}	1.10^{0}	1.10^{0}	8.10^{-1}	5.10^{-1}	3.10^{-1}	8.10^{-2}	1.10^{-2}
			NBP - 5 iter. [101]	1.10^{-1}	1.10^{-1}	8.10^{-2}	4.10^{-2}	2.10^{-2}	5.10^{-3}	8.10^{-4}							
		BCH (cycle red.)	BP - 5 iter.	1.10^{-1}	1.10^{-1}	8.10^{-2}	4.10^{-2}	2.10^{-2}	5.10^{-3}	8.10^{-4}	1.10^{0}	1.10^{0}	8.10^{-1}	5.10^{-1}	2.10^{-1}	6.10^{-2}	1.10^{-2}
			GNBP - 5 iter.	1.10^{-1}	1.10^{-1}	7.10^{-2}	3.10^{-2}	1.10^{-2}	3.10^{-3}	3.10^{-4}	1.10^{0}	1.10^{0}	8.10^{-1}	4.10^{-1}	2.10^{-1}	5.10^{-2}	5.10^{-3}
			NBP - 5 iter. [101]	1.10^{-1}	1.10^{-1}	7.10^{-2}	3.10^{-2}	1.10^{-2}	3.10^{-3}	4.10^{-4}							
			mRRD-RNN [101]				4.10^{-3}	3.10^{-4}	2.10^{-5}								
		BCH (all forms)	MLD (OSD) [101]				1.10^{-3}	3.10^{-5}	3.10^{-7}								
	45	AE (sys.)	BP - 5 iter.	1.10^{-1}	7.10^{-2}	4.10^{-2}	2.10^{-2}	4.10^{-3}	7.10^{-4}	1.10^{-4}	9.10^{-1}	7.10^{-1}	4.10^{-1}	2.10^{-1}	5.10^{-2}	1.10^{-2}	1.10^{-3}
			GNBP - 5 iter.	1.10^{-1}	7.10^{-2}	4.10^{-2}	1.10^{-2}	2.10^{-3}	3.10^{-4}	3.10^{-5}	9.10^{-1}	7.10^{-1}	4.10^{-1}	2.10^{-1}	4.10^{-2}	6.10^{-3}	5.10^{-4}
		BCH (non sys.)	BP - 5 iter. [101]	1.10^{-1}	9.10^{-2}	7.10^{-2}	4.10^{-2}	2.10^{-2}	6.10^{-3}	3.10^{-3}							
			NBP - 5 iter. [101]	1.10^{-1}	9.10^{-2}	6.10^{-2}	3.10^{-2}	1.10^{-2}	3.10^{-3}	5.10^{-4}							
		BCH (cycle red.)	BP - 5 iter.	1.10^{-1}	9.10^{-2}	5.10^{-2}	3.10^{-2}	1.10^{-2}	2.10^{-3}	3.10^{-4}	1.10^{0}	9.10^{-1}	7.10^{-1}	4.10^{-1}	1.10^{-1}	3.10^{-2}	5.10^{-3}
			NBP - 5 iter. [101]	1.10^{-1}	8.10^{-2}	5.10^{-2}	2.10^{-2}	6.10^{-3}	1.10^{-3}	1.10^{-4}							
			Perm-RNN [101]		6.10^{-2}	2.10^{-2}	3.10^{-3}	3.10^{-4}	1.10^{-5}								
		BCH (all forms)	MLD (OSD) [101]		6.10^{-2}	2.10^{-2}	3.10^{-3}	2.10^{-4}	8.10^{-6}								
64	18	AE (sys.)	BP - 5 iter.	1.10^{-1}	8.10^{-2}	4.10^{-2}	2.10^{-2}	1.10^{-3}	3.10^{-3}	2.10^{-3}	6.10^{-1}	5.10^{-1}	2.10^{-1}	1.10^{-1}	5.10^{-2}	2.10^{-2}	1.10^{-2}
			GNBP - 5 iter.	1.10^{-1}	6.10^{-2}	4.10^{-2}	1.10^{-2}	3.10^{-3}	5.10^{-4}	9.10^{-5}	6.10^{-1}	3.10^{-1}	2.10^{-1}	7.10^{-2}	2.10^{-2}	4.10^{-3}	7.10^{-4}
		Irregular LDPC	BP - 5 iter.	1.10^{-1}	8.10^{-2}	4.10^{-2}	1.10^{-2}	3.10^{-3}	3.10^{-4}	3.10^{-5}	6.10^{-1}	4.10^{-1}	3.10^{-1}	1.10^{-1}	3.10^{-2}	4.10^{-3}	3.10^{-4}
			GNBP - 5 iter.	1.10^{-1}	8.10^{-2}	4.10^{-2}	1.10^{-2}	2.10^{-3}	3.10^{-4}	2.10^{-5}	6.10^{-1}	4.10^{-1}	3.10^{-1}	8.10^{-1}	2.10^{-2}	3.10^{-3}	2.10^{-4}
		Regular LDPC	BP - 5 iter.				6.10^{-2}	3.10^{-2}	8.10^{-3}	1.10^{-3}							
	32	AE (sys.)	BP - 5 iter.	1.10^{-1}	9.10^{-2}	4.10^{-2}	2.10^{-2}	6.10^{-3}	2.10^{-3}	4.10^{-4}	8.10^{-1}	6.10^{-1}	3.10^{-1}	2.10^{-1}	5.10^{-2}	1.10^{-2}	4.10^{-3}
			GNBP - 5 iter.	1.10^{-1}	7.10^{-2}	4.10^{-2}	1.10^{-2}	3.10^{-3}	4.10^{-4}	3.10^{-5}	8.10^{-1}	6.10^{-1}	3.10^{-1}	1.10^{-1}	3.10^{-2}	4.10^{-3}	5.10^{-4}
		Irregular LDPC	BP - 5 iter.	1.10^{-1}	5.10^{-2}	2.10^{-2}	6.10^{-3}	1.10^{-3}	1.10^{-4}	8.10^{-6}	8.10^{-1}	5.10^{-1}	3.10^{-1}	8.10^{-2}	2.10^{-2}	2.10^{-3}	2.10^{-4}
			GNBP - 5 iter.	1.10^{-1}	5.10^{-2}	2.10^{-2}	6.10^{-3}	1.10^{-3}	1.10^{-4}	8.10^{-6}	8.10^{-1}	5.10^{-1}	3.10^{-1}	8.10^{-2}	2.10^{-2}	2.10^{-3}	2.10^{-4}
		Regular LDPC	BP - 5 iter.		1.10^{-1}	6.10^{-2}	2.10^{-2}	5.10^{-3}	1.10^{-3}	5.10^{-5}							
	46	AE (sys.)	BP - 5 iter.	9.10^{-2}	7.10^{-2}	4.10^{-2}	1.10^{-2}	3.10^{-3}	7.10^{-4}	6.10^{-5}	9.10^{-1}	8.10^{-1}	5.10^{-1}	2.10^{-1}	6.10^{-2}	1.10^{-2}	1.10^{-3}
			GNBP - 5 iter.	9.10^{-2}	7.10^{-2}	4.10^{-2}	1.10^{-2}	2.10^{-3}	3.10^{-4}	3.10^{-5}	9.10^{-1}	8.10^{-1}	5.10^{-1}	2.10^{-1}	5.10^{-2}	7.10^{-3}	8.10^{-4}
		Irregular LDPC	BP - 5 iter.	9.10^{-2}	7.10^{-2}	4.10^{-2}	1.10^{-2}	2.10^{-3}	3.10^{-4}	3.10^{-5}	9.10^{-1}	8.10^{-1}	5.10^{-1}	2.10^{-1}	5.10^{-2}	7.10^{-3}	5.10^{-4}
			GNBP - 5 iter.	9.10^{-2}	7.10^{-2}	4.10^{-2}	1.10^{-2}	2.10^{-3}	2.10^{-4}	1.10^{-5}	9.10^{-1}	8.10^{-1}	5.10^{-1}	2.10^{-1}	3.10^{-2}	3.10^{-3}	2.10^{-4}
		Regular LDPC	BP - 5 iter.	1.10^{-1}	8.10^{-2}	5.10^{-2}	2.10^{-2}	6.10^{-3}	1.10^{-3}	1.10^{-4}							

BIBLIOGRAPHY

[1] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] "Matrix Calculus." `https://en.wikipedia.org/wiki/Matrix_calculus`. Accessed: 2022-10-17.

[3] "Hexa-X." `https://hexa-x.eu/`. Accessed: 2022-08-29.

[4] N. Varsier, L.-A. Dufrène, M. Dumay, Q. Lampin, and J. Schwoerer, "A 5G New Radio for Balanced and Mixed IoT Use Cases: Challenges and Key Enablers in FR1 Band," *IEEE Communications Magazine*, vol. 59, no. 4, pp. 82–87, 2021.

[5] S. Ali, W. Saad, N. Rajatheva, K. Chang, D. Steinbach, B. Sliwa, C. Wietfeld, K. Mei, H. Shiri, H. Zepernick, T. M. C. Chu, I. Ahmad, J. Huusko, J. Suutala, S. Bhadauria, V. Bhatia, R. Mitra, S. Amuru, R. Abbas, B. Shao, M. Capobianco, G. Yu, M. Claes, T. Karvonen, M. Chen, M. Girnyk, and H. Malik, "6G White Paper on Machine Learning in Wireless Communication Networks," *CoRR*, vol. abs/2004.13875, 2020.

[6] C. F. Miltiadis *et al.*, "Pervasive Artificial Intelligence in Next Generation Wireless: The Hexa-X Project Perspective," in *Proceedings of the First International Workshop on Artificial Intelligence in Beyond 5G and 6G Wireless Networks*, vol. 3189, 2022.

[7] A. Askri and G. R.-B. Othman, "DNN Assisted Sphere Decoder," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1172–1176, 2019.

[8] O. Shental and J. Hoydis, ""Machine LLRning": Learning to Softly Demodulate," in *2019 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–7, 2019.

[9] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of Machine Learning Accelerators," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–12, 2020.

[10] "Edge TPU - Performance Benchmark." `https://coral.withgoogle.com/docs/edgetpu/benchmarks/`. Accessed: 2022-08-25.

[11] J. Hoydis, F. A. Aoudia, A. Valcarce, and H. Viswanathan, "Toward a 6G AI-Native Air Interface," *IEEE Communications Magazine*, vol. 59, no. 5, pp. 76–81, 2021.

[12] G. Larue, M. Dhiflaoui, L.-A. Dufrene, Q. Lampin, P. Chollet, H. Ghauch, and G. Rekaya, "Low-Complexity Neural Networks for Baseband Signal Processing," in *2020 IEEE Globecom Workshops*, pp. 1–6, 2020.

[13] G. Larue, L.-A. Dufrene, Q. Lampin, P. Chollet, H. Ghauch, and G. Rekaya, "Blind Neural Belief Propagation Decoder for Linear Block Codes," in *2021 Joint European Conference on Networks and Communications & 6G Summit*, pp. 106–111, 2021.

[14] G. Larue, L.-A. Dufrene, Q. Lampin, H. Ghauch, and G. Rekaya, "Neural Belief Propagation Auto-Encoder for Linear Block Code Design," *IEEE Transaction on Communications*, 2022.

[15] L.-A. Dufrene, Q. Lampin, and G. Larue, "Procédés et dispositifs pour geler ou adapter les paramètres d'un réseau de neurones utilisé dans un réseau de télécommunications," 2020. WO2022144520A1 - FR3118519A1.

[16] Q. Lampin, G. Larue, and L.-A. Dufrene, "Procédé et système d'adaptation d'un réseau de neurones utilisé dans un réseau de télécommunications," 2020. WO2021255362A1 - FR3111454A1.

[17] "Hexa-X Deliverable D4.1 - AI-Driven Communication & Computation Co-Design: Gap Analysis and Blueprint." `https://hexa-x.eu/wp-content/uploads/2021/09/Hexa-X-D4.1_v1.0.pdf`. Accessed: 2022-08-29.

[18] "Deliverable D4.2 - AI-Driven Communication & Computation Co-Design: Initial Solutions." `https://hexa-x.eu/wp-content/uploads/2022/07/Hexa-X_D4.2_v1.0.pdf`, 2022. Accessed: 2022-10-24.

[19] "Antikythera Mechanism." `https://en.wikipedia.org/wiki/Antikythera_mechanism`. Accessed: 2022-08-31.

[20] "Euclidean Algorithm." `https://en.wikipedia.org/wiki/Euclidean_algorithm`. Accessed: 2022-08-31.

[21] "Timeline of Algorithms." `https://en.wikipedia.org/wiki/Timeline_of_algorithms`. Accessed: 2022-08-31.

[22] "Charles Babbage." `https://en.wikipedia.org/wiki/Charles_Babbage`. Accessed: 2022-08-31.

[23] "Ada Lovelace." `https://en.wikipedia.org/wiki/Ada_Lovelace`. Accessed: 2022-08-31.

[24] "Alan Turing." `https://en.wikipedia.org/wiki/Alan_Turing`. Accessed: 2022-08-31.

[25] "Lovelace, Turing and the invention of computers." `https://www.sciencemuseum.org.uk/objects-and-stories/lovelace-turing-and-invention-computers`. Accessed: 2022-08-31.

[26] "Jacquard Machine." `https://en.wikipedia.org/wiki/Jacquard_machine`. Accessed: 2022-08-31.

[27] "Analytical Engine." `https://en.wikipedia.org/wiki/Analytical_Engine`. Accessed: 2022-08-31.

[28] "Turing Machine." `https://en.wikipedia.org/wiki/Turing_machine`. Accessed: 2022-08-31.

[29] "Turing Test." `https://en.wikipedia.org/wiki/Turing_test`. Accessed: 2022-08-31.

[30] "ENIAC." `https://en.wikipedia.org/wiki/ENIAC`. Accessed: 2022-08-31.

[31] "Transistor." `https://en.wikipedia.org/wiki/Transistor`. Accessed: 2022-08-31.

[32] D. Poole, A. Mackworth, and R. Goebel, *Computational Intelligence: A Logical Approach*. USA: Oxford University Press, Inc., 1997.

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014.

[34] "Web Site - ImageNet Large Scale Visual Recognition Challenge (ILSVRC)." http://www.image-net.org/challenges/LSVRC/. Accessed: 2022-10-05.

[35] D. Silver *et al.*, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, pp. 484–489, jan 2016.

[36] T. Brown *et al.*, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[37] A. Turing, "Computing Machinery and Intelligence," *Mind*, vol. LIX, pp. 433–460, 10 1950.

[38] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[39] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate any Function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993.

[40] H. Siegelmann and E. Sontag, "On the Computational Power of Neural Nets," *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 132–150, 1995.

[41] W. Mcculloch and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[42] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[43] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[44] K. Cho *et al.*, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[45] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning," *CoRR*, vol. abs/1811.03378, 2018.

[46] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*, pp. 9–48. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[47] S. Basodi, C. Ji, H. Zhang, and Y. Pan, "Gradient Amplification: An Efficient Way to Train Deep Neural Networks," *Big Data Mining and Analytics*, vol. 3, no. 3, pp. 196–207, 2020.

[48] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," 2018.

[49] L. Lu, "Dying ReLU and Initialization: Theory and Numerical Examples," *Communications in Computational Physics*, vol. 28, pp. 1671–1706, jun 2020.

[50] P. Billingsley, *Probability and Measure*. John Wiley and Sons, second ed., 1986.

[51] F. Dekking, C. Kraaikamp, H. Lopuhaä, and L. Meester, *A Modern Introduction to Probability and Statistics: Understanding Why and How*. Springer Texts in Statistics, Springer, 2005.

[52] "Probability and Random Variables." https://www.ewi-psy.fu-berlin.de/einrichtungen/arbeitsbereiche/computational_cogni_neurosc/teaching/The_General_Linear_Model_20_211/5_Probability_and_random_variables.pdf. Accessed: 2022-08-29.

[53] E. Ising, "Beitrag zur Theorie des Ferromagnetismus," *Zeitschrift für Physik*, vol. 31, pp. 253–258, 1925.

[54] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[55] L. E. Baum and T. Petrie, "Statistical Inference for Probabilistic Functions of Finite State Markov Chains," *Annals of Mathematical Statistics*, vol. 37, pp. 1554–1563, 1966.

[56] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inf. Theory*, vol. 8, pp. 21–28, 1962.

[57] "Edge TPU - Run Inference at the Edge." `https://cloud.google.com/edge-tpu`. Accessed: 2022-08-25.

[58] "Intel Neural Compute Stick 2." `https://software.intel.com/en-us/neural-compute-stick`. Accessed: 2022-08-25.

[59] "Jetson Nano Brings AI Computing to Everyone." `https://devblogs.nvidia.com/jetson-nano-ai-computing/`. Accessed: 2022-08-25.

[60] "Qualcomm AI is Transforming Industries by Accelerating Global Commercialization of AI." `https://www.qualcomm.com/products/application/smartphones/mobile-ai`. Accessed: 2022-08-25.

[61] "Apple Unleashes M1." `https://www.apple.com/fr/newsroom/2020/11/apple-unleashes-m1/`. Accessed: 2022-08-25.

[62] "Kirin 9000 Is Huawei's First 5nm Chip." `https://www.linleygroup.com/newsletters/newsletter_detail.php?num=6247&year=2020&tag=3`. Accessed: 2022-08-25.

[63] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A Survey of Neuromorphic Computing and Neural Networks in Hardware," *ArXiv*, vol. abs/1705.06963, 2017.

[64] J. Grollier, D. Querlioz, and M. D. Stiles, "Spintronic Nanodevices for Bioinspired Computing," *Proceedings of the IEEE*, vol. 104, no. 10, pp. 2024–2039, 2016.

[65] B. Rajendran, A. Sebastian, M. Schmuker, N. Srinivasa, and E. Eleftheriou, "Low-Power Neuromorphic Hardware for Signal Processing Applications: A Review of Architectural and System-Level Design Approaches," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 97–110, 2019.

[66] T. Kohonen, "The Self-Organizing Map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[67] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[68] I. P. Pavlov and G. V. Anrep, *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*. Oxford University Press: Humphrey Milford, 1927.

[69] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[70] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-Time Analysis of the Multiarmed Bandit Problem," *Mach. Learn.*, vol. 47, p. 235–256, may 2002.

[71] G. Rummery and M. Niranjan, "On-Line Q-Learning Using Connectionist Systems," TR 166, CUED/F-INFENG, 1994.

[72] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization.," in *ICLR (Poster)* (Y. Bengio and Y. LeCun, eds.), 2015.

[73] T. Dozat, "Incorporating Nesterov Momentum into Adam," in *ICLR (Workshop)*, 2016.

[74] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[75] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the Institute of Radio Engineers*, vol. 40, pp. 1098–1101, September 1952.

[76] A. Ahmed, A. Al-Dweik, Y. Iraqi, H. Mukhtar, M. Naeem, and E. Hossain, "Hybrid Automatic Repeat Request (HARQ) in Wireless Communications Systems and Standards: A Contemporary Survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2711–2752, 2021.

[77] C. Shannon, "Communication in the Presence of Noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.

[78] O. Afisiadis, A. Burg, and A. Balatsoukas-Stimming, "Coded LoRa Frame Error Rate Analysis," 2019.

[79] J. G. Proakis, *Digital Communications 5th Edition*. McGraw Hill, 2007.

[80] R. Frank, "Polyphase Codes with Good Non-Periodic Correlation Properties," *IEEE Transactions on Information Theory*, vol. 9, no. 1, pp. 43–45, 1963.

[81] D. Chu, "Polyphase Codes with Good Periodic Correlation Properties (Corresp.)," *IEEE Transactions on Information Theory*, vol. 18, no. 4, pp. 531–532, 1972.

[82] B. Widrow, "Thinking About Thinking: the Discovery of the LMS Algorithm," *IEEE Signal Processing Magazine*, vol. 22, no. 1, pp. 100–106, 2005.

[83] B. Widrow and M. E. Hoff, "Adaptive Switching Circuits," in *1960 IRE WESCON Convention Record, Part 4*, (New York), pp. 96–104, IRE, 1960.

[84] R. W. Lucky, "Automatic Equalization for Digital Communication," *The Bell System Technical Journal*, vol. 44, no. 4, pp. 547–588, 1965.

[85] R. Lucky, "The Adaptive Equalizer," *IEEE Signal Processing Magazine*, vol. 23, no. 3, pp. 104–107, 2006.

[86] M. M. Sondhi and A. J. Presti, "A Self-Adaptive Echo Canceller," *The Bell System Technical Journal*, vol. 45, no. 10, pp. 1851–1854, 1966.

[87] "ImageNet." `https://fr.wikipedia.org/wiki/ImageNet`. Accessed: 2022-08-25.

[88] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[89] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving Deep Neural Networks for LVCSR Using Rectified Linear Units and Dropout," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8609–8613, 2013.

[90] "Tensorflow." `https://www.tensorflow.org/`. Accessed: 2022-08-29.

[91] "PyTorch." `https://pytorch.org/`. Accessed: 2022-08-29.

[92] 3GPP, "NR; Physical Channels and Modulation," TS 38.211, 3GPP, 2017.

[93] 3GPP, "NR; Multiplexing and Channel Coding," TS 38.212, 3GPP, 2017.

[94] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional Radio Modulation Recognition Networks," in *Engineering Applications of Neural Networks* (C. Jayne and L. Iliadis, eds.), (Cham), pp. 213–226, Springer International Publishing, 2016.

[95] N. E. West and T. O'Shea, "Deep Architectures for Modulation Recognition," in *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–6, 2017.

[96] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-Air Deep Learning Based Radio Signal Classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[97] H. Ye, G. Y. Li, and B.-H. Juang, "Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems," *IEEE Wireless Communications Letters*, vol. 7, no. 1, pp. 114–117, 2018.

[98] X. Wang, L. Gao, and S. Mao, "CSI Phase Fingerprinting for Indoor Localization With a Deep Learning Approach," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1113–1123, 2016.

[99] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication Algorithms via Deep Learning," 2018.

[100] Y. Wang, Z. Zhang, S. Zhang, S. Cao, and S. Xu, "A Unified Deep Learning Based Polar-LDPC Decoder for 5G Communication Systems," in *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6, 2018.

[101] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep Learning Methods for Improved Decoding of Linear Codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018.

[102] A. Buchberger, C. Häger, H. Pfister, L. Schmalen, and A. Graell i Amat, "Pruning Neural Belief Propagation Decoders," in *Proc. IEEE International Symposium on Information Theory*, pp. 338–342, 2020.

[103] L. Lugosch and W. J. Gross, "Neural Offset Min-Sum Decoding," in *Proc. IEEE International Symposium on Information Theory*, pp. 1361–1365, 2017.

[104] E. Nachmani, Y. Bachar, E. Marciano, D. Burshtein, and Y. Be'ery, "Near Maximum Likelihood Decoding with Deep Learning," *arXiv, 1801.02726*, 2018.

[105] I. Be'Ery, N. Raviv, T. Raviv, and Y. Be'Ery, "Active Deep Decoding of Linear Codes," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 728–736, 2020.

[106] N. Raviv, A. Caciularu, T. Raviv, J. Goldberger, and Y. Be'ery, "Perm2Vec: Graph Permutation Selection for Decoding of Error Correction Codes using Self-Attention," *IEEE Transactions on Communications*, vol. 39, no. 1, pp. 79–88, 2021.

[107] M. Ebada, S. Cammerer, A. Elkelesh, and S. ten Brink, "Deep Learning-Based Polar Code Design," in *Proc. Allerton Conference on Communication, Control, and Computing*, pp. 177–183, 2019.

[108] A. Elkelesh, M. Ebada, S. Cammerer, L. Schmalen, and S. ten Brink, "Decoder-in-the-Loop: Genetic Optimization-Based LDPC Code Design," *IEEE Access*, vol. 7, pp. 141161–141170, 2019.

[109] S. Cammerer, T. Gruber, J. Hoydis, and S. ten Brink, "Scaling Deep Learning-Based Decoding of Polar Codes via Partitioning," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–6, 2017.

[110] M. Kim, N.-I. Kim, W. Lee, and D.-H. Cho, "Deep Learning-Aided SCMA," *IEEE Communications Letters*, vol. 22, no. 4, pp. 720–723, 2018.

[111] S. Kannan, H. Kim, and S. Oh, "Deep learning and information theory: An emerging interface." Accessed: 2022-08-25, 2018.

[112] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.

[113] S. Dörner, S. Cammerer, J. Hoydis, and S. t. Brink, "Deep Learning Based Communication Over the Air," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 132–143, 2018.

[114] T. J. O'Shea, T. Roy, N. West, and B. C. Hilburn, "Physical Layer Communications System Design Over-the-Air Using Adversarial Networks," in *2018 26th European Signal Processing Conference (EUSIPCO)*, pp. 529–532, 2018.

[115] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[116] P. Baldi, "Autoencoders, Unsupervised Learning, and Deep Architectures," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning* (I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, eds.), vol. 27 of *Proceedings of Machine Learning Research*, (Bellevue, Washington, USA), pp. 37–49, PMLR, 02 Jul 2012.

[117] Z. Qin, X. Tao, J. Lu, W. Tong, and G. Y. Li, "Semantic Communications: Principles and Challenges," 2022.

[118] H. Xie, Z. Qin, X. Tao, and K. B. Letaief, "Task-Oriented Multi-User Semantic Communications," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 9, pp. 2584–2597, 2022.

[119] M. Sana and E. C. Strinati, "Learning Semantics: An Opportunity for Effective 6G Communications," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, pp. 631–636, 2022.

[120] M. Kalfa, M. Gok, A. Atalik, B. Tegin, T. M. Duman, and O. Arikan, "Towards Goal-Oriented Semantic Signal Processing: Applications and Future Challenges," *Digital Signal Processing*, vol. 119, p. 103134, 2021.

[121] H. Seo, J. Park, M. Bennis, and M. Debbah, "Semantics-Native Communication with Contextual Reasoning," *CoRR*, vol. abs/2108.05681, 2021.

[122] E. Bourtsoulatze, D. B. Kurka, and D. Gündüz, "Deep Joint Source-channel Coding for Wireless Image Transmission," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4774–4778, 2019.

[123] M. Jankowski, D. Gündüz, and K. Mikolajczyk, "Wireless Image Retrieval at the Edge," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 89–100, 2021.

[124] M. Ebada, S. Cammerer, A. Elkelesh, and S. ten Brink, "Deep Learning-Based Polar Code Design," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 177–183, 2019.

[125] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Turbo Autoencoder: Deep Learning Based Channel Codes for Point-to-Point Communication Channels," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, (Red Hook, NY, USA), Curran Associates Inc., 2019.

[126] M. Honkala, D. Korpi, and J. M. J. Huttunen, "Deeprx: Fully convolutional deep learning receiver," 2020.

[127] 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.

[128] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, (New York, NY, USA), p. 1–12, Association for Computing Machinery, 2017.

[129] L. Sun, D. Ke, X. Wang, Z. Huang, and K. Huang, "Robustness of Deep Learning-Based Specific Emitter Identification under Adversarial Attacks," *Remote Sensing*, vol. 14, no. 19, 2022.

[130] 3GPP/ETSI, "3GPP TR 25.943 V4.0.0 - ETSI TR 125 943 V4.0.0," 2001.

[131] N. Levinson, "The Wiener (Root Mean Square) Error Criterion in Filter Design and Prediction," *Journal of Mathematics and Physics*, vol. 25, no. 1-4, pp. 261–278, 1946.

[132] R. G. Brown and P. Y. C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering: with MATLAB Exercises and Solutions; 3rd ed.* New York, NY: Wiley, 1997.

[133] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object Recognition with Gradient-Based Learning," in *Shape, Contour and Grouping in Computer Vision*, (Berlin, Heidelberg), p. 319, Springer-Verlag, 1999.

[134] K. Ohnishi and K. Nakayama, "A Neural Demodulator for Quadrature Amplitude Modulation Signals," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 4, pp. 1933–1938 vol.4, 1996.

[135] E. Ceballos, *A Novel Adaptive Multilevel - Quadrature Amplitude Modulation (M-QAM) Receiver Using Machine Learning to Mitigate Multipath Fading Channel Effects*. Master's thesis collection, Department of Electrical Engineering, California State University, Long Beach, 2018.

[136] H.-G. Yeh and H. Seo, "Low Complexity Demodulator for M-ary QAM," in *2007 Wireless Telecommunications Symposium*, pp. 1–6, 2007.

[137] "Product page - USRP B210 Software Defined Radio." `https://www.ettus.com/all-products/ub210-kit/`. Accessed: 2022-09-20.

[138] "Product page - Spirent Vertex Channel Emulator." `https://www.spirent.com/products/radio-frequency-and-wi-fi-channel-emulation-vertex`. Accessed: 2022-09-20.

[139] R. Y. Yen, "Unbiased MMSE vs. Biased MMSE Equalizers," *Journal of Applied Science and Engineering*, vol. 12, pp. 45–56, mars 2009.

[140] 3GPP, "TSG RAN1 86 and 87 Meetings - Finale Minutes Reports," 2016.

[141] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced Complexity Iterative Decoding of Low-Density Parity Check Codes Based on Belief Propagation," *IEEE Transactions on Communications*, vol. 47, pp. 673 – 680, 1999.

[142] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-Complexity Decoding of LDPC Codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[143] F. Miltiadis *et al.*, "Pervasive Artificial Intelligence in Next Generation Wireless: The Hexa-X Project Perspective," in *Proc. International Workshop on Artificial Intelligence in Beyond 5G and 6G Wireless Networks*, 2022.

[144] M. Lian, F. Carpi, C. Häger, and H. D. Pfister, "Learned Belief-Propagation Decoding with Simple Scaling and Snr Adaptation," in *Proc. IEEE International Symposium on Information Theory*, pp. 161–165, 2019.

[145] T. Gruber, S. Cammerer, J. Hoydis, and S. t. Brink, "On Deep Learning-Based Channel Decoding," in *Proc. Conference on Information Sciences and Systems*, pp. 1–6, 2017.

[146] A. Makkuva, X. Liu, M. V. Jamali, H. Mahdavifar, S. Oh, and P. Viswanath, "KO Codes: Inventing Nonlinear Encoding and Decoding for Reliable Wireless Communication via Deep-learning," *arXiv, 2108.12920*, 2021.

[147] H. Ye, L. Liang, and G. Y. Li, "Circular Convolutional Auto-Encoder for Channel Coding," in *Proc. IEEE International Workshop on Signal Processing Advances in Wireless Communications*, pp. 1–5, 2019.

[148] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "LEARN Codes: Inventing Low-Latency Codes via Recurrent Neural Networks," in *Proc. IEEE International Conference on Communications*, pp. 1–7, 2019.

[149] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Turbo Autoencoder: Deep Learning Based Channel Codes for Point-to-Point Communication Channels," in *Proc. IEEE Conference on Neural Information Processing Systems*, p. 11, 2019.

[150] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2007.

[151] L. Lan, Y. Tai, L. Chen, S. Lin, and K. Abdel-Ghaffar, "A Trellis-Based Method for Removing Cycles from Bipartite Graphs and Construction of Low Density Parity Check Codes," *IEEE Communications Letters*, vol. 8, no. 7, pp. 443–445, 2004.

[152] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, pp. 555–584. MIT Press, 2016. `http://www.deeplearningbook.org`.

[153] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and Software Optimizations for Accelerating Deep Neural Networks: Survey of Current Trends, Challenges, and the Road Ahead," *IEEE Access*, vol. 8, pp. 225134–225180, 2020.

[154] E. Nachmani and L. Wolf, "Hyper-Graph-Network Decoders for Block Codes," in *Advances in Neural Information Processing Systems*, pp. 2326–2336, 2019.

[155] A. Buchberger, C. Häger, H. Pfister, L. Schmalen, and A. Graell i Amat, "Learned Decimation for Neural Belief Propagation Decoders," *arXiv, 2011.02161*, 2020.

[156] A. Gu, C. Gulcehre, T. Paine, M. Hoffman, and R. Pascanu, "Improving the Gating Mechanism of Recurrent Neural Networks," in *Proceedings of Machine Learning Research* (H. D. III and A. Singh, eds.), vol. 119, pp. 3800–3809, 2020.

[157] E. Kavvousanos and V. Paliouras, "An Iterative Approach to Syndrome-based Deep Learning Decoding," *IEEE GlobeCom2020 - Machine Learning in Communications Workshop*, 2020.

[158] M. Helmling *et al.*, "Database of Channel Codes and ML Simulation Results," 2019.

[159] G. Hinton, N. Srivastava, and K. Swersky, "Neural Networks for Machine Learning - Lecture 6."

[160] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks," *arXiv, 1506.01186*, 2015.

[161] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9 of *Proceedings of Machine Learning Research*, pp. 249–256, 2010.

[162] E. Jang, S. Gu, and B. Poole, "Categorical Reparameterization with Gumbel-Softmax," in *Proc. International Conference on Learning Representations*, 2017.

[163] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, no. 3–4, p. 229–256, 1992.

[164] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation.," *arXiv, 1308.3432*, 2013.

[165] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "BinaryNet: Training Deep Neural Networks with Weights and Activations constrained to +1 or -1," *arXiv, 1602.02830*, 2016.

[166] J. Ramapuram and R. Webb, "Improving Discrete Latent Representations with Differentiable Approximation Bridges," in *Proc. IEEE International Joint Conference on Neural Networks*, pp. 1–10, 2020.

[167] A. Agresti and B. Coull, "Approximate is Better than "Exact" for Interval Estimation of Binomial Proportions," *The American Statistician*, vol. 52, no. 2, pp. 119–126, 1998.

[168] M. Fossorier and S. Lin, "Soft-Decision Decoding of Linear Block Codes Based on Ordered Statistics," *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, 1995.

[169] H. Xiao-Yu, E. Eleftheriou, and D. Arnold, "Regular and Irregular Progressive Edge-Growth Tanner Graphs," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 386–398, 2005.

[170] G. Liva, L. Gaudio, and T. Ninacs, "Code Design for Short Blocks: A Survey," in *Proc. IEEE European Conference on Networks Communications*, (Athens, Greece), 2016.

[171] I. Dimnik and Y. Be'ery, "Improved Random Redundant Iterative HDPC Decoding," *IEEE Transactions on Communications*, vol. 57, no. 7, pp. 1982–1985, 2009.

ECOLE DOCTORALE

Titre: Modèles IA pour le traitement des signaux numériques dans les futurs réseaux 6G-IoT

Mots clés: 6G, Internet des objets, couche physique, codage de canal, réseaux de neurones, apprentissage machine

Résumé: Les technologies sans fil sont d'une importance capitale pour les sociétés d'aujourd'hui et les futurs réseaux de communication de 6ème génération sont appelés à relever nombre de défis sociétaux et technologiques. Si les infrastructures de communication ont un impact environnemental croissant qu'il est essentiel de réduire, les technologies numériques ont également un rôle à jouer dans la réduction de l'impact de tous les secteurs de l'économie. À cette fin, les réseaux du futurs devront non seulement permettre un transfert d'informations plus efficace, mais aussi répondre aux besoins croissants de capacité d'échange de données. C'est notamment le rôle des cas d'utilisation de l'internet des objets, où un nombre massif de capteurs permet de superviser des systèmes complexes. Ces cas d'utilisation sont associés à de nombreuses contraintes telles que des ressources énergétiques et une complexité limitées. Par conséquent, une couche physique - chargée de la transmission de l'information entre les noeuds du réseau - efficace et peu complexe est absolument cruciale. Dans cette optique, l'utilisation de techniques d'intelligence artificielle est pertinente. D'une part, le cadre mathématique des réseaux neuronaux permet des implémentations matérielles génériques efficaces et peu coûteuses. D'autre part, l'application de procédures d'apprentissage permet d'améliorer les performances de certains algorithmes. Dans ce travail, nous nous intéressons à l'utilisation des réseaux de neurones et de l'apprentissage automatique pour le traitement numérique du signal dans le contexte des réseaux 6G-IoT. En premier lieu, nous nous intéressons à la transcription sous forme de réseaux de neurones de certains algorithmes d'égalisation, de démodulation et de décodage issus de la littérature des communications numériques. Dans un second temps, nous nous intéressons à l'application de mécanismes d'apprentissage sur ces structures de réseaux de neurones afin d'en améliorer les performances. Un décodeur de codes linéaires en bloc est proposé et permet la découverte à l'aveugle d'un schéma de décodage dont les performances sont au moins équivalentes à celles du décodeur de référence. Enfin, une structure de bout en bout est présentée, permettant l'apprentissage conjoint d'un schéma de codage/décodage avec des performances et une complexité comparables aux solutions état de l'art.

Title: AI Models for Digital Signal Processing in Future 6G-IoT Networks

Keywords: 6G, Internet of Things, Physical Layer, Channel Coding, Neural Networks, Machine Learning

Abstract: Wireless technologies are of paramount importance to today's societies and future 6$^{\text{th}}$ generation communication networks are expected to address many societal and technological challenges. While communications infrastructures have a growing environmental impact that needs to be reduced, digital technologies also have a role to play in reducing the impact of all sectors of the economy. To this end, the future networks will not only have to enable more efficient information transfer, but also meet the growing need for data exchange capacity. This is particularly the role of the Internet of Things use cases, where a massive number of sensors allow to monitor complex systems. These use cases are associated with many constraints such as limited energy resources and complexity. Therefore, an efficient and low-complexity physical layer - responsible for the transmission of information between the network nodes - is absolutely crucial. In this regard, the use of artificial intelligence techniques is relevant. On the one hand, the mathematical framework of neural networks allows for efficient and low-cost generic hardware implementations. On the other hand, the application of learning procedures can improve the performance of certain algorithms. In this work, we are interested in the use of neural networks and machine learning for digital signal processing in the context of 6G-IoT networks. First, we are interested in the transcription of certain equalisation, demodulation and decoding algorithms from the digital communications literature into neural networks. Secondly, we are interested in the application of learning mechanisms on these neural network structures in order to improve their performance. A linear block decoder is proposed which allows the blind discovery of a decoding scheme whose performance is at least equivalent to that of the reference decoder. Finally, an end-to-end structure is presented, allowing joint learning of an encoding/decoding scheme with performance and complexity comparable to state-of-the-art solutions.

Institut Polytechnique de Paris
91120 Palaiseau, France