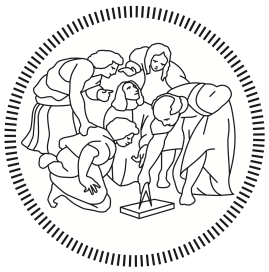
POLITECNICO DI MILANO
School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering
Master of Science in Computer Science and Engineering



Graph Neural Networks and Learned Approximate Message Passing Algorithms for Massive MIMO Detection

Supervisor: Prof. Umberto Spagnolini

Master Thesis by:
Andrea Scotti, 913122

Academic Year 2019-2020

Tengo vicini amici e persone care perché senza radici l'albero cade.
-Marracash

Abstract

Massive multiple-input and multiple-output (MIMO) is a method to improve the performance of wireless communication systems by having a large number of antennas at both the transmitter and the receiver. In the fifth-generation (5G) mobile communication system, Massive MIMO is a key technology to face the increasing number of mobile users and satisfy user demands. At the same time, recovering the transmitted information in a massive MIMO uplink receiver requires more computational complexity when the number of transmitters increases. Indeed, the optimal maximum likelihood (ML) detector has a complexity exponentially increasing with the number of transmitters. Therefore, one of the main challenges in the field is to find the best sub-optimal MIMO detection algorithm according to the performance/complexity trade-off. In this work, all the algorithms are empirically evaluated for large MIMO systems and higher-order modulations.

Firstly, we show how MIMO detection can be represented by a Markov Random Field (MRF) and addressed by the loopy belief propagation (LBP) algorithm to approximately solve the equivalent MAP (maximum a posteriori) inference problem. Then, we propose a novel algorithm (BP-MMSE) that starts from the minimum mean square error (MMSE) solution and updates the prior in each iteration with the LBP belief. To avoid the complexity of computing MMSE, we use Graph Neural Networks (GNNs) to learn a message-passing algorithm that solves the inference task on the same graph.

To further reduce the complexity of message-passing algorithms, we recall how in the large system limit, approximate message passing (AMP), a low complexity iterative algorithm, can be derived from LBP to solve MIMO detection for i.i.d. Gaussian channels. Then, we show numerically how AMP with damping (DAMP) can be robust to low/medium correlation among the channels. To conclude, we propose a low complexity deep neural iterative scheme (Pseudo-MMNet) for solving MIMO detection in the presence of highly correlated channels at the expense of online training for each channel realization. Pseudo-MMNet is based on MMNet algorithm presented in [24] (in turn based on AMP) and it significantly reduces the online training complexity that makes MMNet far from realistic implementations.

Sommario

Massive multiple-input and multiple-output (MIMO) é una tecnica per incrementare l'efficienza della comunicazione wireless con l'ausilio di molteplici antenne a lato trasmettitore e ricevitore. Nella quinta generazione di sistemi di comunicazione mobile (5G), massive MIMO ricopre un ruolo fondamentale per affrontare la continua crescita del numero di dispositivi mobili e per soddisfare le necessitá degli utenti. Allo stesso tempo, recuperare in modo ottimale a lato ricevitore l'informazione trasmessa in un sistema massive MIMO richiede una complessitá computazionale che cresce esponenzialmente con il numero di trasmettitori. Di conseguenza, una delle sfide piú grandi nel campo delle telecomunicazioni é trovare l'algoritmo sub-optimale piú promettente in termini di performance e complessitá computazionale. In questo lavoro, tutti gli algoritmi sono valutati sperimentalmente e in modo empirico in sistemi massive MIMO con alto ordine di modulazione.

In primo luogo, mostriamo come il problema puó essere rappresentato in un campo aleatorio di Markov (MRF) e risolto approssivamente con l'algoritmo loopy belief propagation (LBP) come un problema della stima del massimo a posteriori. Successivamente, proponiamo l'algoritmo BP-MMSE che viene inizializzato con la soluzione del minimo errore medio quadro (MMSE) e iterativamente aggiorna l'informazione a priori con l'algoritmo LBP. Per evitare la complessitá di computare MMSE, usiamo le reti neurali su grafi (GNN) per imparare un algoritmo di scambio di messaggi su grafi per risolvere lo stesso problemo di inferenza.

Per ridurre ulteriormente la complessitá computazionale, da LBP si deriva l'algoritmo approximate message passing (AMP) per canali di trasmissione i.i.d. e Gaussiani. Dimostriamo numericamente che AMP con damping (DAMP) é in grado di performare meglio di MMSE anche per canali di trasmissioni che presentano un livello medio e basso di correlazione. Per concludere, proponiamo un algoritmo iterativo di apprendimento automatico a bassa complessitá (Pseudo-MMNet) per canali di trasmissione che possono essere altamente correlati tra loro, con lo svantaggio di svolgere un nuovo apprendimento per ogni nuova realizzazione del canale. Pseudo-MMNet is fonda su l'algoritmo MMNet proposto in [24] (a sua volta basato su AMP) e riduce significativamente la complessitá computazionale di apprendimento che rende MMNet difficile da considerare per applicazioni reali.

Acknowledgements

The host company of this work is Huawei Technologies Sweden AB. I would like to thank the company for the opportunity to contribute in building a better connected world. During my permanence in Huawei, I benefited significantly from the interaction with my colleagues, especially Nima, Karl and Jinliang. I would like to thank them for their essential support, special advises and precious contribution. Nima also acted as my supervisor inside the company and our discussions helped a lot to shape this thesis and to see hope in the darkness of bad results. Then, I wish to express my appreciation and gratitude to my supervisor Dong from KTH for his endless patience, generous support, and constant guidance. To conclude, I thank my parents, Elisabetta and Roberto, who have supported and encouraged me during my studies.

Contents

Notations

In order to achieve uniformity throughout this thesis and to avoid any possible confusion, the mathematical notations are now presented. Scalars are represented by plain characters, e.g. a. Vectors are represented by a lower-case bold character, e.g. $\mathbf{a}$, and a matrices by an upper-case character in bold, e.g. $\mathbf{A}$. Calligraphic uppercase letters denote sets, e.g. $\mathcal{A}$.

Notation	Definition
$\mathbf{a}^T, \mathbf{A}^T$	Transpose of vector $\mathbf{a}$ and matrix $\mathbf{A}$
$\mathbf{A}^\dagger$	Moore-Penrose Inverse of matrix $\mathbf{A}$.
$\|\| \cdot \|\|_2$	2-norm for vectors and Frobenius norm for matrices
$\mathbf{I}_N$	Identity matrix of size N
$\mathbb{E}_A\{\cdot\}$	Expectation operator with respect to the pdf of random variable A
$\mathbb{VAR}_A\{\cdot\}$	Variance operator with respect to the pdf of random variable A
$\sum_A$	Summation over all the realizations of discrete random variable A
$\arg\max_A$	arg max over all the realizations of discrete random variable A
$\arg\min_A$	arg min over all the realizations of discrete random variable A
$Re(a)$	Real part of a number a
$Im(a)$	Imaginary part of a number a
$\|\mathcal{A}\|$	Cardinality of set $\mathcal{A}$
$\|\mathbf{A}\|$	Determinant of matrix $\mathbf{A}$

Notation	Definition
$\mathbb{R}$	Set of real numbers
$\mathbb{C}$	Set of complex numbers
$\mathcal{A} \cup \mathcal{B}$	Union between sets $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A}/\mathcal{B}$	Set difference of $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A} \times \mathcal{B}$	Cartesian product between sets $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A}^N$	N-ary Cartesian power of set $\mathcal{A}$
a^t	Value of a at iteration t
a_l	l-th entry of vector $\mathbf{a}$
$\mathbf{a}_l$	l-th column of matrix $\mathbf{A}$
$\mathbf{a}_l^r$	l-th row of matrix $\mathbf{A}$
a_{ij}	element in row i and column j of matrix $\mathbf{A}$
$a^{(d)}$	Value of d-th data sample
$\{a^k\}_{k=1}^K$	Set of values $a^1, ..., a^K$
$m_{i \to j}$	Message from i to j
$\approx$	Approximately equal
$\propto$	Equal up to normalization constant
$\sim$	Drawn from probability distribution
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution where μ is the mean and σ^2 the variance
$ne(a)$	Set of nodes a neighbors

Acronyms

4G	Fourth-generation
5G	Fifth-generation
AMP	Approximate Message Passing
ANN	Artificial Neural Network
AWGN	Additive White Gaussian Noise
BP	Belief Propagation
BS	Base Station
DAMP	Damping Approximate Message Passing
DL	Deep Learning
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
GTA	Gaussian Tree Approximation
i.i.d.	Independent and identically distributed
ISTA	Iterative Soft Thresholding Algorithm
KLD	Kullback-Leibler Divergence
LAMP	Learned Approximate Message Passing
LBP	Loopy Belief Propagation
LISTA	Learned Iterative Soft Thresholding Algorithm
LSTM	Long Short-term Memory
LTE	Long-term evolution
MAP	Maximum A Posteriori
MIMO	MIMO Multiple-Input Multiple-Output
ML	Maximum Likelihood
MLP	Multilayer Perceptron
MMSE	Minimum Mean Square Error
MRF	Markov Random Field
NN	Neural Network
PGM	Probabilistic Graphical Model
QAM	Quadrature Amplitude Modulation
RNN	Recurrent Neural Network
SER	Symbol Error Rate
SNR	Signal to Noise Ratio
UE	User equipment
ZF	Zero Forcing

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Multiple-antenna technology, also known as multiple-input multiple-output (MIMO), is one of the most important techniques for advanced wireless communications systems. MIMO technology utilizes multiple antennas at the transmitter and receiver and, therefore, is able to achieve a higher data rate through spatial multiplexing. Long-term evolution (LTE) based fourth generation (4G) mobile communication system allows for up to eight antenna elements at the base station (BS), while the fifth-generation (5G) of cellular communication systems is equipped with a massive number of antennas that simultaneously serves multiple single-antenna user equipments (UEs) on the same time-frequency resource [39]. 5G with Massive MIMO can achieve an order of magnitude higher spectral efficiency (measured in bits/s/Hz) than the LTE legacy standards [35].

Because the transmitted signals come from different antennas, they interfere with each other. This is the main reason why the signal detection problem in MIMO systems is one of the major challenges at the receiver side. The optimal detection method consists of an exhaustive search over all the possible symbols (from the operating constellation) for each transmitter antenna. The computational complexity grows exponentially in the number of transmitting antennas. Unfortunately, such a brute force approach is not realizable for a large number of transmitting antennas. Therefore, researchers proposed several alternative suboptimal solutions that achieve low complexity at the expense of accuracy.

Recently, deep learning (DL) has demonstrated that it can help to offload the prediction complexity on the training phase. The promising results presented in the recently published papers are a perfectly valid motivation to keep investigating in this direction.

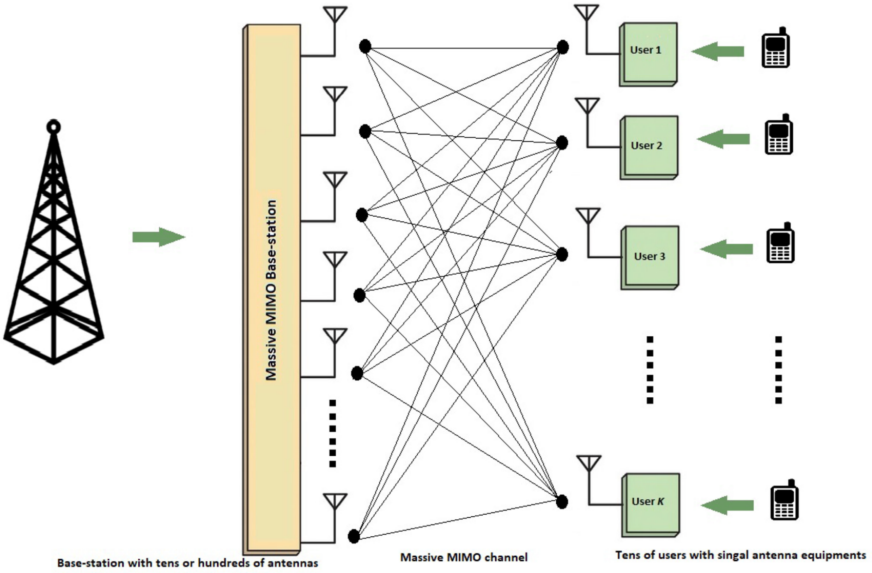


Figure 1.1: Massive MIMO architecture [3].

1.2 Related work

This work mainly focuses on MIMO detection with the aid of graphical models. Detectors are based on the belief propagation (BP) algorithm [40] where the MIMO system is modeled as a fully connected bipartite factor graph [27] and the problem becomes solving an inference task on a graphical model. When graphs contains cycles, the algorithm is also called loopy belief propagation (LBP) and its convergence is not always guaranteed.

In order to solve this issue, the input data and messages can be assumed to be Gaussian distributed, (see, e.g., [6]). In this case, the message update rule is much simpler to compute because each message and posterior probability can be represented by a pair of mean and variance. However, the authors of [42] show that in the case of MIMO detection, where the input data is drawn from a discrete constellation, the algorithm can converge only to the linear minimum mean squared error (MMSE) solution [38] whose performance is known to be inferior to the maximum likelihood (ML) optimal detector when the input is not Gaussian distributed.

Better approaches to reduce the complexity are presented in [17, 16, 42, 29] where the LBP algorithm is run on a equivalent graphical model with a different structure: the pair-wise Markov random field (MRF).

In [17], the fully connected MRF is approximated with a tree on the basis of the Kullback-Leibler divergence (KLD) optimality criterion and then BP is run on the approximated graph. In [26] the message update scheme is modified by adding noise to oscillating messages or by damping the messages as it also done in [33]. Instead, in [14] they construct a dynamic

asynchronous message schedule that differs from the classical single round-robin ordering. Another contribution to the same problem is disclosed by [37], where the authors provide a different approach to compute the message updates by assigning to each edge a probability to appear in the distribution over the spanning trees of the graph that optimizes the underlying variational problem.

Recently, neural networks have been used to solve inference tasks in loopy graphs. The authors of [41] propose a Graph Neural Networks (GNNs) framework to learn a message-passing algorithm that finds inspiration in LBP. GNNs can be seen as an extension of recurrent neural networks that operate on graph-structured inputs [36, 28]. Our work is based on the GNNs variant presented in [41] that, according to [15], is referred to as message-passing neural networks. Each node in GNNs encodes in a hidden state vector the probabilistic information about the corresponding variable in the graphical model. The states are iteratively updated by aggregating the messages coming from the neighbor nodes. Both the messages and the hidden states are outputs of expressive neural networks. To our best knowledge, GNNs have not been applied to MIMO detection problem yet.

In every iteration, the number of exchanged messages in message-passing algorithms is equal to the number of edges in the graphical model. The MIMO MRF is a fully connected graph. Therefore, the complexity of message-passing algorithms becomes prohibitive for large systems if they are not properly parallelized. In order to work with low-complexity iterative algorithms, [23] proposes a new class of algorithms that replace the exchange of messages with matrix multiplications. The authors consider the MIMO factor graph and they simplify the LBP algorithm by assuming a Gaussian distribution for the marginal density functions of the messages. The resulting algorithm is LAMA (short for LArge MIMO Approximate message passing) that is able to achieve the individually-optimal data detector [19] under certain assumptions on the constellation and the structure of the channel matrix: for instance, the channels need to be i.i.d. (independent and identically distributed). LAMA relies on approximate message passing (AMP) which is a class of algorithms that finds an impressive success in the field of compressed sensing [32]. As explained in [32], AMP is used to solve the sparse linear inverse problem that is strictly related to the MIMO detection problem. In the same work, the author shows how to derive AMP from LBP under certain assumptions that are partially used to derive the LAMA algorithm too.

Except for the Onsager correction [32], AMP algorithm has the same iterative scheme of another class of algorithms, well known as iterative soft thresholding algorithm (ISTA). ISTA have been proposed in the literature to solve the sparse linear inverse problem [7]. Unfortunately, it suffers from a slow convergence rate compared to AMP (due to the lack of the Onsager correction), so the authors of [18] proposed a neural network architecture

(LISTA), closely related to ISTA, to speed up the recovery of the signal. In this case, a training dataset is used to train the parameters of the network.

A similar idea is applied in [24] to solve the MIMO detection problem. In this work, the authors propose distinct neural network architectures for different channel models. They show that for i.i.d. Gaussian channels the network (MMNet-iid) is trained offline for multiple channel matrix realizations and the number of parameters to learn is much lower than in the case of spatially correlated channels. Moreover, in the latter case, the network (MMNet) needs to be trained online for every realization of the channel. The online training that they propose introduces a latency that cannot be afforded in realistic implementations.

OAMPNet [20] is another model-driven deep learning network for MIMO detection based on the classical orthogonal AMP (OAMP) algorithm [31]. OAMPNet is supposed to work for a larger class of channel (unitarily-invariant) matrices than AMP. However, we don't consider OAMPNet as a baseline for this work because the authors of [24] show that MMNet outperforms OAMPNet both in performance and complexity.

The achievements of this thesis are mainly based on the work in [24], the experiments in [8], and the analytic tools in [2]. Based on AMP, [8] proposes a novel neural-network architecture, learned AMP (LAMP), that maintains the presence the Onsager correction. The numerical experiments in [8] suggest that LAMP significantly improves upon LISTA in both accuracy and complexity.

To conclude, we consider a new extension of LISTA, termed Ada-LISTA [2], that has been introduced to deal with varying model scenarios. Especially, the authors give an analytic proof of the linear rate convergence of the algorithm.

1.3 Contribution

This thesis addresses the MIMO detection problem by applying the latest advances in supervised deep learning and approximate inference on probabilistic graphical models (PGMs). First of all, we demonstrate how the problem can be represented by a PGM and how to solve it with an iterative approximate inference method such as LBP. We propose a novel algorithm (BP-MMSE) that combines LBP and the MMSE solution to solve the problem for higher-order modulations, e.g. Quadrature Amplitude Modulation-16 (QAM-16) and QAM-64.

Then, we show that a GNN-based approach inspired by BP can solve the same inference task by outperforming the classical LBP, even without the help of the MMSE prior information. We compare the results with the MMSE baseline algorithm under different number of receiver antennas, (signal-to-noise ratio) SNR values, and modulation orders. Furthermore, we

empirically analyze the convergence of the algorithm and propose a strategy to prune the edges in the graph in order to reduce the complexity of exchanging messages between the nodes of the graphical model during each iteration.

Furthermore, we present the AMP algorithm for real-valued MIMO systems and how it is analytically derived from BP under the assumption that the channels are i.i.d. distributed. Then, we show how to slightly modify the AMP algorithm by adding damping after each layer. The resulting scheme, Damping AMP (DAMP), is tested on i.i.d. Gaussian channels and compared with the MMSE and MMNet-iid baselines for different SNR values. Subsequently, we test the robustness of DAMP when it operates with spatially correlated channels.

In the end, we propose a new algorithm, Pseudo-MMNet, that is inspired by MMNet and optimized to work even for strongly spatially correlated channels at the expense of online training for every realization of the channel. We report the similarities between Pseudo-MMNet and Ada-LISTA, whose convergence is proved analytically. To conclude we show that it is possible to control the trade-off between performance and complexity by adjusting the number of network layers or the number of epochs and batch size during training.

1.4 Benefits, ethics and sustainability

This thesis is aligned with the vision for sustainable development outlined in the Sustainable Development Goals [34]. Indeed, we conduct this research with the scope of building a better "connected" world. We believe that bringing connectivity in a efficient way both in crowded cities and remote areas will help:

- to improve the well-being of people and the quality of education;
- to increase the access to job opportunities and unlock smart working where nowadays is still unthinkable;
- to bring innovation in the industry and to improve infrastructures for "smarter" cities.

On one hand, solving MIMO detection for large systems can be an intensive task in terms of computational complexity. Therefore, it might raise concern regarding the sustainability of its direct and indirect effects, e.g. the energy consumption. On the other hand, this work aims to improve the current methods for MIMO detection with a highlighted focus on reducing the computational complexity of the solution by preserving the desired performance. For this reason, we dedicate Section 6.3 to provide an complexity analysis of the algorithms proposed in the rest of the work.

1.5 Outline

The remaining part of this document is organized as follows:

- Chapter 2 describes the state of the art theory and techniques needed to introduce our contribution. We focus on classical and machine learning MIMO detectors, probabilistic graphical models, approximate message passing and iterative soft thresholding algorithms, supervised deep learning and graph neural networks.
- Chapter 3 provides a description of the proposed algorithms: GNNs for MIMO detection, DAMP and Pseudo-MMNet.
- Chapter 4 describes the methodology followed by our approach and the experimental setup.
- Chapter 5 provides an empirical analysis of the algorithms performance compared with the baseline solution.
- Chapter 6 goes over the experiments results and the computational complexity of the proposed algorithms.
- Chapter 7 summarizes our work and proposes possible future improvements.

Chapter 2

Background

2.1 MIMO in wireless communication

One of the main challenges of wireless communication is interference: multiple users have access to the same channel simultaneously. Moreover, user devices move frequently and the surrounding environment can change. So, the channel is shared and time-varying. The work of this thesis focuses on signal detection in the multiuser communication scenario.

Wireless communication requires a system able to coordinate multiple antennas at the receiver unit to detect the signals sent from wireless devices. These devices operate as mobile transmitters within a limited coverage area commonly known as cell. Here, we consider the uplink communication in a cellular system where the base station has the role of central coordinator. The system described above can be modeled by the MIMO system:

$$\tilde{\mathbf{y}} = \tilde{\mathbf{H}}\tilde{\mathbf{x}} + \tilde{\mathbf{n}}, \tag{2.1}$$

where, in the case of narrowband channels, each channel becomes a complex scalar: the channel gain from antenna $T\tilde{x}_j$ and antenna $T\tilde{x}_i$ is defined as $\tilde{h}_{ij} \in \mathbb{C}$. We denote with $\tilde{N}_t$ the number of transmitter antennas and $\tilde{N}_r$ the number of receiver antennas. The measurements across receiver antennas can be stack into vector $\tilde{\mathbf{y}}$ of length $\tilde{N}_r$. Similarly for transmitted signals into vector $\tilde{\mathbf{x}}$ of length $\tilde{N}_t$, and antennas noises into vector $\tilde{\mathbf{n}}$ of length $\tilde{N}_r$. The vectors $\tilde{\mathbf{y}}$, $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{n}}$ are related through the channel matrix $\tilde{\mathbf{H}}$ of dimension $\tilde{N}_r \times \tilde{N}_t$.

Definition 1 (Narrowband MIMO) *The MIMO system is modeled by the linear system*

$$\tilde{\mathbf{y}} = \tilde{\mathbf{H}}\tilde{\mathbf{x}} + \tilde{\mathbf{n}}, \tag{2.2}$$

where $\tilde{\mathbf{H}} \in \mathbb{C}^{\tilde{N}_r} \times \mathbb{C}^{\tilde{N}_t}, \tilde{\mathbf{y}} \in \mathbb{C}^{\tilde{N}_r}, \tilde{\mathbf{n}} \in \mathbb{C}^{\tilde{N}_r}, \tilde{\mathbf{x}} \in \tilde{\mathcal{A}}^{\tilde{N}_t}$ *where* $\tilde{\mathcal{A}} \subset \mathbb{C}$ *is a discrete finite alphabet.*

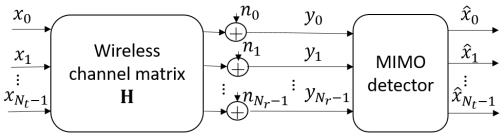


Figure 2.1: Graphical representation of MIMO detection.

In the model defined in 1 we make the following assumptions:

- the input power satisfies $\mathbb{E}\{\tilde{\mathbf{x}}^H\tilde{\mathbf{x}}\} = \tilde{N}_t$;
- noise is complex zero-mean Gaussian: $n_i \sim \mathcal{CN}(0, \sigma^2)$ and the covariance matrix is $\mathbb{E}\{\tilde{\mathbf{n}}\tilde{\mathbf{n}}^H\} = \sigma^2\mathbf{I}_{\tilde{N}_r}$
- the column of $\tilde{\mathbf{H}}$ are normalized, $||\tilde{\mathbf{h}}_i||_2 = 1$;
- SNR $= 10\log_{10}\frac{\mathbb{E}\{||\tilde{\mathbf{H}}\tilde{\mathbf{x}}||_2^2\}}{\mathbb{E}\{||\tilde{\mathbf{n}}||_2^2\}}$ then SNR $= 10\log_{10}\frac{\mathbb{E}\{||\tilde{\mathbf{H}}\tilde{\mathbf{x}}||_2^2\}}{\tilde{N}_r\sigma^2}$
- the channel $\tilde{\mathbf{H}}$ is known at the receiver side.

2.1.1 Channel models

In this work we are going to work with two distinct channel models: i.i.d. Gaussian and Kronecker channel model.

i.i.d. Gaussian channel model

The i.i.d. Gaussian channel model is the most used in literature to test and compare MIMO detection algorithms. It has a strong underlying assumption for which every channel gain $\tilde{h}_{ij}$ is independent from all the others such that $\tilde{h}_{ij} \sim \mathcal{CN}(0, \frac{1}{\tilde{N}_r})$. This assumption makes the model pretty far from a realistic scenario where channels are spatially correlated, and they depend on each other. In this work, we normalize the columns of $\tilde{\mathbf{H}}$ such that each column $\tilde{\mathbf{h}}_i$ satisfies $||\tilde{\mathbf{h}}_i||_2 = 1$.

Kronecker channel model

In order to simulate the spatial correlation among channels, we consider the Kronecker channel model:

$$\tilde{\mathbf{H}} = \mathbf{R}_R^{1/2}\tilde{\mathbf{K}}\mathbf{R}_T^{1/2}, \tag{2.3}$$

where $\tilde{k}_{ij} \sim \mathcal{CN}(0, \frac{1}{\tilde{N}_r})$ and $\mathbf{R}_R, \mathbf{R}_T$ are the spatial correlation matrices at the receiver and the transmitter side respectively, that are generated according to the exponential correlation model [30]. The correlations at receiver and transmitter side are addressed with coefficients ρ_r, ρ_t respectively.

In this work, we normalize the columns of $\tilde{\mathbf{H}}$ such that each column $\tilde{\mathbf{h}}_i$ satisfies $||\tilde{\mathbf{h}}_i||_2 = 1$.

2.1.2 Real-valued system model

In order to remove the difficulties related to work with complex-valued systems, we simplify the model in Definition 1 by converting it into a real-valued system. The converted model treats the real and imaginary parts of complex values separately by defining a new transmitted symbol vector as

$$\mathbf{x} = [Re(\tilde{\mathbf{x}}^T)\ Im(\tilde{\mathbf{x}}^T)]^T, \tag{2.4}$$

where $Re(\tilde{\mathbf{x}})$ and $Im(\tilde{\mathbf{x}})$ are the real and imaginary parts of $\tilde{\mathbf{x}}$ respectively. Following the same structure, the received measurement vector and noise vector become

$$\mathbf{y} = [Re(\tilde{\mathbf{y}}^T)\ Im(\tilde{\mathbf{y}}^T)]^T, \tag{2.5}$$

$$\mathbf{n} = [Re(\tilde{\mathbf{n}}^T)\ Im(\tilde{\mathbf{n}}^T)]^T. \tag{2.6}$$

Then the new channel matrix is

$$\mathbf{H} = \begin{bmatrix} Re(\tilde{\mathbf{H}}) & -Im(\tilde{\mathbf{H}}) \\ Im(\tilde{\mathbf{H}}) & Re(\tilde{\mathbf{H}}) \end{bmatrix}. \tag{2.7}$$

Following the previous definitions of $\mathbf{y}, \mathbf{H}, \mathbf{x}, \mathbf{n}$, the new model is defined as follows.

Definition 2 (Real-valued MIMO)

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}, \tag{2.8}$$

$$N_r = 2\tilde{N}_r,\ N_t = 2\tilde{N}_t. \tag{2.9}$$

We call the new real-valued alphabet $\mathcal{A}$. In the QAM-M case, $\mathcal{A} = Re(\tilde{\mathcal{A}}) = Im(\tilde{\mathcal{A}})$.

2.1.3 MIMO detection

The goal of this thesis is to analyze and discuss alternative solutions for the problem of MIMO detection which we define in this way.

Definition 3 (MIMO Detection) *Given the model described in Definition 2, a MIMO detection method aims to recover the transmitted signals* $\mathbf{x}$ *given the receiver measurements* $\mathbf{y}$*, the estimated channel* $\mathbf{H}$ *and the covariance matrix of* $\mathbf{n}$*.*

The optimal detector for solving the problem in Definition 3 is the ML detector:

Definition 4 (ML Detector)

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}\in\mathcal{A}^{N_t}} p(\mathbf{y}|\mathbf{x},\mathbf{H}) = \arg\min_{\mathbf{x}\in\mathcal{A}^{N_t}} ||\mathbf{y}-\mathbf{H}\mathbf{x}||^2. \tag{2.10}$$

The best estimation $\hat{\mathbf{x}}$ *for* $\mathbf{x}$ *is the one the maximize the likelihood* $p(\mathbf{y}|\mathbf{x},\mathbf{H})$*. The problem of this method is the high computational complexity which is due to the exhaustive search over all possible values of* $\mathbf{x} \in \mathcal{A}^{N_t}$*.*

Hard Detectors

Hard detectors are a class of detectors that predict values $\mathbf{z} \in \mathbb{R}^{N_t}$ not necessarily from the alphabet $\mathcal{A}^{N_t}$. Here the predicted symbol $\hat{x}_i$ is the one that satisfies

$$\hat{x}_i = \arg\min_{x\in\mathcal{A}} |x - z_i|. \tag{2.11}$$

A detector more famous in literature than in practice is zero-forcing (ZF).

Definition 5 (ZF detector) *When* $N_r \geq N_t$ *and there are at least* N_t *linearly independent columns in* $\mathbf{H}$*, by applying the pseudo inverse* $\mathbf{H}^\dagger$ *of the channel to* $\mathbf{y}$

$$\hat{\mathbf{x}} = \mathbf{H}^\dagger\mathbf{y} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y} \approx \mathbf{z} + \mathbf{w} \tag{2.12}$$

we recover $\mathbf{z}$ *plus a new noise* $\mathbf{w}$ *that can grow very large when* $(\mathbf{H}^T\mathbf{H})$ *is almost singular and its inversion leads to very high values.*

In order to avoid the noise enhancement which arises in Definition 5, what we do in practice is to use a minimum mean squared error detector:

Definition 6 (MMSE detector)

$$\hat{\mathbf{x}} = (\mathbf{H}^T\mathbf{H} + \sigma^2\mathbf{I}_{N_t})^{-1}\mathbf{H}^T\mathbf{y} \approx \mathbf{z} + \mathbf{w} \tag{2.13}$$

We can notice that when SNR $\to \infty$, $\sigma \to 0$ and MMSE is equivalent to ZF. MMSE gives good results in practice but its performance is still far from the optimal one in different scenarios and it doesn't scale very well with the number of antennas because of the inverse of a matrix with dimensions $N_t \times N_t$. MMSE will be the benchmark method for this thesis.

Performance metrics

In this work we use the Symbol Error Rate (SER) as performance metric to compare detection algorithms at different SNR measures. The SER is defined in this way:

$$SER = \frac{no.\ of\ symbols\ in\ error}{total\ no.\ of\ transmitted\ symbols}. \tag{2.14}$$

2.1.4 QAM modulation

Modulation is the process of varying the phase and/or the amplitude and/or the frequency of a periodic waveform, called the carrier signal, to transmit information. Quadrature Amplitude Modulation produces a signal in which two carriers shifted in phase by 90 degrees (they are in quadrature) are modulated and combined. Basic signals exhibit only two positions which allow the transfer of either a 0 or 1. Using QAM there are many different points that can be used, each having defined values of phase and amplitude. The constellation points are usually arranged in a square grid, the constellation diagram (Figure 2.2), with equal vertical and horizontal spacing. In digital telecommunications the data is usually binary so the number of points in the grid is usually a power of 2. QAM is usually square, so the most common forms are QAM-4, QAM-16, QAM-64, and QAM-256. Different values are assigned to different points such that each single signal is able to transfer data at a much higher rate. However, if the mean energy of the constellation has to remain the same, the points must be closer to each other and thus they are more susceptible to noise.

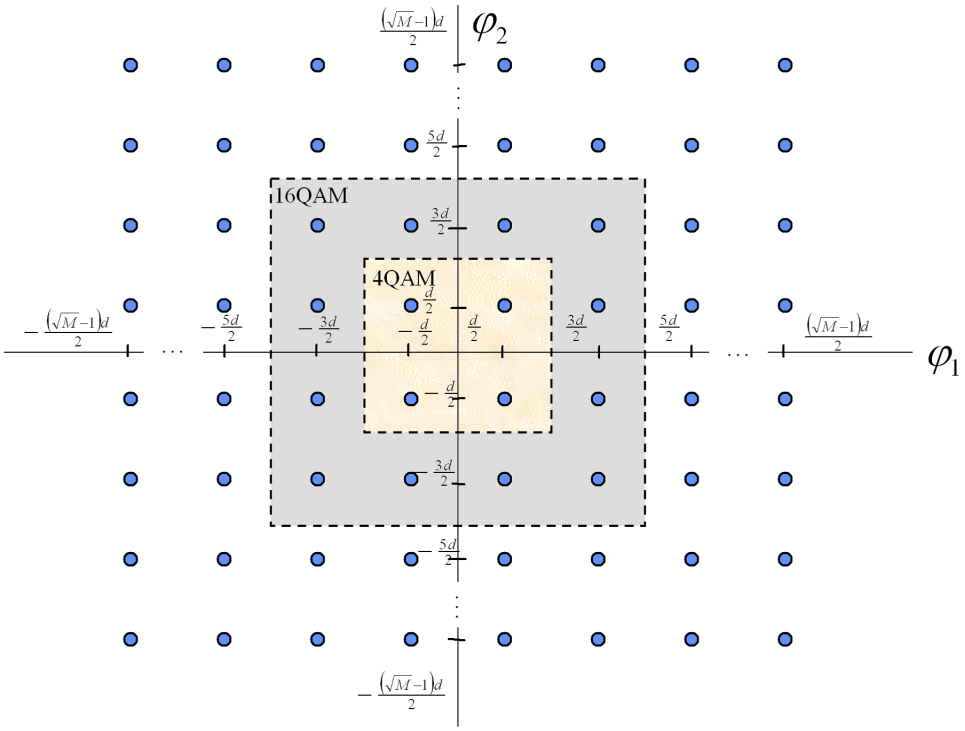


Figure 2.2: Squared QAM constellations [11]

For QAM-M, where M is a square number and a power of two, the

points assume the values

$$\pm\frac{d}{2}, \pm\frac{3d}{2}, ..., \pm\frac{(\sqrt{M}-1)d}{2}$$

along each axis, where d is the minimum distance between points in the constellation. Let's now consider the case where $d = 2$. The average power of QAM-M constellation is the double of the power along each axis:

$$E_{QAM-M} = 2\frac{\sum_{i=0}^{\sqrt{M/2}} 2(2i+1)^2}{\sqrt{M}} = 2\frac{M-1}{3}, \tag{2.15}$$

where i is an integer number. So, in order to work with normalized (on average) constellation symbols s, a QAM-M symbol $\tilde{s}$ becomes

$$s = \frac{\tilde{s}}{\sqrt{\frac{2(M-1)}{3}}}. \tag{2.16}$$

2.2 Inference in probabilistic graphical models

PGMs are graphical representations of probability distributions. They combine probability theory and graph theory in a unique framework. Graphical models allow to write efficient inference algorithms for several classes of probability distributions. Given a distribution $p(x_0, ..., x_{N-1})$, inference is the process of estimating unknown quantities, such as the marginal distribution of x_l (for all $l = 0, 1, ..., N-1$), from known quantities, such as the parameters of the distribution.

Given N discrete random variables $\mathcal{X} = \{x_0, .., x_{N-1}\}, x_l \in \mathcal{A}$, we consider probability distributions which can be factorized as follows

$$p(x_0, ..., x_{N-1}) = \frac{1}{Z}\prod_{k=0}^{M-1}\psi_k(\mathcal{X}_k). \tag{2.17}$$

where $\psi_k(\mathcal{X}_k)$ is well known as potential. It is a non-negative function of the subset of variables $\mathcal{X}_k \subseteq \mathcal{X}$. Z, instead, is a constant that ensures normalization and it takes the name of partition function.

A graphical model that corresponds to the probability distribution in (2.17) is the factor graph. The factor graph has a factor node for each factor ψ_k, and a variable node for each variable x_i. For each $x_l \in \mathcal{X}_k$, an undirected link is made between factor ψ_k and variable x_l. We are interested in factor graphs for marginal inference that is concerned with the computation of the distribution of a subset of variables. In our case, we want to compute the marginal distribution for each variable x_l which is equivalent to solve the following problem for every symbol $s \in \mathcal{A}$:

$$p(x_l = s) = \frac{1}{Z}\sum_{\mathcal{X}-x_l} p(x_0, ..., x_{l-1}, s, x_{l+1}, .., x_{N-1}). \tag{2.18}$$

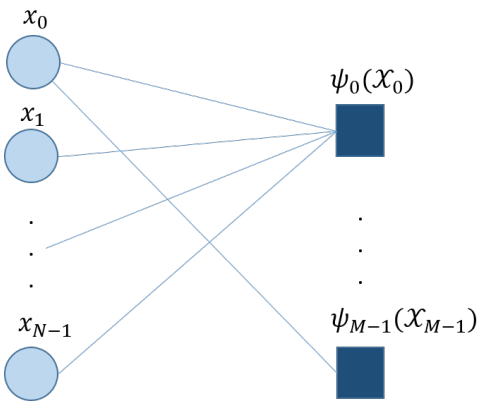


Figure 2.3: Graphical representation of a factor graph.

Solving the problem in (2.18) requires to iterate over all the possible realizations of $\mathcal{X}/\{x_l\}$. Therefore, the computational complexity of the task is in the order of $\mathcal{O}(|\mathcal{A}|^{N-1})$. If $|\mathcal{A}| > 1$, then the complexity increases exponentially with N and the problem becomes intractable.

2.2.1 Loopy belief propagation

Loopy Belief Propagation, which we will henceforth abbreviate as LBP, is an algorithm that calculates the approximate marginal distributions in graphical models that corresponds to the class of probability distributions in (2.17). The calculation is done by message-passing on the corresponding factor graph. There are two types of messages exchanged along the edges of the graph:

- the message $m_{k\to l}(x_l)$ sent from factor node k to variable node l;
- the message $m_{l\to k}(x_l)$ sent from variable node l to factor node k.

We denote with $ne(a)$ the neighbors of a node a. The update rules for the messages at iteration t are:

$$m^t_{l\to k}(x_l) \propto \prod_{a\in ne(l), a\neq k} m^{t-1}_{a\to l}(x_l), \tag{2.19}$$

$$m^t_{k\to l}(x_l) \propto \sum_{\mathcal{X}_k - x_l} \psi_k(\mathcal{X}_k) \prod_{b\in\mathcal{X}_k, b\neq l} m^t_{b\to k}(x_b). \tag{2.20}$$

After T iterations, the final step is to compute the marginal probabilities, or belief $b(x_l)$ for each variable node l by multiplying all the incoming messages:

$$b_l(x_l) \propto \prod_{k\in ne(l)} m^T_{k\to l}(x_l). \tag{2.21}$$

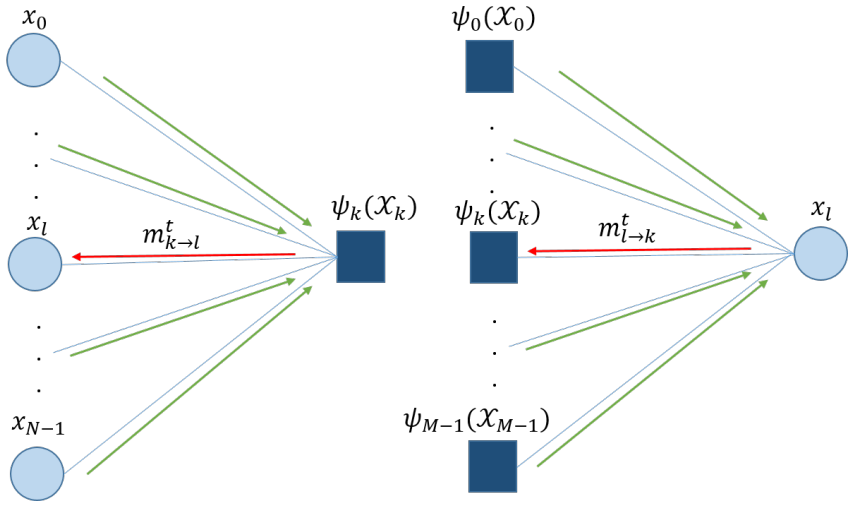


Figure 2.4: LBP message updates.

In each iteration, the number of exchanged messages is equal to the number of edges in the graph. Therefore, the overall complexity of a single iteration is proportional to the number of edges in the graph.

For graphs with loops, the number of iterations cannot be known a priori and convergence is not guaranteed in general. Instead, belief propagation is guaranteed to converge and to be exact on tree-structured graphs.

2.2.2 MIMO as a factor graph

Given the constrained linear system in Definition 2, the posterior probability $p(\mathbf{x}|\mathbf{y})$ is factorized with the Bayes's rule in the following way:

$$p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})p(\mathbf{x}) \propto \prod_{k=0}^{N_r-1} p(y_k|\mathbf{x}) \prod_{l=0}^{N_t-1} p(x_l), \tag{2.22}$$

where

$$p(\mathbf{y}|\mathbf{x}) \propto \exp(-\frac{1}{2\sigma^2}||\mathbf{y}-\mathbf{H}\mathbf{x}||^2), \tag{2.23}$$

that can be rewritten in the factorized form

$$p(\mathbf{y}|\mathbf{x}) \propto \prod_{k=0}^{N_r-1} p(y_k|\mathbf{x}) = \prod_{k=0}^{N_r-1} \exp(-\frac{1}{2\sigma^2}(y_k - \mathbf{h}_k^r\mathbf{x})), \tag{2.24}$$

where $\mathbf{h}_k^r$ is the k-th row of the matrix $\mathbf{H}$. We assume priors $p(\mathbf{x}) = \prod_{l=0}^{N_t-1} p(x_l)$, with the following distribution for each transmit symbol x_l:

$$p(x_l) = \sum_{s\in\mathcal{A}} \frac{1}{|\mathcal{A}|}\delta(x_l - s), \tag{2.25}$$

where δ is the Dirac delta distribution.

The goal of MIMO detection is to solve the following maximum a posteriori (MAP) problem:

$$\hat{\mathbf{x}}_{MAP} = \arg\max_{\mathbf{x}\in\mathcal{A}^{Nt}} p(\mathbf{x}|\mathbf{y}). \tag{2.26}$$

In order to solve the given inference problem we rely on a factor graph where probability densities $p(y_k|\mathbf{x})$ and $p(x_l)$ are designed as factor nodes and the variable x_l is the variable node. On the given factor graph we run the BP

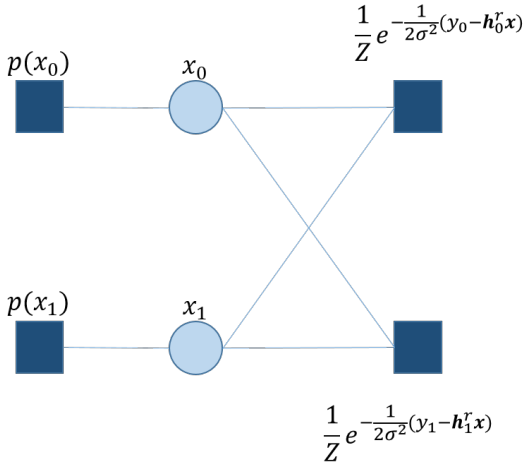


Figure 2.5: Factor graph for MIMO system 2×2.

algorithm where the update rules for belief propagation are

$$m^t_{l\to k}(x_l) \propto p(x_l) \prod_{a\neq k} m^{t-1}_{a\to l}(x_l), \tag{2.27}$$

for the message from variable node l to factor node k and

$$m^t_{k\to l}(x_l) \propto \int_{\mathbb{R}} p(y_k|\mathbf{x}) \prod_{b\neq l} m^t_{b\to k}(x_b) dy_k, \tag{2.28}$$

for the message from factor node k to variable node l. There is no closed form solution for the integral in the previous equation, so the computation of the integral is quite expensive. In order to overcome this issue, in the next section we reformulate the problem in another graphical model, the MRF.

After T iterations of LBP, the approximate solution to the MAP problem in (2.26) is given by computing for each entry of $\mathbf{x}$:

$$\hat{x}_l = arg\max_{x_l\in\mathcal{A}} p(x_l|\mathbf{y}) = arg\max_{x_l\in\mathcal{A}} p(x_l) \prod_a m^{T-1}_{a\to l}(x_l). \tag{2.29}$$

2.2.3 Markov random field

A MRF is a set of random variables described by an undirected graph $\mathcal{G} = \{V, E\}$ where for every node $i \in V$ it is true that

$$p(x_i|x \in \mathcal{X}/\{x_i\}) = p(x_i|ne(i)). \tag{2.30}$$

A factor graphs whose potentials are functions of one or two variables can be represented as a pair-wise MRF where each variable is associated to a node. Each self potential $\phi_i(x_i)$ is assigned to node $i \in V$ and each pair potential $\phi_{ij}(x_i, x_j)$ is assigned to an edge $e \in E$ that connects node $i \in V$ to node $j \in V$ as shown in Figure 2.6. In pair-wise MRF the probability

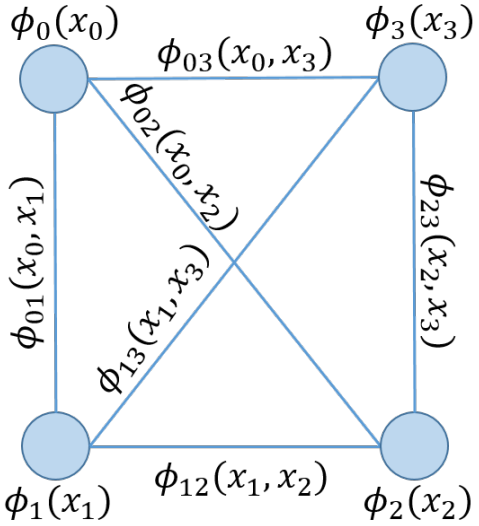


Figure 2.6: Fully-connected pair-wise Markov Random Fields with 4 variables

distribution has the following form:

$$p(x_0, ..., x_{N-1}) = \frac{1}{Z} \prod_{i \in V} \phi_i(x_i) \prod_{(i,j) \in E} \phi_{ij}(x_i, x_j), \tag{2.31}$$

where V, E are the set of nodes and edges in the corresponding pair-wise MRF and $|V| = N$. Deriving from Equations 2.21 and 2.20, the update rule for LBP messages in a pair-wise MRF is

$$m^t_{i \to j}(x_j) \propto \sum_{x_i} \phi_i(x_i)\phi_{ij}(x_i, x_j) \prod_{b \in ne(i)/j} m^{t-1}_{b \to i}(x_i), \tag{2.32}$$

where $ne(i)$ is the set of neighbors of i in the pair-wise MRF. The belief, or marginal distribution, of each variable can be computed by multiplying the

incoming messages with the self potential $\phi_l(x_l)$:

$$b_l(x_l) \propto \phi_l(x_l) \prod_{b \in ne(l)} m_{b \to l}(x_l). \tag{2.33}$$

2.2.4 MIMO as a Markov random field

By recalling (2.22), given the constrained linear system in Definition 2, the posterior probability $p(\mathbf{x}|\mathbf{y})$ is factorized with the Bayes's rule in the following way:

$$p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})p(\mathbf{x}) = \exp(-\frac{1}{2\sigma^2}||\mathbf{H}\mathbf{x} - \mathbf{y}||^2)p(\mathbf{x}). \tag{2.34}$$

The goal is to solve the MAP problem presented in (2.26). The posterior probability in (2.34) can be factorized into a pairwise MRF:

$$p(x_0, .., x_{Nt-1}|\mathbf{y}) \propto \prod_{i \in V} \phi_i(x_i) p_i(x_i) \prod_{i \neq j, i, j \in V} \phi_{ij}(x_i, x_j), \tag{2.35}$$

where

$$\phi_i(x_i) = e^{\frac{1}{\sigma^2}(\mathbf{y}^T \mathbf{h}_i x_i - \frac{1}{2}\mathbf{h}_i^T \mathbf{h}_i x_i^2)}, \tag{2.36}$$

$$\phi_{ij}(x_i, x_j) = e^{-\frac{1}{\sigma^2}\mathbf{h}_i^T \mathbf{h}_j x_i x_j} \tag{2.37}$$

and σ^2 is the noise variance as in Definition 1 and $\mathbf{h}_i$ is the i-column of $\mathbf{H}$. By applying the LBP algorithm described in Equations 2.32, where the initial messages are the uniform prior probabilities over symbols, we can approximate the solution of the MAP problem in (2.26) by solving a simpler MAP problem for each variable. Indeed, after convergence, for each variable x_i we compute (2.33) and hard detect the transmitted symbol $\hat{x}_i$ with

$$\hat{x}_l = \arg\max_{x_l \in \mathcal{A}} b_l(x_l). \tag{2.38}$$

From the experiments in [16] we know that applying the standard LBP with uniform prior doesn't lead to good results when solving MIMO detection. Moreover, when the size of the graph increases, the product of the incoming messages in (2.32) may easily result in an arithmetic underflow/overflow condition. Therefore, the first modification that they propose in [16] is to work with log messages:

$$\log m_{i \to j}^t(x_j) \propto \log \sum_{x_i} \exp(\log \phi_i(x_i) + \log \phi_{ij}(x_i, x_j) + \sum_{b \in ne(i)/j} \log m_{b \to i}^{t-1}(x_i)). \tag{2.39}$$

Then we apply the following approximation

$$\sum_i \exp^{a_i} \approx \exp^{\max_i a_i}, \ \mathbf{a} \in \mathbb{R}^N, \ N \in \mathbb{N}, \tag{2.40}$$

to obtain:

$$\log m_{i\to j}^{t}(x_j) \propto \max_{x_i}(\log \phi_i(x_i) + \log \phi_{ij}(x_i, x_j) + \sum_{b\in ne(i)/j} \log m_{b\to i}^{t-1}(x_i)). \tag{2.41}$$

In the experiments in [16], the authors show that this variant is still not good enough to do better than the MMSE detector (Definition 6). In order to assist the LBP algorithm to concentrate on the right target, the authors suggest to combine the result in the MMSE detector in the LBP iterations as a pseudo prior. By following the same reasoning in Section 3.1, the pseudo log prior $\log p_l(x_l)$ is set to

$$\log p_l(x_l) = -\frac{(z_l^{MMSE} - x_l)^2}{2c_{ll}^{MMSE}} \tag{2.42}$$

where

$$\mathbf{z}^{MMSE} = (\mathbf{H}^T\mathbf{H} + \sigma^2\mathbf{I}_{N_t})^{-1}\mathbf{H}^T\mathbf{y} \tag{2.43}$$

and

$$\mathbf{C}^{MMSE} = \sigma^2(\mathbf{H}^T\mathbf{H} + \sigma^2\mathbf{I}_{N_t})^{-1}. \tag{2.44}$$

From now on we discard the log notation, and we initialize the messages as

$$m_{i\to j}^{0}(x_j) \propto \lambda p_j(x_j), \tag{2.45}$$

we rewrite the message update equation for $t > 0$ as follows

$$m_{i\to j}^{t}(x_j) \propto \max_{x_i}(\lambda p_i(x_i) + \phi_i(x_i) + \phi_{ij}(x_i, x_j) + \sum_{b\in ne(i)/j} m_{b\to i}^{t-1}(x_i)). \tag{2.46}$$

and the log belief $b_l(x_l)$ becomes

$$b_l(x_l) \propto \lambda p_l(x_l) + \phi_l(x_l) + \sum_{b\in ne(l)} m_{b\to l}^{T}(x_l), \tag{2.47}$$

where λ is a parameter to tune and the transmitted symbols $\hat{x}_l$ are hard detected in the following way

$$\hat{x}_l = \arg\max_{x_l\in\mathcal{A}} b(x_l). \tag{2.48}$$

2.3 Sparse linear inverse problem

In this section we examine a class of low-complexity iterative thresholding algorithms to recover unknown sparse signals from a set of linear measurements. We recall the same problem in Definition 3 by relaxing the assumption on $\mathbf{x}$. Here $\mathbf{x} \in \mathbb{R}^{Nt}$ and it is sparse. We address it as the the sparse

linear inverse problem that is usually solved by turning it into the following convex optimization problem

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \frac{1}{2}||\mathbf{y} - \mathbf{H}\mathbf{x}||_2^2 + \lambda||\mathbf{x}||_1, \tag{2.49}$$

where $\lambda > 0$ is a tunable parameter that controls the tradeoff between sparsity and the quality of recovery of $\hat{\mathbf{x}}$.

2.3.1 ISTA

An appealing solution to problem in (2.49) is A class of techniques denoted as iterative soft thresholding algorithms. The scheme of an iteration of ISTA is presented in the following formulation.

Scheme 1 (ISTA)

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \alpha\mathbf{H}^T\mathbf{r}^t \tag{2.50}$$

$$\hat{\mathbf{x}}^{t+1} = \eta_{ST}(\mathbf{z}^t, \lambda) \tag{2.51}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} \tag{2.52}$$

In Scheme 1, $\hat{\mathbf{x}}^0 = 0$, $\mathbf{r}^0 = \mathbf{y}$, $\alpha > 0$ is a stepsize, $\mathbf{r}^t$ the residual error at iteration t and η_{ST} $\mathbb{R}^{Nt} \rightarrow \mathbb{R}^{Nt}$ is a component-wise soft-thresholding shrinkage function:

$$\eta_{ST}(z_j, \lambda)_j = sign(z_j)\max(|r_j| - \lambda, 0). \tag{2.53}$$

The solution to the problem in (2.49), is given after T iterations by $\hat{\mathbf{x}}^T$.

2.3.2 Approximate message passing

In [12], the authors introduce a simple modification to ISTA algorithm by taking inspiration from belief propagation in graphical models. They propose an algorithm well known as AMP that exhibits the same low computational complexity of ISTA with faster converge and stronger reconstruction power.

Scheme 2 (AMP for Sparse Linear Inverse Problem)

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \mathbf{H}^T\mathbf{r}^t \tag{2.54}$$

$$\hat{\mathbf{x}}^{t+1} = \eta_{ST}(\mathbf{z}^t, \lambda^t) \tag{2.55}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} + b^t\mathbf{r}^t \tag{2.56}$$

The algorithm start by initializing $\hat{\mathbf{x}}^0 = \mathbf{0}$ and $\mathbf{r}^0 = y$. Then b^t and λ^t are computed as follows

$$b^t = \frac{1}{N_r}||\hat{\mathbf{x}}^t||_0, \tag{2.57}$$

$$\lambda^t = \frac{\theta}{\sqrt{N_r}}||\mathbf{r}^t||_2, \tag{2.58}$$

where θ is a parameter to tune. By comparing ISTA and AMP we can notice two main differences:

- in AMP the residual $\mathbf{r}^t$ contains the so called Onsager correction term $b^t\mathbf{r}^{t-1}$;
- AMP shrinkage threshold λ^t depends on the iteration t.

As shown in [13], the AMP model is provably accurate in the large-system limit, when $N_t, N_r \to \infty$. The Onsager correction assures a better performance than ISTA when $\mathbf{H}$ is i.i.d. Gaussian with $h_{ij} \sim \mathcal{N}(0, \frac{1}{N_r})$ because we can model the input $\mathbf{z}^{t+1}$ of the shrinkage function as follows [5]

$$\mathbf{z}^t = \mathbf{x}^* + \mathcal{N}(\mathbf{0}, (\sigma^t)^2\mathbf{I}_{N_t}). \tag{2.59}$$

In (2.59) we are saying that the shrinkage input is an additive white Gaussian noise-corrupted (AWGN-corrupted) version of the true signal $\mathbf{x}^*$ with known variance $(\sigma^t)^2$. In other words, the purpose of the Onsager correction is to decouple and make Gaussian the errors across iterations. Therefore, AMP soft threshold shrinkage function can be substitute by a denoiser function $\eta(\mathbf{z}^t, (\sigma^t)^2)$ which takes the variance of the noise $(\sigma^t)^2$ as input. In this way we have a new definition for b^t:

$$b^t = \frac{1}{N_r}\sum_{j=0}^{N_t-1}\frac{\partial\eta_j(\mathbf{z}^t, (\sigma^t)^2)}{\partial z_j}, \tag{2.60}$$

while $(\sigma^t)^2$ has not a general formulation because it strongly depends on the problem we are tempting to solve. We rewrite AMP in the denoiser version because it will be useful for later comparisons:

Scheme 3 (AMP)

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \mathbf{H}^T\mathbf{r}^t \tag{2.61}$$

$$\hat{\mathbf{x}}^{t+1} = \eta(\mathbf{z}^t, (\sigma^t)^2) \tag{2.62}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} + b^t\mathbf{r}_t \tag{2.63}$$

AMP in the latter Scheme 3 is equivalent to AMP in Scheme 2 where $\eta(\mathbf{z}, (\sigma^t)^2) = \eta_{ST}(\mathbf{z}, \lambda^t)$. In the next section we provide the denoiser η and the formulas for $(\sigma^t)^2$ and b^t to solve MIMO detection with AMP.

2.4 AMP for MIMO detection

The first attempt to solve MIMO detection with an approach that relies on AMP was done in [22]. The authors call the algorithm LAMA and they provide a derivation of the detector for the complex-valued system in [23] by starting from the LBP message update equations in the MIMO factor graph presented in Section 2.2.2. In the derivation, the authors rely on the theorems provided and proved in the derivation of the original AMP algorithm in [32]. Here we provide a derivation of the algorithm for the real-valued MIMO system.

The first step is to recall the a posteriori probability and the LBP update message equations in Section 2.2.2. Now we consider the case where the entries h_{ij} of $\mathbf{H}$ are i.i.d. and we have a large sized system $N_t \to \infty$, $N_r \to \infty$. Moreover, let $\hat{x}^t_{l\to k}$ and $\sigma^2\tau^t_{l\to k}$ be the mean and the variance of the distribution of the message $m^t_{l\to k}$. Under the previous assumptions, we can use Lemma 5.4.1 in [32] to approximate the messages $m^t_{k\to l}(x_l)$ from factor node k to variable node l with the Gaussian density:

$$\hat{\phi}^t_{k\to l}(x_l) = \sqrt{\frac{h_{kl}^2}{\pi\sigma^2(1+\tau^t_{k\to l})}}\exp(-\frac{(h_{kl}s_l - r^t_{k\to l})^2}{\sigma^2(1+\tau^t_{k\to l})}), \tag{2.64}$$

where $\tau^t_{k\to l} \approx \tau^t$ and the residual and variance terms are given by:

$$r^t_{k\to l} = y_k - \sum_{b\neq l} h_{k,b}\hat{x}^t_{b\to k},\ \tau^t_{k\to l} = \sum_{b\neq l} h^2_{k,b}\tau^t_{b\to k}. \tag{2.65}$$

By following the derivation in B, at the next iteration the messages from variable node to factor node are

$$m^{t+1}_{l\to k}(x_l) \propto \prod_{a\neq k} m^t_{a\to l}(x_l)p(x_l) = \phi^{t+1}_{l\to k}(x_l)(1+\mathcal{O}(\frac{x_l^2}{N_r})), \tag{2.66}$$

where we use the $\mathcal{O}$ notation to describe the error term in the approximation to our mathematical function. In the case of large system size ($N_r \to \infty$), we can approximate $\phi^{t+1}_{l\to k}(x_l)$ to the density function

$$\phi^{t+1}_{l\to k}(x_l) = f(x_l|\sum_{a\neq k} h_{a,l}r^t_{a\to l}, \sigma^2(1+\tau^t)), \tag{2.67}$$

where

$$f(s|z,\tau) = \frac{1}{Z}\exp(-\frac{(z-s)^2}{\tau})\sum_{s\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\delta(x_l - s). \tag{2.68}$$

The next step is to set the mean and the variance of the message $m^{t+1}_{l\to k}$ at the next iterations to:

$$\hat{x}^{t+1}_{l\to k} = F(\sum_{a\neq k} h_{a,l}r^t_{a\to l}, \sigma^2(1+\tau^t)), \tag{2.69}$$

$$\tau_{l\to k}^{t+1} = \frac{1}{\sigma^2} G(\sum_{a\neq k} h_{a,l} r_{a\to l}^{t}, \sigma^2(1+\tau^t)), \tag{2.70}$$

where

$$F(z,\tau) = \frac{\sum_{s\in\mathcal{A}} s \exp(-\frac{(z-s)^2}{2\tau})}{\sum_{s\in\mathcal{A}} \exp(-\frac{(z-s)^2}{2\tau})}, \tag{2.71}$$

and

$$G(z,\tau) = F^{(2)} - F^2, \tag{2.72}$$

such that

$$F^{(2)} = \frac{\sum_{s\in\mathcal{A}} s^2 \exp(-\frac{(z-s)^2}{2\tau})}{\sum_{s\in\mathcal{A}} \exp(-\frac{(z-s)^2}{2\tau})}. \tag{2.73}$$

In (2.74) and (2.75) we expand the messages $\hat{x}_{l\to k}^{t+1}$ and $r_{k\to l}^{t+1}$ by parting the edge independent contribution $(\hat{x}_l^{t+1}, r_k^{t+1})$ of the messages from the dependent one $(\delta\hat{x}_{l\to k}^{t+1}, \delta r_{k\to l}^{t+1} = \mathcal{O}(\frac{1}{\sqrt{N_t}}))$:

$$\hat{x}_{l\to k}^{t+1} = \hat{x}_l^{t+1} + \delta\hat{x}_{l\to k}^{t+1} + \mathcal{O}(\frac{1}{\sqrt{N_t}}), \tag{2.74}$$

$$r_{k\to l}^{t+1} = r_k^{t+1} + \delta r_{k\to l}^{t+1} + \mathcal{O}(\frac{1}{\sqrt{N_t}}). \tag{2.75}$$

After substituting Equations 2.74, 2.75 in the general Equations 2.69, 2.65, then writing the Taylor expansion of the latter and finally following the rest of the derivation in Lemma 5.3.4 in [32] (here η stands for our F) we can write:

$$\hat{x_l}^{t+1} = F(\hat{x}_l + \sum_a h_{a,l} r_a^t, \sigma^2(1+\tau^t)), \tag{2.76}$$

$$r_k^{t+1} = y_k - \sum_b h_{k,b}\hat{x}_b^{t+1} + \frac{1}{\delta N_r} \sum_b^{N_t-1} \frac{\partial F(\hat{x_b} + \sum_a h_{a,b} r_a^t, \sigma^2(1+\tau^t))}{\partial(\hat{x_b} + \sum_a h_{a,b} r_a^t)}. \tag{2.77}$$

We recall

$$\tau_{k\to l}^{t+1} = \sum_{b\neq l} H_{k,b}^2 \tau_{b\to k}^t, \tag{2.78}$$

and we approximate $\tau_{k\to l}^{t+1}$ with an edge independent quantity τ^{t+1}

$$\tau_{k\to l}^{t+1} \approx \tau^{t+1} = \frac{1}{N_r}\sum_{k}^{N_r-1}\sum_{b}^{N_t-1} h_{k,b}^2 \tau_{b\to k}^{t} \approx$$

$$\approx \frac{1}{N_r}\sum_{k}^{N_r-1}\sum_{b}^{N_t-1} h_{k,b}^2 \frac{1}{\sigma^2} G(\hat{x_b}^t + \sum_{a}^{N_r-1} h_{a,b} r_a^t, \sigma^2(1+\tau^t)) =$$

$$= \frac{1}{\sigma^2 N_r}\sum_{b}^{N_t-1} G(\hat{x_b}^t + \sum_{a}^{N_r-1} h_{a,b} r_a^t, \sigma^2(1+\tau^t)) \sum_{k}^{N_r-1} h_{k,b}^2 =$$

$$= \frac{1}{\sigma^2 N_r}\sum_{b}^{N_t-1} G(\hat{x_b}^t + \sum_{a}^{N_r-1} h_{a,b} r_a^t, \sigma^2(1+\tau^t)),$$

since we assume in Definition 1 to work with column normalized $\mathbf{H}$.

Until here, we can summarize the iterative scheme as it follows:

$$\hat{x_l}^{t+1} = F(\hat{x}_l + \mathbf{h}_l^T \mathbf{r}^t, \sigma^2(1+\tau^t)) \tag{2.79}$$

$$r_k^{t+1} = y_k - \mathbf{h}_k^r \hat{\mathbf{x}}^{t+1} + \frac{1}{\delta N_r}\sum_{b}^{N_t-1} \frac{\partial F(\hat{x}_b + \mathbf{h}_b^T \mathbf{r}^t, \sigma^2(1+\tau^t))}{\partial(\hat{x}_b + \mathbf{h}_b^T \mathbf{r}^t)} \tag{2.80}$$

$$\tau^{t+1} = \frac{1}{\sigma^2 N_r}\sum_{b}^{N_t-1} G(\hat{x_b}^t + \mathbf{h}_b^T \mathbf{r}^t, \sigma^2(1+\tau^t)). \tag{2.81}$$

Now we define $z_l^t = \hat{x}_l + \mathbf{h}_l^T \mathbf{r}^t$ and rewrite the scheme in the matrix form by letting the functions F and G to work element wise and by using the notation $\langle\rangle$ to define $\langle \mathbf{v} \rangle = \sum_{k=0}^{N-1} v_k$:

$$\mathbf{z}^t = \hat{\mathbf{x}} + \mathbf{H}^T \mathbf{r}^t \tag{2.82}$$

$$\hat{\mathbf{x}}^{t+1} = F(\mathbf{z}^t, \sigma^2(1+\tau^t)) \tag{2.83}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} + \frac{1}{\delta N_r}\langle \frac{\partial F(\mathbf{z}^t, \sigma^2(1+\tau^t))}{\partial \mathbf{z}^t} \rangle \mathbf{r}^t \tag{2.84}$$

$$\tau^{t+1} = \frac{1}{\sigma^2 N_r}\langle G(\mathbf{z}^t, \sigma^2(1+\tau^t)) \rangle. \tag{2.85}$$

At the end we define $\sigma_t^2 = \sigma^2(1+\tau^t)$ and by exploiting the fact that $G(z,\tau) = \tau \frac{\partial F(z,\tau)}{\partial z}$ (Proof A) we can rewrite the Onsager term in the following way:

$$\frac{1}{\delta N_r}\langle \frac{\partial F(\mathbf{z}^t, \sigma^2(1+\tau^t))}{\partial \mathbf{z}^t} \rangle \mathbf{r}^t = \frac{1}{\delta N_r}\frac{1}{\sigma^2(1+\tau^t)} G(\mathbf{z}^t, \sigma^2(1+\tau^t))\mathbf{r}^t = \tag{2.86}$$

$$= \frac{1}{\delta N_r}\frac{1}{\sigma^2(1+\tau^t)}(\tau^{t+1}\sigma^2 N_r)\mathbf{r}^t = \frac{\tau^{t+1}}{\delta(1+\tau^t)}\mathbf{r}^t. \tag{2.87}$$

Following the previous derivation we define the AMP iterative algorithm for real-valued MIMO systems:

Definition 7 (AMP-Gaussian) *Initialize* $\hat{\mathbf{x}}^0 = \mathbb{E}_X[X] = 0,\ \mathbf{r}^0 = \mathbf{y},\ \tau^0 = \frac{\mathbb{VAR}_X[X]=1}{\sigma^2 N_r}$. *Then for every iteration* $t = 0, 1, 2..., T-1$ *we compute the following steps:*

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \mathbf{H}^T \mathbf{r}^t \tag{2.88}$$

$$\hat{\mathbf{x}}^{t+1} = F(\mathbf{z}^t, \max(\sigma^2, \lambda)(1+\tau^t)) \tag{2.89}$$

$$\tau^{t+1} = \frac{1}{\max(\sigma^2, \lambda) N_r} \langle G(\mathbf{z}^t, \max(\sigma^2, \lambda)(1+\tau^t)) \rangle \tag{2.90}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} + \frac{\tau^{t+1}}{\delta(1+\tau^t)} \mathbf{r}^t, \tag{2.91}$$

where δ *and* λ *are parameters to tune and* $F(z, \tau)$ *is the Gaussian denoiser defined in* (2.71) *and* $G(z, \tau)$ *the function defined in Equation* (2.73). *Both* $F(z, \tau)$ *and* $G(z, \tau)$ *operate element-wise on vectors. The operator* $\max(\sigma^2, \lambda)$ *serves to avoid numerical issues in the computation of* $F(z, \tau)$ *and* $G(z, \tau)$ *when* $\sigma^2 \to 0$.

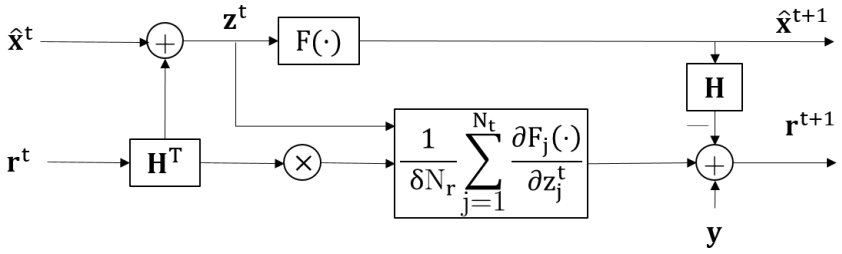


Figure 2.7: Iteration t *of AMP-Gaussian. This graphical representation does not include the computation of* τ^{t+1}.

2.5 Supervised deep learning

Supervised DL is a subset of machine learning that uses a training dataset $\{(\mathbf{y}^{(d)}, \mathbf{x}^{(d)})\}_{d=1}^D,\ \mathbf{y}^{(d)} \in \mathbb{R}^{M^0}, \mathbf{x}^{(d)} \in \mathbb{R}^{M^T}$, where D and T are positive integers, to learn the parameters of an Artificial Neural Networks (ANNs), with the goal to predict the unknown label $\hat{\mathbf{x}}$ associated with new data $\mathbf{y}$.

Multilayer Perceptron (MLP) is a class of ANNs that simply concatenates T basic blocks, commonly known as layers:

$$\mathbf{V}^t \in \mathbb{R}^{M^t} \times \mathbb{R}^{M^{t-1}},\ \mathbf{y}^t = \Omega(\mathbf{V}^t \mathbf{y}^{t-1}) : \mathbb{R}^{M^{t-1}} \to \mathbb{R}^{M^t}, \tag{2.92}$$

where $\mathbf{V}^t$ is a linear operator and Ω is a non linear function. The whole net is the result of the concatenation of T layers:

$$\hat{\mathbf{x}} = g(\mathbf{y}) = g(\mathbf{y}^0) = \Omega(\mathbf{V}^T \Omega(\mathbf{V}^{T-1} ... \Omega(\mathbf{V}^1 \mathbf{y}^0))). \tag{2.93}$$

During training, we learn the values of $\mathbf{V}^1, ..., \mathbf{V}^T$ in order to reduce the prediction error on the labels in the training data. It is common practice to define a loss function L that quantifies the prediction error according to the given problem. Therefore, DL becomes the following minimization problem:

$$\arg\min_{\mathbf{V}^1,...,\mathbf{V}^T} L([\mathbf{x}^{(1)}, ..., \mathbf{x}^{(D)}], [\hat{\mathbf{x}}^{(1)}, ..., \hat{\mathbf{x}}^{(D)}]) \tag{2.94}$$

In order to find the optimal parameters $\mathbf{V}^1, ..., \mathbf{V}^T$ that minimize the loss function on the training dataset, we firstly compute the gradients of L with respect to the parameters. Then, we update the parameters according to the respective gradients by applying gradient descent. In this work, we consider the Adam optimizer [25] as the update rule for the parameters.

Before the start of the learning procedure, the training dataset is split in batches of the same size. When the training algorithm starts, it iterates over epochs and in each epoch we repeat a training step for every batch of the dataset. A training step consists in two passes:

- forward pass: it computes the outputs $\hat{\mathbf{x}}^{(1)}, ..., \hat{\mathbf{x}}^{(D)}$ given the inputs and labels $\{(\mathbf{y}^{(d)}, \mathbf{x}^{(d)})\}_{d=1}^{D}$, where D now indicates the size of the batch;
- backward pass: update the values of parameters $\mathbf{V}^1, ..., \mathbf{V}^T$ by calculating their gradients with respect all the batch and applying gradient descent.

2.5.1 Preventing overfitting

Solving the minimization problem in (2.94) for the training dataset does not ensure to reduce the prediction error on the unknown labels in the test dataset. Therefore, it becomes crucial to choose a training dataset and a loss function that help to minimize the gap between the training loss and the test loss. This gap is well known as the generalization error. The generalization error tends to increase when the training dataset is too small with respect to the degree of freedom of the model (the number of parameters to learn can be used as a measure of the degree of freedom). This flexibility also means that the model is prone to overfit on the training dataset.

The best way to prevent overfitting is to increase the training dataset. However, this is not always possible. Therefore, in the DL community several techniques have been proposed to avoid the problem. Here, we present only a subset of them that have been used for the scope of this thesis.

Early stopping

Early stopping is a technique that provides a criteria to decide when to stop the training according to the prediction error on a validation dataset that

is separated from the training and testing dataset before the start of the training. On one hand, as the training steps go by, the prediction error on the training dataset naturally goes down. On the other hand, after a while, the validation error stops decreasing and starts to go back up. When it does, it means that the model has started to overfit the training data. Therefore, we stop the training when we observe that the validation error stops to decrease.

Dropout

Dropout is a technique that can be applied to any layer t of the network (except the output layer). At every training step, each entry of $\mathbf{y}^t$ drops out temporarily with a probability p (also known as dropout rate). When an entry drops out, it's value is set to zero. Therefore, its contribution will be ignored during the current training step. Dropout is only applied during training and it is not used during testing.

Cross validation

In the model, the parameters $\mathbf{V}^1, ..., \mathbf{V}^T$ are not the only one that need to be optimized. There is another class of parameters, called hyperparameters, that cannot be learned with the backpropagation procedure describe above, but that still need to be tuned to achieve the desired performance from the model. Examples of hyperparameters are the number of layers and the hidden dimension M^t of every layer t, the learning rate for gradient descend and the dropout rate.

To find the best values of the hyperparameters we need a validation dataset of samples not observed during the training. For every predefined combinations of hyperparameters, at the end of the training, we validate the performance of the model on the validation dataset. The model with the combination of hyperparameters that leads to the best performance is the best candidate for testing.

2.5.2 Vanishing gradient problem

When the number of layers in ANN increases, we need to face with the vanishing gradient problem. It arises when the gradients of the weights shrink as they backpropagate to lower layers (towards the input layer). If gradient values become extremely small, they don't contribute in learning the new values of the weights. This can happen when the network works with activation functions which allow values of the gradient in the interval $[0, 1]$, like sigmoid or hyperbolic function. Since the gradients are multiplied among each other a number of times equal to the number of layers T (through which they are backpropagated), the total gradient is the product of T terms whose values are between $[0, 1]$.This means that the total gradient goes toward the

value zero exponentially with T. To solve this problem there are several solutions described in literature. Here, we present only the three of them that are of interest for this work. The first one is to use activation functions whose gradient is one for the input values which lead to non-zero active values. One example, is the ReLU function:

$$ReLU(x) = \max(0, x). \tag{2.95}$$

The second solution is to introduce in the loss function a penalty term L^{t+1} for the output of each layer $t = 0, 1, ..., T-1$:

$$L = \lambda^T L^T + \lambda^{T-1} L^{T-1} + ... + \lambda^1 L^1, \tag{2.96}$$

where each λ^t is an hyperparameter to tune such that

$$\sum_{t=1}^{T} \lambda^t = 1. \tag{2.97}$$

The third solution is to rely on ad-hoc networks that prevent the vanishing gradient problem trough a gate mechanism. LSTM [21] and GRU [10] are examples of such networks. They are meant to process variable-length sequences of inputs, by keeping an internal state that serves as a memory.

2.5.3 Graph neural networks

Performing inference tasks in PGMs is intractable in general. There are various techniques for approximate inference in PGM and one of the most promising one is Graph Neural Networks (GNNs). GNNs combine the advantages of DL and PGMs in a unique framework to capture the structure of the data into feature vectors that are updated through message passing between nodes. Indeed, GNNs have vector-valued nodes $\mathbf{u}_i \in \mathbb{R}^{S_u}$, where S_u is a positive integer, that encode the probabilistic information about the variables in the respective graphical model originated from a probability distribution with the same structure in (2.17).

The values of $\{\mathbf{u}_i\}$ are iteratively updated by a RNN whose input includes the value of $\mathbf{u}_i$ at the previous iteration together with the information coming from the neighbor nodes states $\mathbf{u}_j : j \in ne(i)$ on the specified graph $\mathcal{G} = \{V, E\}$, where V, E are the set of nodes and edges respectively in $\mathcal{G}$.

The are several architectures for GNNs in literature. In this work we follow the guidelines in [41] that is built upon the Gated Graph Neural Networks (GG-NNs) [28], which adds a Gated Recurrent Unit (GRU) [10] at each node to integrate incoming information with past states. In this case, GNNs are inspired by the scheme of LBP and they are composed by three main modules: a propagation, an aggregation and a readout module. The first two modules operate at every iteration t while the readout module is involved only in the last iteration T.

The propagation module outputs the updated message $\mathbf{m}_{i\to j}^t$ for each direct edge $e_{ij} \in E$ by

$$\mathbf{m}_{i\to j}^t = M(\mathbf{u}_i^{t-1}, \mathbf{u}_j^{t-1}, \epsilon_{ij}), \tag{2.98}$$

where ϵ_{ij} is the information associated to the edge e_{ij} and M is a MLP with ReLU as activation functions. Therefore, the information exchanged between two nodes at iteration t is an encoding of the concatenation of the feature vectors of the two nodes and the information along the direct edge between them.

The aggregation module operates at a node level by aggregating the incoming messages $\mathbf{m}_{j\to i}^t$ at node $i \in V$, with $j \in ne(i)$, by following

$$\mathbf{u}_i^t = U(\mathbf{u}_i^{t-1}, \sum_{j\in ne(i)} \mathbf{m}_{j\to i}^t), \tag{2.99}$$

where U is a GRU [10].

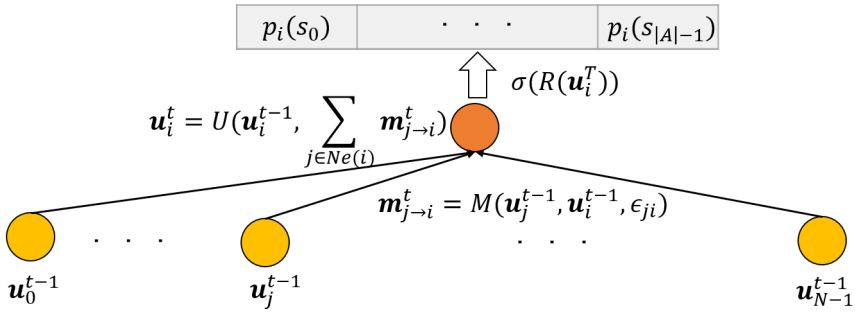


Figure 2.8: GNNs' message and state updates.

After a fixed number of iterations T, which is an hyperparameter to choose carefully, the feature vectors $\mathbf{u}_i$ are used to make inference with the readout module. If the problem that we want to solve is to compute the marginal probabilities of discrete random variables as described in (2.18), the readout module is a MLP R of the feature vector $\mathbf{u}_i$ followed by the softmax function $\sigma : \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}^{|\mathcal{A}|}$. The softmax function maps the non-normalized output $\mathbf{z}$ of the network R to a probability distribution over predicted output symbols:

$$\hat{p}(x_i = s_k) = \sigma(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{|\mathcal{A}|} e^{z_j}}. \tag{2.100}$$

The parameters of M, U and R are shared across all the graph and we learn them with supervised learning by minimizing the loss function between the true probabilities $p(x_i)$ and the predicted ones $\hat{p}(x_i)$.

A good candidate loss function L is the cross-entropy:

$$L = -\sum_{x_i} p(x_i) \log \hat{p}(x_i), \tag{2.101}$$

where $\hat{p}(x_i)$ is the output of the T-layers GNN.

2.5.4 Learned ISTA

Learned ISTA (LISTA) has been introduced in [18] by rewriting ISTA in the following scheme:

$$\hat{\mathbf{x}}^{t+1} = \eta_{ST}(\mathbf{S}\hat{\mathbf{x}}^t + \mathbf{B}\mathbf{y}, \lambda^t), \tag{2.102}$$

with

$$\mathbf{B} = \beta\mathbf{H}^T,\ \mathbf{S} = \mathbf{I}_{Nt} - \mathbf{B}\mathbf{H},\ \lambda^t = \lambda. \tag{2.103}$$

LISTA is the result of unfolding T iterations of (2.102) and freeing the parameters $\Theta = \{\lambda^t, \mathbf{S}, \mathbf{B}\}$ to be learned in a supervised learning fashion. Given a dataset $\{(\mathbf{y}^{(d)}, \mathbf{x}^{(d)})\}_{d=1}^{D}$, $\mathbf{y}^{(d)} \in \mathbb{R}^{N_r}, \mathbf{x}^{(d)} \in \mathbb{R}^{N_t}$ the learnable parameters are optimized through backpropagation by minimizing the loss function:

$$L^T(\Theta) = \frac{1}{D}\sum_{d=1}^{D} ||\hat{\mathbf{x}}^T - \mathbf{x}^{(d)}||_2^2, \tag{2.104}$$

where $\hat{\mathbf{x}}^T$ is the output of the T-layers network with input $\mathbf{y}^{(d)}$ and parameters Θ. Simulations in [18] shows that LISTA converges faster than ISTA in general.

2.5.5 Adaptive LISTA

The main drawback of LISTA compared to ISTA is that, once it is trained over a given $\mathbf{H}$, its performance degrades when it is required to make predictions on different matrices. So, a new full training is necessary if $\mathbf{H}$ evolves with time.

Adaptive LISTA (Ada-LISTA) is proposed in [2] to combine the efficiency and the fast convergence rate of LISTA with the high adaptivity and applicability of ISTA.

Scheme 4 (Ada-LISTA - single weight matrix)

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \alpha^t \mathbf{H}^T\mathbf{W}^T\mathbf{r}^t \tag{2.105}$$

$$\hat{\mathbf{x}}^{t+1} = \eta_{ST}(\mathbf{z}^t, \lambda^t) \tag{2.106}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} \tag{2.107}$$

where $\Theta = \{\lambda^t, \alpha^t, \mathbf{W}\}$ are free parameters to be learned in a supervised learning fashion. Given a dataset $\{(\mathbf{y}^{(d)}, \mathbf{x}^{(d)}, \mathbf{H}^{(d)})\}_{d=1}^{D}$, $\mathbf{y}^{(d)} \in \mathbb{R}^{N_r}, \mathbf{x}^{(d)} \in$

$\mathbb{R}^{N_t}, \mathbf{H}^{(d)} \in \mathbb{R}^{N_r} \to \mathbb{R}^{N_t}$ the learnable parameters are optimized through backpropagation similarly to LISTA, as described in Section 2.5.4.

The experiments in [2] show that a slightly modified version of Ada-LISTA, defined in 4, with only one training matrix, performs similarly or even better than LISTA when the latter is trained for each new realization of $\mathbf{H}$, both in the case of noisy $\mathbf{H}$ and random i.i.d. Gaussian $\mathbf{H}$.

In the same work, the authors provide a theorem that provides the sufficient conditions for the convergence of the algorithm (specially on the structure of the matrix $\mathbf{W}$).

2.5.6 Learned AMP

Motivated by the AMP algorithm, the authors of [8] propose the use of Onsager correction in deep neural networks to decouples prediction errors across layers. They propose to unfold AMP and construct a neural network with tunable parameters Θ that are learned in a supervised fashion similarly to LISTA in Section 2.5.4. They call this network Learned AMP (LAMP). By starting from AMP in Definition 2, let's generalize $\mathbf{H}, \mathbf{H}^T$ with two learnabel matrices $\mathbf{H}^t, \mathbf{A}^t$:

$$\mathbf{r}^t = \mathbf{y} - \mathbf{H}^t\hat{\mathbf{x}}^t + b^t\mathbf{r}^{t-1}, \tag{2.108}$$

$$\hat{\mathbf{x}}^{t+1} = \eta_{ST}(\hat{\mathbf{x}}^t + \mathbf{A}^t\mathbf{r}^t, \lambda^t), \tag{2.109}$$

and by substituting λ^t for the soft shrinkage thresholding function (2.58) and assuming $\mathbf{H}^t = \beta^t\mathbf{H}$:

$$\mathbf{r}^t = \mathbf{y} - \beta^t\mathbf{H}\hat{\mathbf{x}}^t + b^t\mathbf{r}^{t-1}, \tag{2.110}$$

$$\hat{\mathbf{x}}^{t+1} = \eta_{ST}(\hat{\mathbf{x}}^t + \mathbf{A}^t\mathbf{r}^t, \frac{\theta}{\sqrt{N_r}}||\mathbf{r}^t||_2). \tag{2.111}$$

Let's now define $\overline{\mathbf{x}}^t = \beta^t\hat{\mathbf{x}}^t$ and $\overline{\mathbf{A}}^t = \beta\mathbf{A}^t$ and we can write

$$\mathbf{r}^t = \mathbf{y} - \mathbf{H}\overline{\mathbf{x}}^t + b^t\mathbf{r}^{t-1}, \tag{2.112}$$

$$\overline{\mathbf{x}}^{t+1} = \beta^{t+1}\eta_{ST}(\frac{\overline{\mathbf{x}}^t + \overline{\mathbf{A}}^t\mathbf{r}^t}{\beta^t}, \frac{\theta}{\sqrt{N_r}}||\mathbf{r}^t||_2). \tag{2.113}$$

By observing that $\eta_{ST}(\mathbf{z}, \lambda) = \frac{\eta_{ST}(\beta z, \beta\lambda)}{\beta}$ for any $\beta > 0$, and by defining $\overline{\beta} \triangleq \frac{\beta^{t+1}}{\beta^t}, \overline{\theta}^t \triangleq \beta^t\theta$:

$$\overline{\mathbf{x}}^{t+1} = \frac{\beta^{t+1}}{\beta}\eta_{ST}(\overline{\mathbf{x}}^t + \overline{\mathbf{A}}^t\mathbf{r}^t, \frac{\beta^t\theta}{\sqrt{N_r}}||\mathbf{r}^t||_2) = \overline{\beta}^t\eta_{ST}(\overline{\mathbf{x}}^t + \overline{\mathbf{A}}^t\mathbf{r}^t, \frac{\overline{\theta}^t}{\sqrt{N_r}}||\mathbf{r}^t||_2). \tag{2.114}$$

Recalling the definition of b^t in (2.57):

$$\mathbf{r}^t = \mathbf{y} - \mathbf{H}\overline{\mathbf{x}}^t + \frac{\overline{\beta}^{t-1}}{N_r}||\overline{\mathbf{x}}^t||_0\mathbf{r}^{t-1}. \tag{2.115}$$

Now we drop the $(\bar{\cdot})$ and provide the definition of LAMP:

Scheme 5 (LAMP)

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \mathbf{A}^t\mathbf{r}^t \tag{2.116}$$

$$\hat{\mathbf{x}}^{t+1} = \beta^t\eta_{ST}(\mathbf{z}_t, \frac{\theta^t}{\sqrt{M}}||\mathbf{r}^t||_2) \tag{2.117}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} + \frac{\beta^t}{N_r}||\hat{\mathbf{x}}^{t+1}||_0\mathbf{r}^t \tag{2.118}$$

In the Scheme 5, the parameters to learn are $\Theta = \{\mathbf{A}^t, \theta^t, \beta^t\}_{t=0}^{T-1}$. The training strategy for LAMP is different from LISTA. In [9] the authors explained that they tried the standard back-propagation approach (as in LISTA), where Θ were jointly optimized from the initialization $\mathbf{A}^t = \mathbf{H}^T, \theta^t = 1, \beta^t = 1$, where $t = 0, ..., T-1$. However, they found out that the parameters converged to a bad local minimum. This unwanted behavior is due to the vanishing gradient problem, explained in Section 2.5.2. Thus, they propose a hybrid approach that combine a "layer-wise" and "global" optimization that helps to avoid this problem. They learn Θ by learning each layer one by one, starting from a 1-layer network $t = 0$ where

$$\mathbf{A}^0 = \gamma^{-1}\mathbf{H}^T(\mathbf{H}\mathbf{H}^T + \mathbf{I}_{N_r})^{-1}, \tag{2.119}$$

and γ ensures that the trace of $\mathbf{H}\mathbf{A}^0$ is equal to N_t and $\theta_0,\ \beta_0$ are optimized trough backpropagation by minimizing the loss function $L^1(\{\theta^0, \beta^0\})$ as defined in (2.104). For the next layers, $t = 1, ..., T-1$ the training procedure is the following:

1. initialize $\mathbf{A}^t$ as in 2.119 and θ^t, β^t with the same values as $\theta^{t-1}, \beta^{t-1}$;

2. optimize θ^t, β^t alone using backpropagation;

3. re-optimize all parameters $\{\mathbf{A}^i, \theta^i, \beta^i\}_{i=0}^t$ using backpropagation to minimize the loss $L^{t+1}(\{\mathbf{A}^i, \theta^i, \beta^i\}_{i=0}^t)$ as defined in (2.104).

Numerical experiments in [8] show that LAMP network significantly improves LISTA network in both accuracy and complexity. In the same work, the authors compare LAMP with $\mathbf{B}^t$ shared among the layers and $\mathbf{B}^t$ that can vary layer by layer.

2.6 Deep learning for MIMO detection

2.6.1 MMNet

MMNet [24] is a DL MIMO detection scheme that aims to achieve a practical trade-off between computational complexity and SER performance both on i.i.d. Gaussian channels and spatially-correlated channels. MMNet is inspired by ISTA and AMP frameworks presented in the previous sections. It preserves the overall structure of AMP scheme in Definition 3, except for the Onsager term that here is missing. At the same time, it introduces in the model a degree of flexibility, that combined with an online training algorithm, allows to outperforms the state-of-the-art detection algorithms on realistic spatially correlated channels. The degree of flexibility is determined by the learnable parameters in the model that differ according to the type of channel we are working with, by leading to a common scheme with different learnable parameters for i.i.d. Gaussian channels and spatially-correlated channels.

Given the MIMO detection problem defined in Definition 3, MMNet estimates $\hat{\mathbf{x}}^T$, after T iterations with the following scheme:

Scheme 6 (MMNet)

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \mathbf{A}^t\mathbf{r}^t \tag{2.120}$$

$$(\boldsymbol{\sigma}_t)^2 = \frac{\boldsymbol{\theta}^t}{N_t}\left(\frac{||\mathbf{I} - \mathbf{A}^t\mathbf{H}||_F^2}{||\mathbf{H}||_F^2}[||\mathbf{r}^t||_2^2 - N_r\sigma^2]_+ + ||\mathbf{A}^t||_F^2\sigma^2\right) \tag{2.121}$$

$$\hat{\mathbf{x}}^{t+1} = F(\mathbf{z}^t, (\boldsymbol{\sigma}_t)^2) \tag{2.122}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} \tag{2.123}$$

In the previous Scheme 6, the starting values are $\hat{\mathbf{x}}^0 = 0, \mathbf{r}^0 = \mathbf{y}$ and $F(\mathbf{z}, \sigma)$ is the element-wise Gaussian denoiser defined in (2.71). In MMNet the learnable parameters Θ are $\mathbf{A}^t \in \mathbb{R}^{N_t} \times \mathbb{R}^{N_r}$ and $\boldsymbol{\theta}^t \in \mathbb{R}^{N_t}$ that assume different values in each iteration. MMNet concatenates T layers of the above form by leading to a deep network that is trained in a supervised fashion like LISTA (Section 2.5.4). In this case, the loss function to minimize is the following

$$L(\Theta) = \frac{1}{T}\sum_{t=1}^{T}||\hat{\mathbf{x}}^t - \mathbf{x}||_2^2, \tag{2.124}$$

where $\mathbf{x}$ is the true solution for the given problem.

MMNet for i.i.d. Gaussian channels

For i.i.d. Gaussian channels we have

$$\mathbf{A}^t = \alpha^t\mathbf{H}^T,\ \alpha^t \in \mathbb{R}, \tag{2.125}$$

and the denoiser variance $\boldsymbol{\sigma}^t$ has the same value in all the entries of the vector

$$\sigma_0^t = \sigma_1^t = ... = \sigma_{N_t}^t, \tag{2.126}$$

with

$$\theta_0^t = \theta_1^t = ... = \theta_{N_t}^t. \tag{2.127}$$

Here the supervised learning is done over a training dataset

$$\{(\mathbf{y}^{(d)}, \mathbf{x}^{(d)}, \sigma^{(d)}, \mathbf{H}^{(d)})\}_{d=1}^{D},$$

where $\mathbf{H}^{(d)}$ is randomly sampled from the i.i.d. Gaussian channel model. The training procedure optimizes the learnable parameters for different realizations of $\mathbf{H}$ and then it is tested over unseen realizations of $\mathbf{H}$ drawn from the same distribution.

MMNet for spatially correlated channels

For spatially correlated channels, MMNet doesn't impose any structure on the matrix $\mathbf{A}^t$ which is fully learnable and different in each layer. Here the supervised learning is done over a training dataset $\{(\mathbf{y}^{(d)}, \mathbf{x}^{(d)}, \sigma^{(d)})\}_{d=1}^{D} \cup \mathbf{H}$, where $\mathbf{H}$ is fixed. This solution is designed for realistic scenarios, however the complexity of the online training introduce a latency which makes MMNet far from practical implementation.

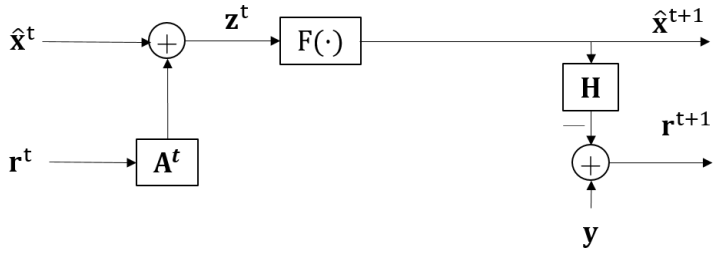


Figure 2.9: Layer t of MMNet. This graphical representation does not include the computation of $(\boldsymbol{\sigma}_t)^2$.

Chapter 3

Algorithms design

3.1 BP-MMSE

The BP algorithm provided in [16] for solving MIMO detection in a MRF does not significantly improve over MMSE performance when working with higher modulation order such as QAM-16 or QAM-64. In this section we introduce BP-MMSE, a novel algorithm based on [16]. BP-MMSE applies a modification of BP on the MRF that yields the MMSE prior information on each variable.

In order to derive the MMSE formula for the initial prior, let's consider the posterior probability $p(\mathbf{x}|\mathbf{y})$ in (2.22). Here we approximate the exact probability function with

$$p(\mathbf{x}|\mathbf{y}) \propto \exp(-\frac{1}{2\sigma^2}||\mathbf{y}-\mathbf{H}\mathbf{x}||^2), \tag{3.1}$$

that can be rewritten as a multivariate Gaussian distribution $f(\mathbf{x};\mathbf{z},\mathbf{C})$:

$$p(\mathbf{x}|\mathbf{y}) = f(\mathbf{x};\mathbf{z},\mathbf{C}) = \frac{1}{\sqrt{(2\pi)^{N_t}|\mathbf{C}|}} \exp\left(-\frac{1}{2}(\mathbf{z}-\mathbf{x})^T\mathbf{C}^{-1}(\mathbf{z}-\mathbf{x})\right), \tag{3.2}$$

where the mean $\mathbf{z}$ is the least square error estimator

$$\mathbf{z} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}, \tag{3.3}$$

and $\mathbf{C}$ the relative covariance matrix

$$\mathbf{C} = \sigma^2(\mathbf{H}^T\mathbf{H})^{-1}. \tag{3.4}$$

The next step is to follow the suggestion in [17] to further approximate $f(\mathbf{x};\mathbf{z},\mathbf{C})$ with the product of marginals of Gaussian densities $f(x_l;z_l,c_{ll})$:

$$f(\mathbf{x};\mathbf{z},\mathbf{C}) \approx \prod_{l=0}^{N_t-1} f(x_l;z_l,c_{ll}) = \prod_{l=0}^{N_t-1} \frac{1}{\sqrt{2\pi c_{ll}}} \exp\left(-\frac{(z_l-x_l)^2}{2c_{ll}}\right). \tag{3.5}$$

The MMSE approach (Definition 6) is known to work better than the ZF (Definition 5), therefore we modify $\mathbf{z}$ and $\mathbf{C}$ such that

$$\mathbf{z}^{MMSE} = (\mathbf{H}^T\mathbf{H} + \sigma^2\mathbf{I}_{N_t})^{-1}\mathbf{H}^T\mathbf{y}, \tag{3.6}$$

and

$$\mathbf{C}^{MMSE} = \sigma^2(\mathbf{H}^T\mathbf{H} + \sigma^2\mathbf{I}_{N_t})^{-1}. \tag{3.7}$$

The initial prior $p_l^0(x_l)$ for the variable node l is

$$p_l^0(x_l) = f(x_l; z_l^{MMSE}, c_{ll}^{MMSE}). \tag{3.8}$$

Similarly to Section 2.2.4, the messages $\log m_{i\to j}^0(x_j)$ are initialized with the logarithm of the initial prior $p_j^0(x_j)$

$$\log m_{i\to j}^0(x_j) \propto \log p_j^0(x_j) \propto -\frac{(z_j^{MMSE} - x_j)^2}{2c_{ll}^{MMSE}}. \tag{3.9}$$

For next iterations $t > 0$, the log messages $\log m_{i\to j}^t(x_j)$ are given by

$$\log m_{i\to j}^t(x_j) \propto \log \sum_{x_i} \exp(a_{i\to j}^t(x_j, x_i)), \tag{3.10}$$

where

$$a_{i\to j}^t(x_j, x_i) = \log p_i^t(x_i) + \log \phi_i(x_i) + \log \phi_{ij}(x_i, x_j) + \sum_{b\in ne(i)/j} \log m_{b\to i}^{t-1}(x_i), \tag{3.11}$$

and $p_i^t(x_i)$ is the prior at node i at iteration t.

We use some approximations to circumvent the numerical issues in practical implementation: the potential factors $\phi_i(x_i), \phi_{ij}(x_i, x_j)$ are slightly modified with respect to Section 2.2.4:

$$\phi_i(x_i) = e^{\frac{1}{\max(\sigma^2,\lambda)}(\mathbf{y}^T\mathbf{h}_i x_i - \frac{1}{2}\mathbf{h}_i^T\mathbf{h}_i x_i^2)}, \tag{3.12}$$

$$\phi_{ij}(x_i, x_j) = e^{-\frac{1}{\max(\sigma^2,\lambda)}\mathbf{h}_i^T\mathbf{h}_j x_i x_j}, \tag{3.13}$$

where λ is a parameter to tune.

To avoid further numerical issues at the exponential operator we subtract

$$a_{i\to j}^t(x_j)^* = \max_{x_i} a_{i\to j}^t(x_j, x_i) \tag{3.14}$$

from the argument of the exponential and add it back outside the summation:

$$\log m_{i\to j}^t(x_j) \propto a_{i\to j}^t(x_j)^* + \log \sum_{x_i} \exp(a_{i\to j}^t(x_j, x_i) - a_{i\to j}^t(x_j)^*). \tag{3.15}$$

In this way the biggest argument in the exponential will be equal to 0. This ensures that the dominant numerical contributions in the exponential are computed accurately.

The prior $p_l^{t+1}(x_l)$ at next iteration $t+1$ is equal to the updated probability at iteration t

$$p_l^{t+1}(x_l) = p_l^{t+1|t}(x_l)p_l^t(x_l) = p_l^0(x_l)\prod_{q=0}^{t} p_l^{q+1|q}, \tag{3.16}$$

and equivalently

$$\log p_l^{t+1}(x_l) = \log p_l^{t+1|t}(x_l) + \log p_l^t(x_l), \tag{3.17}$$

where

$$\log p_l^{t+1|t}(x_l) \propto \log \phi_l(x_l) + \sum_{b\in ne(l)} \log m_{b\to l}^{t-1}(x_l). \tag{3.18}$$

The predicted value $\hat{x}_l^{t+1}$ is the expected value of x_l with respect to $p_l^{t+1|t}(x_l)$:

$$\hat{x}_l^{t+1} = \mathbb{E}_{x_l}\{x_l\} = \frac{1}{Z}\sum_{x_l} x_l \exp(\log p_l^{t+1|t}(x_l) - a_l^t), \tag{3.19}$$

where Z is the normalization constant

$$Z = \sum_{x_l} \exp(\log p_l^{t+1|t}(x_l) - a_l^t), \tag{3.20}$$

and

$$a_l^t = \max_{x_l} \log p_l^{t+1|t}(x_l). \tag{3.21}$$

3.2 MIMO-GNN

The GNN framework presented in Section 2.5.3 can be used to infer the a posteriori probability $p(\mathbf{x}|\mathbf{y})$ to recover the transmitted symbols $\mathbf{x}$ in the MIMO detection problem in Definition 1. In this case, GNNs are built upon the MIMO MRF presented in Section 2.2.4. The information ϵ_{ij} along each edge e_{ij} is the following feature vector

$$\epsilon_{ij} = [-\mathbf{h}_i^T\mathbf{h}_j,\ \sigma^2]. \tag{3.22}$$

The initial hidden vector $\mathbf{u}_i^0$ of each node i is initialized as follows:

$$\mathbf{u}_i^0 = \mathbf{W}[\mathbf{y}^T\mathbf{h}_i,\ \mathbf{h}_i^T\mathbf{h}_i,\ \sigma^2] + \mathbf{b}. \tag{3.23}$$

Since we want to work with an hidden state of a given size S_u, to simplify the implementation we encode the initial vector $[\mathbf{y}^T\mathbf{h}_i,\ \mathbf{h}_i^T\mathbf{h}_i,\ \sigma^2]$ with a

linear transformation given by a learnable matrix $\mathbf{W} \in \mathbb{R}^{S_u} \times \mathbb{R}^3$ and a learnable vector $\mathbf{b} \in \mathbb{R}^{S_u}$. The functions M and R are chosen to be two neural networks with two hidden layers and ReLU as activation functions. Both M and R implement dropout between hidden layers (rate of 0.1 in M and rate of 0.2 in R). The first and the second hidden layer output sizes are l and $\frac{l}{2}$ parameters respectively. The read out network R is followed by a softmax function to output the predicted normalized density probability $\hat{p}(x_l)$ for each transmitted symbol x_l. Instead, the function U is defined by a GRU network followed by a linear layer that ensures that the output size is equal to S_u. In the experiments the dimension of the GRU hidden state is l.

The predicted value for a transmitted symbols $\hat{x}_l$ is the expected value of x_l with probability distribution $\hat{p}(x_l)$:

$$\hat{x}_l = \mathbb{E}_{x_l}\{x_l\} = \sum_{s \in \mathcal{A}} s\hat{p}(s). \tag{3.24}$$

The network is trained as explained in Section 4.3.4 and the hyperparameters design is provided in Section 3.4.

3.3 MIMO-GNN-MMSE

In the previous section we solve the MIMO detection problem under the uniform prior assumption over the unknown symbols $\mathbf{x}$. In this section we select the prior $p(\mathbf{x})$ as in Section 3.1 to enrich our dataset.

We choose $f(x_l; z_l^{MMSE}, c_{ll}^{MMSE})$, defined in(3.5), to be the prior $p_l(x_l)$ for x_l, where z_l^{MMSE} and c_{ll}^{MMSE} are defined in (3.6) and (3.7) respectively. Moreover, we enrich the feature vector ϵ_{ij} by including the correlation ρ_{ij}^{MMSE} between the variables x_i and x_j where ρ_{ij}^{MMSE} is

$$\rho_{ij}^{MMSE} = \frac{(c_{ij}^{MMSE})^2}{c_{ii}^{MMSE} c_{jj}^{MMSE}}. \tag{3.25}$$

In the implementation we reuse the same model in Section 3.2 (with the same hyperparameters) and we only modify the information along the edges ϵ_{ij} and the initial value of the hidden states $\mathbf{u}_i^0$.

The information along each edge e_{ij} is the following feature vector

$$\epsilon_{ij} = [\rho_{ij}^{MMSE},\ -\mathbf{h}_i^T \mathbf{h}_j,\ \sigma^2]. \tag{3.26}$$

The initial hidden vector $\mathbf{u}_i^0$ of each node i is initialized as follows:

$$\mathbf{u}_i^0 = \mathbf{W}[\mathbf{z}_i^{MMSE},\ c_{ii}^{MMSE},\ \mathbf{y}^T \mathbf{h}_i,\ \mathbf{h}_i^T \mathbf{h}_i,\ \sigma^2] + \mathbf{b}, \tag{3.27}$$

where $\mathbf{W} \in \mathbb{R}^{S_u} \times \mathbb{R}^5$ is a learnable matrix and $\mathbf{b} \in \mathbb{R}^{S_u}$ is a learnable vector.

3.3.1 Edge pruning

In GNNs the most computationally expensive operation is the execution of the function M for all the graph edges in each iteration. In MIMO detection the graph is fully connected, therefore, M is executed $TN_t(N_t - 1)$ times.

We propose a strategy to prune some of the edges in the graph in order to reduce the overall complexity by still keeping a good performance. Our approach is inspired by the pruning strategy presented in the work of Gaussian Tree Approximation for MIMO detection [17]. Here, the authors approximate the MRF graph with the maximum spanning tree over the same graph where each edge is weighted with the correlation coefficient ρ_{ij}^{MMSE}.

In this work, for each node j we keep only the incoming edges e_{ij} that satisfy the following condition:

$$\rho_{ij}^{MMSE} \geq \frac{1}{N_t - 1} \sum_{b=0, b \neq j}^{N_t - 1} \rho_{bj}^{MMSE}. \tag{3.28}$$

On one hand, the edge pruned variant of GNNs with MMSE prior, that we call MIMO-GNN-SPARSE, keeps the same initial value of the hidden states $\mathbf{u}_i^0$ defined for the fully connected graph in MIMO-GNN-MMSE. On the other hand, the information along each edge e_{ij} is modified in order to keep into account the number of incoming edges in j that can vary from node to node:

$$\epsilon_{ij} = [\rho_{ij}^{MMSE},\ -\frac{1}{2}\mathbf{h}_i^T\mathbf{h}_j,\ \sigma^2,\ |ne(j)|], \tag{3.29}$$

where $|ne(j)|$ is the number of incoming edges in j in the pruned graph.

3.4 GNNs hyperparameters design

An hyperparameter is a parameter whose value is used to control the learning process and have to be tuned such that the model can achieve its best performance. How to properly tune the hyperparameters is discussed in Section 2.5.1.

In GNNs, the hyperparameters are:

- the number of hidden layers in M and R;
- the number of parameters in each layer of M and R and in U;
- the size of the messages S_m;
- the size of the hidden states S_u;
- the number of iterations T;
- the learning rate;

- the batch size;
- the dropout rate;
- the number of training epochs.

In solving MIMO detection, S_m and S_u need to be readapted for each modulation order. Moreover, the number of parameters in M and U may change according to the presence or absence of an informative prior. Indeed, the process of learning can be more complex without the informative prior.

In this work, we choose to set the dimension S_m of the messages $\mathbf{m}_{i\to j}^t$ as the double of the constellation modulation size (or similarly as the double of the messages dimension in BP). We set the hidden state size S_u equal to S_m.

In the next table you can find the values of S_m and S_u for each constellation size:

Table 3.1: Values of S_m and S_u for each modulation order.

QAM	S_m	S_u
16	8	8
64	16	16

In Section 3.2 we indicate how to control the number of parameters in M, R, U (and therefore the complexity of the model) with an hyperparameter l. In the experiments $l = 128$ results to be a good pick in general. However, in MIMO-GNN working at modulation QAM-64, l must be increased to enable the learning. In our experiments l is set to 256.

3.5 DAMP

DAMP stands for Damping Approximate Message Passing. It's a low complexity MIMO detection scheme designed for i.i.d. Gaussian channels but intended to be robust to low/medium level of correlation among channels. It can been seen as a generalization of the AMP-Gaussian (Definition 7) algorithm where the damping operator is introduced at the end of each layer.

In iterative algorithms damping consists in attenuating the predictions at iteration t with the values at the previous iteration $t-1$ through a weighted average parametrized by the damping factor ρ. In AMP damping is applied to predictions $\tilde{\mathbf{x}}^{t+1}$ in the following way:

$$\hat{\mathbf{x}}^{t+1} = \rho\tilde{\mathbf{x}}^{t+1} + (1-\rho)\hat{\mathbf{x}}_t. \tag{3.30}$$

In Definition 8 we present the DAMP algorithm and how we output predictions $\tilde{\mathbf{x}}^{t+1}$ at each iteration t.

Definition 8 (DAMP) *DAMP is an iterative algorithm. At the end of each algorithm iteration t, the layer t exchanges the predicted output $\hat{\mathbf{x}}_t$ with next iteration. Firslty, we initialize $\hat{\mathbf{x}_0} = \mathbb{E}_X[X] = 0$, $\mathbf{r}_0 = \mathbf{y}$, $\tau_0 = \frac{\mathbb{VAR}_X[X]=1}{\sigma^2 N_r}$. Then for every iteration $t = 0, 1, 2..., T-1$ we compute the following steps:*

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \mathbf{H}^T \mathbf{r}^t \tag{3.31}$$

$$\tilde{\mathbf{x}}^{t+1} = F(\mathbf{z}^t, \max(\sigma^2, \lambda)(1+\tau^t)) \tag{3.32}$$

$$\hat{\mathbf{x}}^{t+1} = \rho \tilde{\mathbf{x}}^{t+1} + (1-\rho)\hat{\mathbf{x}}_t \tag{3.33}$$

$$\tau^{t+1} = \frac{1}{\max(\sigma^2, \lambda) N_r} \langle G(\mathbf{z}^t, \max(\sigma^2, \lambda)(1+\tau^t)) \rangle \tag{3.34}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} + \frac{\tau^{t+1}}{\delta(1+\tau^t)} \mathbf{r}^t, \tag{3.35}$$

where δ, λ and the damping factor ρ are parameters to tune and $F(z, \tau))$ is the Gaussian denoiser defined in (2.71) *and $G(z, \tau)$ the function defined in* (2.73)*. Both $F(z, \tau)$ and $G(z, \tau)$ operate element-wise on vectors.*

In [8], it has been shown that AMP converges much faster than ISTA because of the presence of the Onsager correction term. However, in MMNet work [24] they manage to achieve faster convergence by removing the Onsager term and letting the stepsize α_t to vary from layer to layer.

DAMP, instead, reintroduces the Onsager term. With a larger number of iterations, DAMP ensures robustness in presence of low/medium level of correlation in the channel.

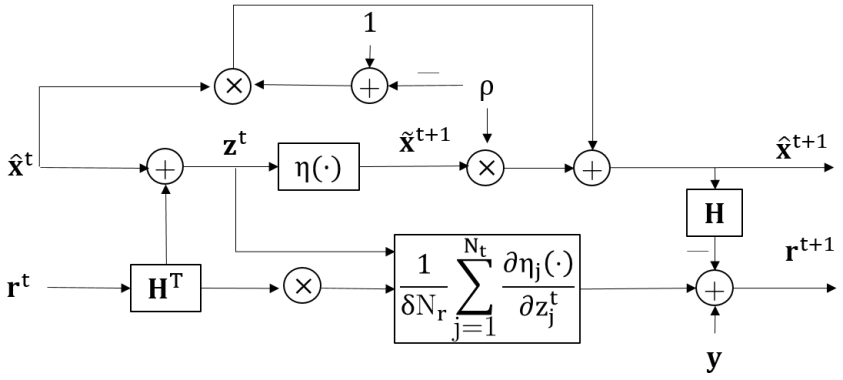


Figure 3.1: Iteration t of DAMP. This graphical representation does not include the computation of τ^{t+1}.

3.6 Pseudo-MMNet

Pseudo-MMNet is a deep learning MIMO detection scheme based on MMNet that solves the difficulties of DAMP when it works with strongly correlated

channels at the expense of performing an online training. Online training consists in training the network every time we observe a new channel realization. The sequential online training complexity of MMNet introduces a latency that cannot be afforded in realistic implementations. In MMNet, the authors don't provide any informative initialization to the parameters to learn. Therefore, the training requires several epochs. Psuedo-MMNet solves this problem and moreover it reduces the number of parameters to learn.

Definition 9 (Pseudo-MMNet) *Pseudo-MMNet is the concatenation of T layers. At the end of each algorithm iteration t, the layer t exchanges the predicted output $\hat{\mathbf{x}}_t$ with next layer. Given $\hat{\mathbf{x}}_0 = 0$ and $\mathbf{A}^t = \alpha^t \mathbf{A}$, each layer is composed of a linear operator*

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \mathbf{A}^t(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^t) \tag{3.36}$$

and a denoiser operator

$$\hat{\mathbf{x}}^{t+1} = F(\mathbf{z}^t, \sigma^t) \tag{3.37}$$

where F is the Gaussian denoiser defined in (2.71) and σ_t^2 is the denoiser variance

$$(\boldsymbol{\sigma}^t)^2 = \frac{\theta^t}{N_t}\left(\frac{||\mathbf{I} - \mathbf{A}^t\mathbf{H}||_F^2}{||\mathbf{H}||_F^2}[||\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^t||_2^2 - N_r\sigma^2]_+ + ||\mathbf{A}^t||_F^2\sigma^2\right) \tag{3.38}$$

In Pseudo-MMNet the learnable parameters are the matrix $\mathbf{A} \in \mathbb{R}^{Nt} \times \mathbb{R}^{Nr}$ that is tied for each layer, $\alpha^t \in \mathbb{R}$ and $\theta^t \in \mathbb{R}^{Nt}$ that can assume different values in each iteration t. The matrix $\mathbf{A}$ is initialized with $\mathbf{H}^\dagger$, the pseudo inverse of $\mathbf{H}$.

The resulting deep network is trained in a supervised fashion like LISTA (Section 2.5.4). The training dataset is of the form $\{(\mathbf{y}^{(d)}, \mathbf{x}^{(d)}, \sigma^{(d)})\}_{d=1}^{D} \cup \mathbf{H}$, where $\mathbf{H}$ is fixed. The collection of parameters to learn is $\Theta = \{\alpha^t, \theta^t\}_{t=0}^{T-1} \cup \mathbf{A}$ and the loss function to minimize is $L^T(\Theta)$ as defined in (2.104).

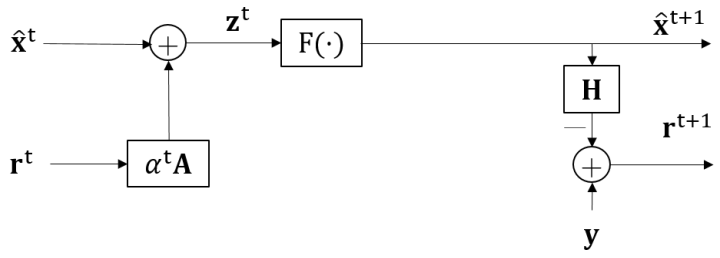


Figure 3.2: Layer t of Pseudo-MMNet. This graphical representation does not include the computation of $(\boldsymbol{\sigma}^t)^2$.

3.6.1 Why Pseudo-MMNet works?

In order to understand why Pseudo-MMNet works, let's consider the errors $\mathbf{e}_{lin}^t = \mathbf{z}^t - \mathbf{x}$ and $\mathbf{e}_{den}^t = \hat{\mathbf{x}}^{t+1} - \mathbf{x}$ at the outputs of the linear and denoiser stages respectively, at iteration t. Then, we substitute (2.1) in (3.36) and we obtain:

$$\mathbf{z}^t = \mathbf{x} + \mathbf{e}_{lin}^t = \mathbf{x} + (\mathbf{I} - \mathbf{A}^t\mathbf{H})\mathbf{e}_{den}^{t-1} + \mathbf{A}^t\mathbf{n}. \tag{3.39}$$

If $\mathbf{A}^t = \mathbf{H}^\dagger$ and $\mathbf{H}$ has linearly independent columns it happens that

$$\mathbf{z}^t = \mathbf{x} + \mathbf{H}^\dagger\mathbf{n}. \tag{3.40}$$

However, there is no guarantee that $\mathbf{H}$ has linearly independent columns. If $\mathbf{H}$ is ill-conditioned, we incur in the problem of noise enhancement. In order to avoid this problem, Pseudo-MMNet initializes $\mathbf{A}^t = \mathbf{H}^\dagger$ but it relies on an online training of $\mathbf{A}^t$ to:

- reduce the error $\mathbf{e}_{lin}^t$ by learning the right balance between the two error terms in (3.39);
- shape the distribution of $\mathbf{e}_{lin}^t$ to make it suitable for the subsequent denoiser F.

In MMNet [24] and LISTA [18] simulations, it is empirically shown that by learning a linear operator for each layer it is possible to achieve great results. Learning those operators requires a lot of training that actually can be avoided. Instead, in Ada-LISTA [2] the authors analytically show that it is sufficient to learn a single matrix shared across the layers to solve the sparse linear inverse problem. Moreover, this matrix has similar properties of the pseudo-inverse $\mathbf{H}^\dagger$ of $\mathbf{H}$. More details can be found in [2].

3.6.2 Why is online training required?

In order to avoid the complexity of an online training we removed $\mathbf{A}$ from the parameters to learn and set $\mathbf{A} = \mathbf{H}^\dagger$. We trained the algorithm offline similarly to MMNet-iid (Section 2.6.1) over i.i.d. randomly sampled Gaussian channel realizations. However, it turned out that the resulting algorithm got stuck at the local minima solution of ZF. Indeed, the first layer of this alternative solution is equivalent to the zero-forcing solution (when $\alpha^t = 1$).

3.7 Comparison among schemes

In order to have a better understanding of the relation among AMP, ISTA, LAMA, AMP-Gaussian, DAMP, LAMP, Ada-LISTA, MMNet-iid, MMNet and Pseudo-MMNet, we provide a common general scheme for the algorithms and a table to show analogies and differences among them. Here,

LISTA is not included in the comparison because its scheme cannot be seen as a specialization of the following general form. Here, we don't provide a comparison for the term $\boldsymbol{\sigma}^t$. However, the reader can find the formula of $\boldsymbol{\sigma}^t$ along with the definition of each algorithm.

Scheme 7 (General Scheme)

$$\mathbf{z}^t = \hat{\mathbf{x}}^t + \alpha^t \mathbf{A}^t \mathbf{r}^t \tag{3.41}$$

$$\tilde{\mathbf{x}}^{t+1} = \beta^t \eta(\mathbf{z}^t, (\boldsymbol{\sigma}^t)^2) \tag{3.42}$$

$$\hat{\mathbf{x}}^{t+1} = \rho^t \tilde{\mathbf{x}}^{t+1} + (1-\rho^t)\hat{\mathbf{x}}_t \tag{3.43}$$

$$\mathbf{r}^{t+1} = \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}^{t+1} + \frac{1}{\delta^t N_r} \sum_{j=0}^{N_t-1} \frac{\partial [\eta(\mathbf{z}^t, \boldsymbol{\sigma}^t)]_j}{\partial z_j^t} \mathbf{r}^t \tag{3.44}$$

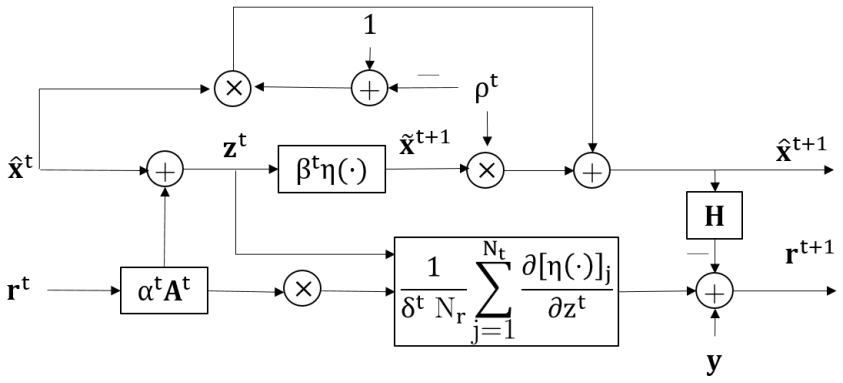


Figure 3.3: Layer t of the general scheme of low-complexity iterative algorithm.

Table 3.2: Scheme comparison between classical iterative algorithms.

	AMP	ISTA	LAMA	AMP-Gaussian	DAMP
α^t	1	α	1	1	1
$\mathbf{A}^t$	$\mathbf{H}^T$	$\mathbf{H}^T$	$\mathbf{H}^T$	$\mathbf{H}^T$	$\mathbf{H}^T$
β^t	β^t	1	1	1	1
η	η_{ST}	η_{ST}	F^*	F	F
ρ^t	1	1	1	1	ρ
$\frac{1}{\delta^t}$	1	0	1	1	$\frac{1}{\delta}$

The first thing to notice is that AMP is equivalent to ISTA when we remove the Onsager correction ($\frac{1}{\delta}$) and $\alpha = 1$. As we explained in 2.3.2 the Onsager correction speeds up the convergence of the algorithm. Therefore, the main idea behind MMNet-iid is to modify ISTA by learning a different α^t for each layer. In this way, it is possible to achieve faster convergence

Table 3.3: Scheme comparison between NN-based iterative algorithms.

	LAMP	Ada-LISTA	MMNet-iid	MMNet	Pseudo-MMNet
α^t	1	α^t	α^t	1	α^t
$\mathbf{A}^t$	$\mathbf{A}^t$	$\mathbf{H}^T\mathbf{W}^T$	$\mathbf{H}^T$	$\mathbf{A}^t$	$\mathbf{A}$
β^t	β^t	1	1	1	1
η	η_{ST}	η_{ST}	F	F	F
ρ^t	1	1	1	1	1
$\frac{1}{\delta^t}$	β^t	0	0	0	0

even without the presence of Onsager correction. Moreover, MMNet-iid is optimized for MIMO detection because the non linear function η is the Gaussian denoiser F designed to include the prior knowledge over the transmitted symbols.

Another approach is LAMA [22], that re-adapts AMP for complex-valued MIMO systems by substituting in AMP the soft thresholding function η_{ST} with the Gaussian denoiser F^* for complex values (the definition of F^* is provided in [22] as F). In this work we provide an equivalent alternative to LAMA for real-valued systems. We name the algorithm AMP-Gaussian and the main differences with LAMA are the Gaussian denoiser F for real values and the presence of a correction factor $\frac{1}{\delta}$ in front of the Onsager term. Then we have DAMP, a generalization of AMP-Gaussian that applies damping at the end of each iteration.

To conclude the analysis we observe that Ada-LISTA, MMNet and Pseudo-MMNet generalize ISTA by substituting the matrix $\mathbf{H}^T$ in front of the residual $\mathbf{r}^t$ with a learnable matrix. In MMNet the matrix $\mathbf{A}^t$ is allowed to change layer by layer while in Ada-LISTA and Pseudo-MMNet the learnable matrix is tied for each layer. Ada-LISTA works with the soft thresholding function η_{ST} because is designed to solve the Sparse Linear Inverse Problem. Instead, Pseudo-MMNet applies the Gaussian denoiser F for solving MIMO detection.

In this work we tried to learn $\alpha^t, \rho^t, \delta^t, \theta^t$ similarly to MMNet-iid (Section 6), where σ^t is computed as in 2.4 given $\beta^t = 1, \mathbf{A}^t = \mathbf{H}^t$. In the experiments we don't provide the testing results of this attempt because by comparing the performance with DAMP, where the parameters ρ, δ are tuned manually, we can see that:

- on i.i.d. Gaussian channels the performance slightly improves at low SNR values but it does not for high SNR values;
- on correlated channels the performance degrades significantly because the parameters overfit to the training dataset which is generated with i.i.d. Gaussian channels.

Chapter 4

Methodology

Our work is based on a deductive approach: it evaluates the hypothesis and the solution's performance by using experimental measurements and comparisons. The scientific method involves an empirical investigation It compares the proposed approaches to other baseline solutions using synthetic data and metrics that allow to quantify the performances. The outcomes of the experiments are based on the data generated by our simulators.

4.1 Compared algorithms

In the next experiments, we consider the the following algorithms:

- **MMSE**: linear minimum mean square error baseline detector in Definition 6.
- **MMNet-iid**: MMNet network for i.i.d. Gaussian channels described in Section 2.6.1. The network is implemented with 10 layers as proposed in [24] and trained as outlined in Section 4.3.1.
- **AMP-Gaussian**: AMP algorithm for real-valued system MIMO detection (Section 2.4). The parameter δ and λ are set manually as $\frac{1}{\delta} = 1.4$ and $\lambda = 0.005$.
- **DAMP-ρ**: if damping factor $\rho = 1$, it is equivalent to **AMP-Gaussian**. It is the algorithm described in Section 3.5.
- **Pseudo-MMNet**: Pseudo-MMNet algorithm defined in Definition 9. The network is trained as outlined in Section 4.3.3.
- **MMNet**: MMNet network for spatially correlated channels described in Section 2.6.1. The network is implemented with 10 layers and trained with the procedure presented in Section 4.3.2.

- **BP-MMSE**: LBP algorithm with MMSE prior presented in Section 3.1. The algorithm runs for 5 iterations. Increasing the number of iterations does not help to improve the performance.

- **MIMO-GNN**: GNNs presented in Section 3.2 without any prior on the transmitted symbols. The network implements 10 layers.

- **MIMO-GNN-MMSE**: GNNs presented in Section 3.3 with MMSE prior on the transmitted symbols. The network implements 10 layers.

- **MIMO-GNN-SPARSE**: edge pruned variant of **GNN-MMSE**, presented in Section 3.3.1. The network implements 10 layers.

4.2 Dataset

In all comparisons, the data is generated through the MIMO model in Definition 1 by following the same approach as in [24] and according to the assumptions listed in Definition 1. We consider the channel models presented in Section 2.1.1: i.i.d. Gaussian channel model and Kronecker channel model. The algorithms are tested on randomly sampled i.i.d. Gaussian channels and randomly sampled Kronecker channels with different correlation levels at receiver side. We start our analysis by considering correlation only at receiver side because we assume that the scheduler at base station selects the users that reduce the correlations. Furthermore, we also carry out investigation of taking into account correlation at transmitter side.

The data is converted in the equivalent real-valued representation as explained in Section 2.1.2. On one hand, we analyze the MIMO configuration with 32 transmitters ($N_t = 64$) and 64 receivers ($N_r = 128$) for the low complexity iterative algorithms. On the other hand, GNN-based algorithms are analyzed on two MIMO configurations with $N_t = 32$ and system size ratios $\frac{N_t}{N_r}$ of 0.25 and 0.5. The choice of these configurations allows the comparison with previous works like [24] and, at the same time, they are meant for realistic massive MIMO scenarios. The modulation schemes considered in this work are QAM-16 and QAM-64 and it is fixed for all the transmitters in each experiment.

The operating SNR interval depends on the MIMO configuration and modulation scheme. Here, it is selected such that the best scheme performance can achieve a SER between $10^{-2} - 10^{-3}$ in the experiment.

4.2.1 Offline training

During offline training, there are three sources of randomness for a sample: signal $\mathbf{x}$, channel noise $\mathbf{n}$ and channel matrix $\mathbf{H}$. In each sample the channel matrix $\mathbf{H}$ is sampled from the respective channel model. The channel noise

standard deviation σ is derived from the SNR value as explained in Definition 1 and the transmitted signal $\mathbf{x}$ is generated randomly and uniformly over the corresponding constellation set.

According to the MIMO model in Defintion 1, each batch assumes the following form $\{(\mathbf{y}^{(d)}, \mathbf{H}^{(d)}, \sigma^{(d)}, \mathbf{x}^{(d)})\}_{d=1}^{D}$ where $\mathbf{x}^{(d)}$ are the values to predict.

In this work, the algorithms are trained offline over channel matrices randomly sampled from the i.i.d. Gaussian channel model. The algorithms are then tested on the same channel model and the Kronecker channel model with same sources of randomness and different random seed.

4.2.2 Online training

Instead, in online training the channel matrix $\mathbf{H}$ is fixed and randomly sampled from the Kronecker channel model. By removing $\mathbf{H}$ from the sources of randomness, each batch size assumes the following form $\{(\mathbf{y}^{(d)}, \sigma^{(d)}, \mathbf{x}^{(d)})\}_{d=1}^{D}$ $\cup\mathbf{H}$. The algorithms are then tested on the same channel matrix with same sources of randomness and different random seed.

4.2.3 GNN-based models

After generating a sample, for GNN-based algorithms we need to build a fully connected graph (Section 3.2). This operation is quite expensive in computational terms. Indeed, generating a different batch for each training step reduces substantially the speed of the training. Because of that, GNN-based models are trained on a prebuild dataset of size 65536 and batch size $D = 64$.

Training on a fixed dataset introduces the risk of overfitting. For this reason, we generate an additional validation dataset to find the best hyperparameters of the model and to apply the early stopping (Section 2.5.1). The size of the (additional) validation dataset is 25% of the training dataset size.

For GNNs, we decide to keep the noise standard deviation $\sigma^{(d)}$ fixed within each batch. In such a way, we notice an overall better performance during testing. If σ is allowed to assume different values within the same batch, the dominant contribution in the loss function comes from data generated with more noise (σ with high values). Therefore, the model is trained to work better for lower SNR than higher SNR.

Our GNNs implementation requires the output labels to be a discrete probability $p(\mathbf{x})$ over $\mathbf{x}$ while the labels $\mathbf{x}^{(d)}$ in our dataset are the true transmitted symbols. The best approach to obtain the labels would be to compute the true posterior probability in (2.22) $p(\mathbf{x}^{(d)}|\mathbf{y}^{(d)})$. However, the task is exponentially expensive and the time available for this work is limited.

Therefore, we opt fot one-hot encoded labels where:

$$p(x_l)^{(d)} = \begin{cases} 1, & \text{if } x_l = x_l^{(d)}. \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

4.2.4 Low-complexity iterative based models

Instead, for low-complexity iterative models, the batch is randomly generated before each training step. The complexity of generating a batch is negligible compared to the forward and backward training steps combined. Therefore, we can avoid to train the network on a fixed prebuild dataset and we nullify the risk of overfitting (that usually arises when the model is trained on a fixed dataset).

4.3 Training

All the trainable algorithms have been implemented and trained in Tensorflow 2.0.0 [1]. GNN-based algorithms are implemented with the Graph Nets open-source software library for building graph networks [4].

For each model, we run a separate training for every MIMO configuration and modulation order. We train GNN and AMP based algorithms in different ways. On one hand, **MMNet-iid**, **MMNet**, **Pseudo-MMNet** are trained with a random batch at each epoch. On the other hand, generating graph based batches is highly computationally expensive, therefore **MIMO-GNN**, **MIMO-GNN-MMSE** and **MIMO-GNN-SPARSE** are trained over a pre-build dataset at each epoch. This implementation choice introduces the risk of overfitting during training. Therefore, we manage the problem with the dropout technique and early stopping (Section 2.5.1).

In this section we describe the details of the training procedure for each algorithm.

4.3.1 MMNet-iid training procedure

MMNet-iid described in Section 2.6.1 has been trained in three different ways due to the fact that the original training strategy in [24] didn't work as expected. We list the three variants of the algorithm implemented in this work:

- **MMNet-iid-original**: the network is trained as explained in [24] for 10k epochs with Adam optimizer [25] and learning rate 0.001 to minimize the loss function presented in Section 2.6.1. Each training batch has 500 samples.

- **MMNet-iid**: the network is trained for a total of 1100 epochs with Adam optimizer [25] and learning rate of 0.005. Each training batch

has size 500. We optimize $\Theta^T = \{\alpha^t, \theta^t\}_{t=0}^{T-1}$ by learning layers one by one, starting from a 1-layer network when $t = 0$. The parameters are optimized trough backpropagation by minimizing the loss function $L_1(\Theta_1)$ as defined in (2.104). For the next layers, $t = 1, ..., T-1$ the training procedure is the following:

1. initialize α^t, θ^t with the values $\alpha^{t-1}, \theta^{t-1}$ at the previous layer;
2. optimize all parameters Θ^{t+1} using backpropagation for $e(t)$ epochs to minimize the loss $L^{t+1}(\Theta^{t+1})$ as defined in (2.104).

In our case $e(t) = 20t$. There, for 10 layers the total number of epochs is 1100.

- **MMNet-iid-extra**: firstly the network is trained in the same way as **MMNet-iid**. Then the network is additionally trained for 1000 additional epochs with Adam optimizer [25] and learning rate 0.001 to minimize the loss $L^{t-1}(\Theta^{t-1})$ as defined in (2.104). Each training batch has 500 samples.

4.3.2 MMNet training procedure

MMNet described in Section 2.6.1 has not been trained according to the original training strategy in [24]. By following the original procedure, the network was not able to outperform MMSE during testing. Therefore, the network is trained for 2000 epochs (instead of 1000) with the Adam optimizer [25] and learning rate of 0.001 to minimize the loss $L^T(\Theta^T)$ as defined in (2.104). Each training batch has a size 500. The main difference with the original approach lies in the choice of the loss function. Here, we consider only the error between the true signals $\mathbf{x}$ and the output of the last layer $\mathbf{x}^T$. Instead, in [24], the loss function takes into account the error at the output of each layer t.

4.3.3 Pseudo-MMNet

Pseudo-MMNet is trained for 25 epochs with Adam optimizer [25] and learning rate 0.005 to minimize the loss $L^T(\Theta^T)$ in (2.104). Each training batch 500 samples.

4.3.4 GNNs

Each GNN-based model is trained for 1000 epochs with Adam optimizer [25] and learning rate 0.0001 to minimize the loss L in (2.101). Each training batch has 64 samples.

After each epoch, we compute the average loss over the validation dataset: if the validation loss is less than the best one computed until the current

epoch, then the current model temporarily becomes the best one. The last saved temporary model is the one used for testing purposes.

Chapter 5

Experiments

In this chapter we analyze and compare the performance of the algorithm according to the SER metric (2.1.3). The results are always averaged over 10000 simulations for $N_t = 64$ and 20000 simulations for $N_t = 32$.

5.1 DAMP

5.1.1 Experiment 1

The experiment compares the **MMSE**, **AMP-Gaussian**, **DAMP-0.55**, **DAMP-0.75**, **DAMP-0.95** algorithms for the MIMO configuration of 32 transmitters ($N_t = 64$) and 64 ($N_r = 128$) receivers with modulation scheme QAM-64. We test the algorithms on i.i.d. Gaussian channels and Kronecker channel model with correlation coefficient $\rho_r = 0.3$ at receiver side. The algorithms are compared for different number of running iterations (10, 12, 14). In each simulation we sample a random channel matrix according to the chosen channel model. The goal of the experiment is to understand the effect of channel correlation on **AMP-Gaussian** and on the alternative with damping.

Figure 5.1 shows that when the channels are i.i.d. and Gaussian distributed, **AMP-Gaussian** with 14 iterations outperforms the **MMSE** baseline algorithm significantly (around 2.5dB gain over MMSE at SER of 10^{-2}). On the other hand, comparing Figure 5.1 and Figure 5.2, we observe that when the channel correlation at the receiver side increases to $\rho_R = 0.3$ the **AMP-Gaussian** performance degrades. Particularly, **AMP-Gaussian** SER performance degrades at high SNR values. However, comparing the DAMP curves, it can be observed that choosing the right damping factor can improve the performance at a high SNR regime.

In **DAMP-0.75** we register a degradation around 1dB at SER of 10^{-2} by lowering the number of iterations from 14 to 10 when channels are i.i.d. and Gaussian distributed. On Kronecker channel model with correlation $\rho_r = 0.3$ at receiver side, **DAMP-0.75** with 10 layers achieves only 0.5dB

gain over **MMSE**. Here we compare four different values of damping ρ_r and we empirically observe that the optimal one for our system is between 0.55 and 1. After several evaluations we decide to use DAMP $\rho = 0.75$ that runs for 14 iterations as a benchmark for the next experiments. Both on i.i.d. Gaussian channels and on Kronecker channel model with correlation $\rho_r = 0.3$ at receiver side, **DAMP-0.75** outperforms **MMSE** with a gap around 2.5dB at SER of 10^{-2}.

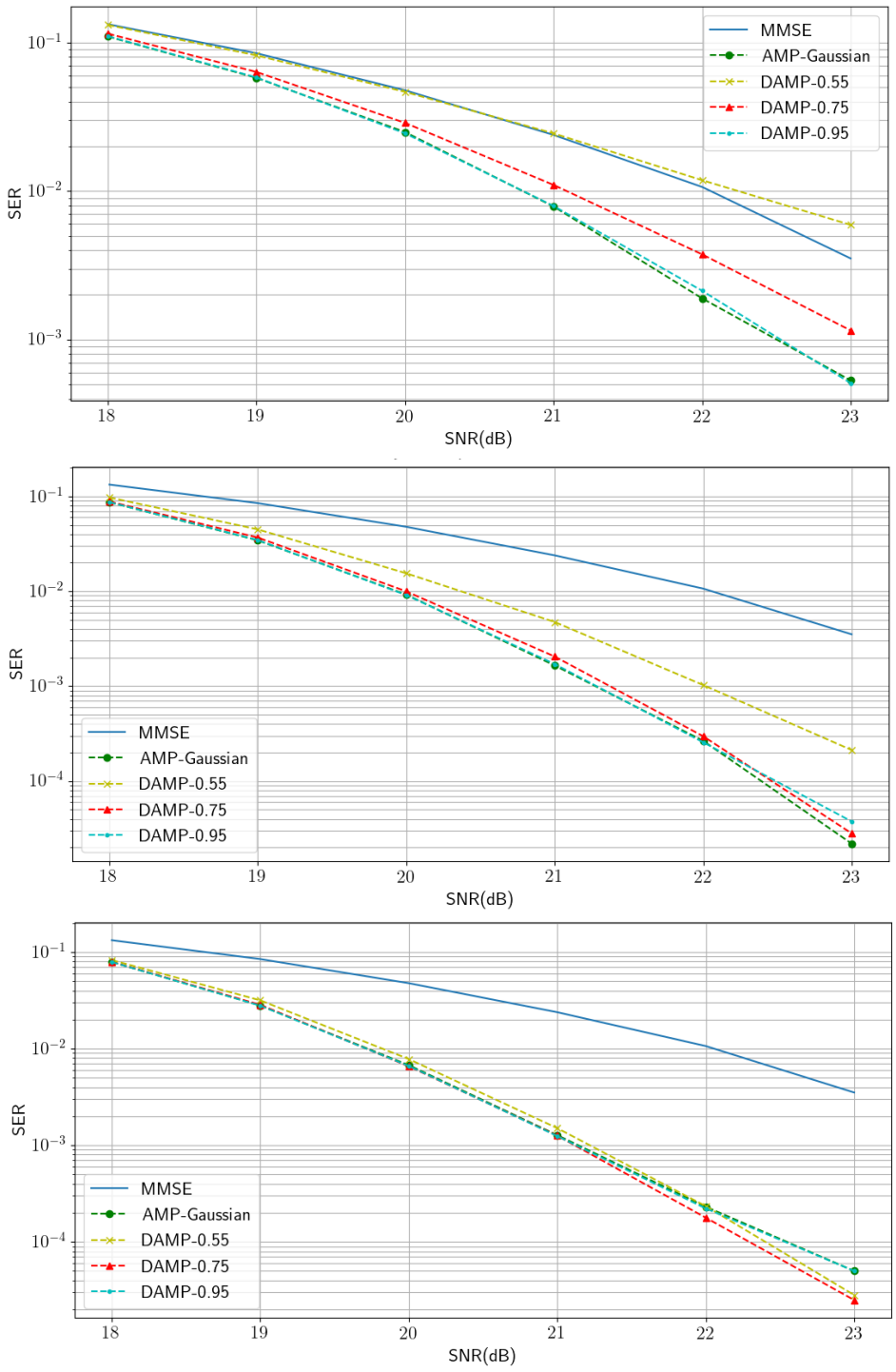


Figure 5.1: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system with 32 transmitters, 64 receivers with randomly sampled i.i.d. Gaussian channels. DAMP runs for three different number of iterations: 10, 12, 14 (from top to bottom).

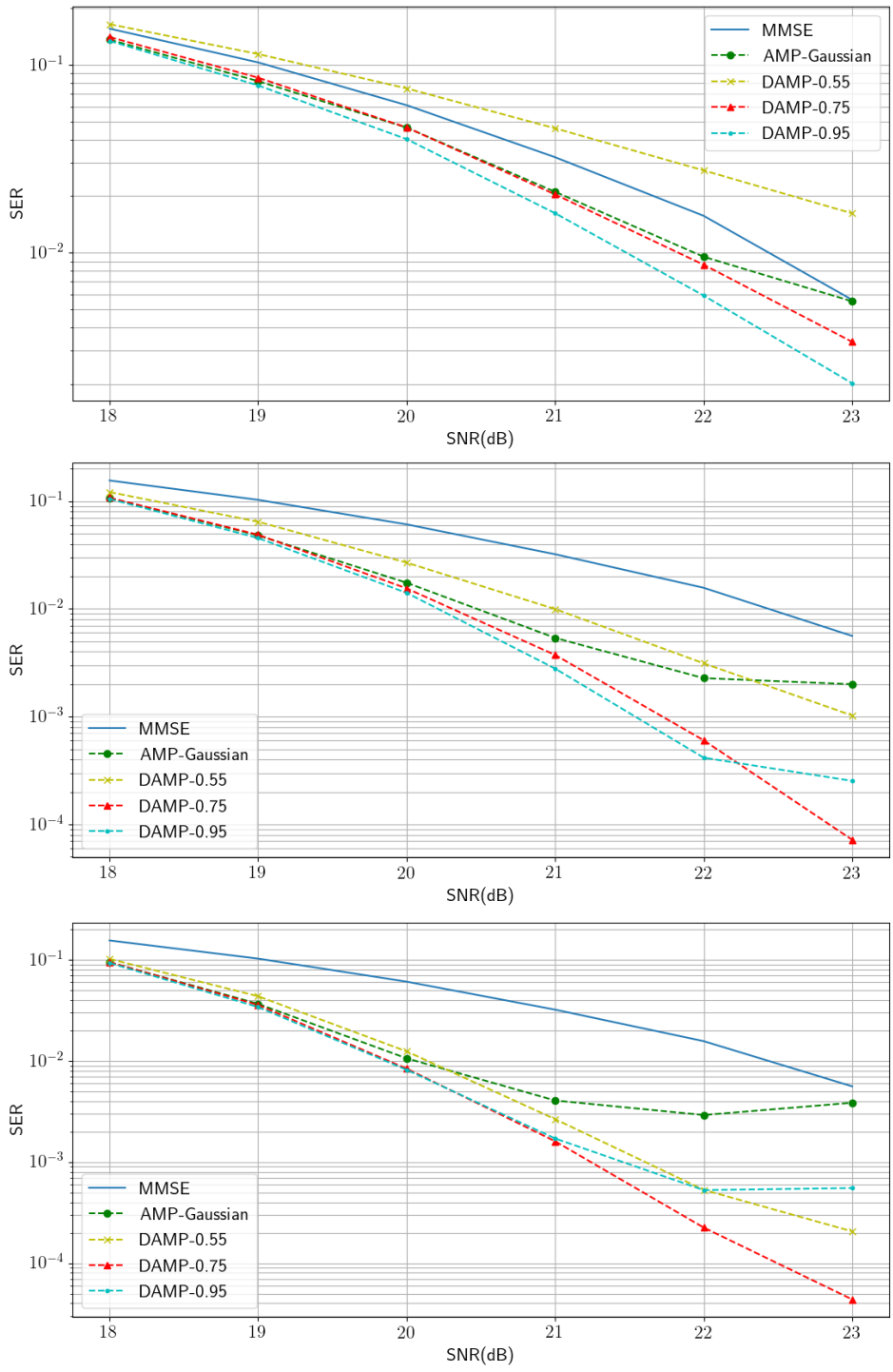


Figure 5.2: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system with 32 transmitters, 64 receivers with Kronecker channel model with correlation $\rho_r = 0.3$ at receiver side. DAMP runs for three different number of iterations: 10, 12, 14 (from top to bottom).

5.2 Offline training

In this section, the algorithms are trained offline over randomly sampled i.i.d. Gaussian channel matrices. Then, the trained algorithms are tested on randomly sampled i.i.d. Gaussian and Kronecker channel matrices. In each simulation we sample a random channel matrix according to the chosen channel model.

5.2.1 Experiment 2

The experiment compares the **MMSE**, **MMNet-iid**, **MMNet-iid-original**, **MMNet-iid-extra** algorithms for the MIMO configuration of 32 transmitters and 64 receivers with modulation scheme QAM-64. We test the algorithms for i.i.d. Gaussian channels.

The goal of the experiment is to understand how the training strategy impacts on the testing performance of MMNet-iid. A discussion of this experiment can be found in Section 6.2.

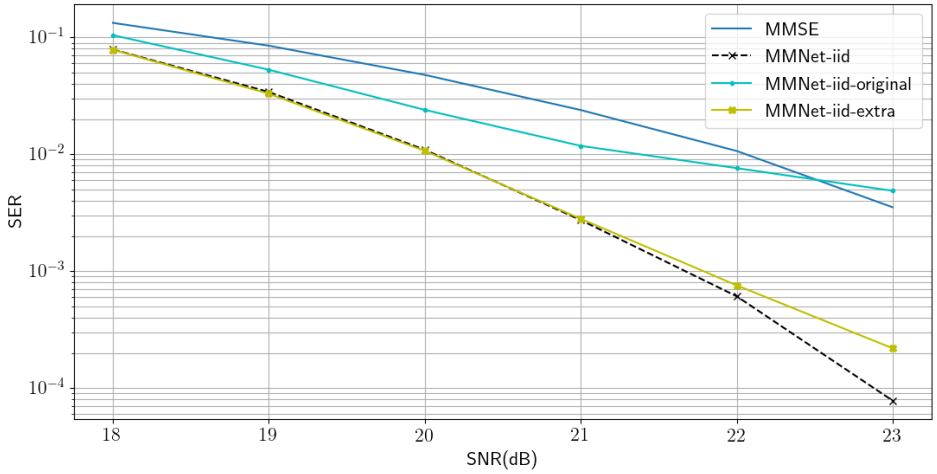


Figure 5.3: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system with 32 transmitters, 64 receivers with randomly sampled i.i.d. Gaussian channels.

5.2.2 Experiment 3

The experiment compares **MMSE**, **DAMP-0.75** with 14 layers, **AMP-Gaussian** with 10 layers and **MMNet-iid** with 10 layers, for the MIMO configuration of 32 transmitters and 64 receivers with modulation scheme QAM-64. We test the algorithms on i.i.d. Gaussian channels and Kronecker channel model with correlation coefficients $\rho_r = 0.15$ and $\rho_r = 0.3$ at receiver side. Then we test the algorithms on the Kronecker channel model with correlation coefficients $\rho_r = 0.3, \rho_t = 0.3$ both at receiver and transmitter side. The goal of the experiment is to analyze the performance of **MMNet-iid** for

different channel correlations. Here, both **AMP-Gaussian** and **MMNet-iid** runs for 10 iterations to understand the improvement brought by the offline training on the performance.

On i.i.d. Gaussian channels, **MMNet-iid** outperforms significantly the **MMSE** baseline. With only ten layers, it has around 2dB gain over **MMSE** at SER of 10^{-2} and it has maximum 0.5 dB degradation compared to **DAMP-0.75** in the SNR region of Figure 5.4. In Figure 5.4, **MMNet-iid** achieves around 2 dB gain relative to **MMSE** at correlation $\rho_r = 0.15$ at receiver side. However, at higher correlation values the performance drops dramatically. Thanks to the offline learning, **MMNet-iid** outperforms **AMP-Gaussian** when they run for the same number of iterations and channels are i.i.d. and Gaussian distributed or lowly correlated.

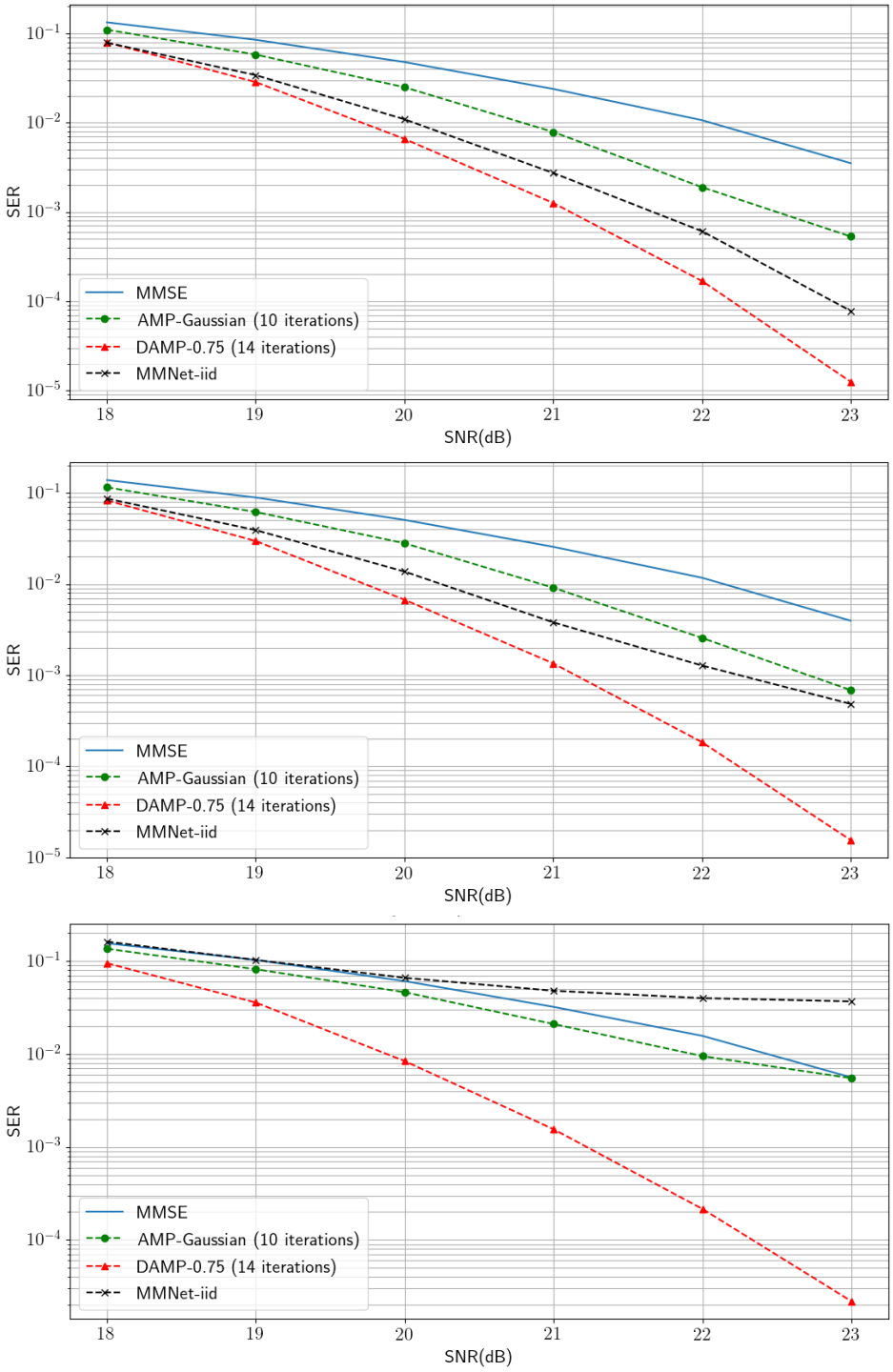


Figure 5.4: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system of 32 transmitters, 64 receivers with randomly sampled i.i.d. Gaussian channels (top) and Kronecker channels with correlation $\rho_r = 0.15$ (middle) and $\rho_r = 0.3$ (bottom) at receiver side.

At the end, we want to understand how **DAMP-0.75** behaves when it is tested on channels correlated both at receiver and transmitter side.

Both on i.i.d. Gaussian channels and on Kronecker channels with correlation $\rho_r = 0.3$ at receiver side, **DAMP-0.75** outperforms **MMSE** of around 2.5dB at SER of 10^{-2} (Figure 5.4). The same performance gain is preserved even by introducing an additional correlation $\rho_t = 0.3$ at transmitter side (Figure 5.5).

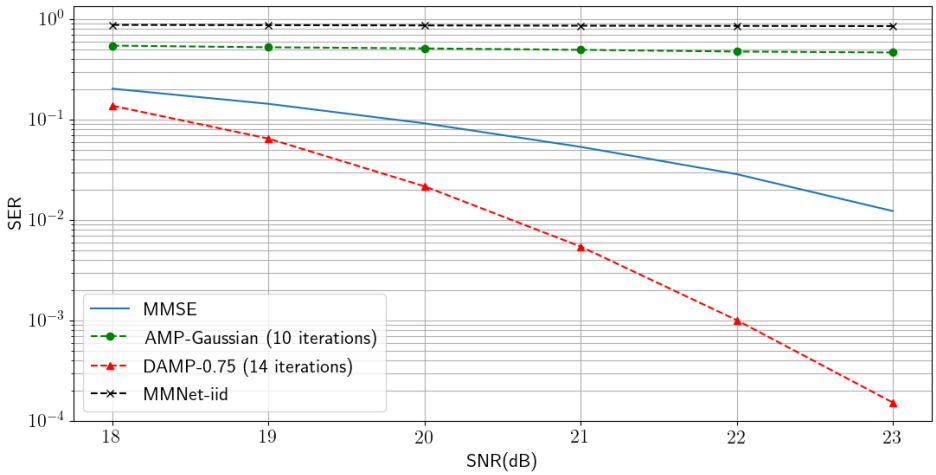


Figure 5.5: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system of 32 transmitters, 64 receivers with Kronecker channels with correlation $\rho_r = 0.3$ at receiver side and $\rho_t = 0.3$ transmitter side.

5.2.3 Experiment 4

The experiment compares the **MMSE**, **BP-MMSE**, **AMP-Gaussian**, **DAMP-0.75**, **MIMO-GNN**, **MIMO-GNN-MMSE** and **MIMO-GNN-SPARSE** algorithms for the MIMO configurations with $N_t = 32, N_r = 64$ and $N_t = 32, N_r = 128$ and modulation schemes QAM-16 and QAM-64. We test the algorithms on i.i.d. Gaussian channels and on Kronecker channel model with correlation coefficients $\rho_r = \rho_t = 0.3$.

The goal of the experiment is to analyze the performance, and the robustness to correlation in the channels, of GNN-based algorithms and **BP-MMSE** compared to AMP-based algorithms. For sake of comparison, **AMP-Gaussian** and **DAMP-0.75** run for 75 iterations. Increasing the number of iterations does not further improve the performance.

Firstly, the experiments show that **BP-MMSE**, **DAMP-0.75**, **AMP-Gaussian**, **MIMO-GNN**, **MIMO-GNN-MMSE** have approximately the same performance on i.i.d. Gaussian channels and modulation QAM-16 (Figure 5.6, 5.7). By increasing the modulation order to QAM-64 or by

introducing correlation in the channel (Figure 5.8), GNN-based algorithms become sensitive to the prior. In these scenarios, **MIMO-GNN** suffers of a performance degradation at high SNR regime. Then, we observe that the best scheme has always a larger performance gain over **MMSE** when the system ratio is $\frac{N_t}{N_r} = 0.5$, between 2dB and 2.5dB when SER is 10^{-2}. This result can be explained by looking at simulations in [24] where the performance gap between MMSE and the optimal ML detector is more prominent when the system ratio increases. In the experiments, **MIMO-GNN-SPARSE** curve is always between **MMSE** and **MIMO-GNN-MMSE** curves: even if removing edges from the graph reduces the performance, it still outperforms **MMSE**. For correlated channels, **MIMO-GNN-SPARSE** is comparable to **MIMO-GNN** in terms of SER.

In Figure 5.9, we show the convergence of the GNN-based algorithms according to the $\frac{2||\mathbf{u}_i^t - \mathbf{u}_i^{t-1}||}{||\mathbf{u}_i^t|| + ||\mathbf{u}_i^{t-1}||}$ metric. We observe how the node hidden state $\mathbf{u}_i$ changes from iteration $t-1$ to iteration t by computing the euclidean norm of the difference $\mathbf{u}_i^t - \mathbf{u}_i^{t-1}$. Since the norm of this difference could be affected by the norm of the hidden state itself, we divide it by the mean of the hidden state norms in the two iterations. In this way, the comparison among different algorithms it is more reliable. The results are average over 32000 different nodes. We can notice that all the algorithms steps toward convergence after each iteration. Especially, **MIMO-GNN-SPARSE** seems to have a better convergence due to the sparsity in the graph.

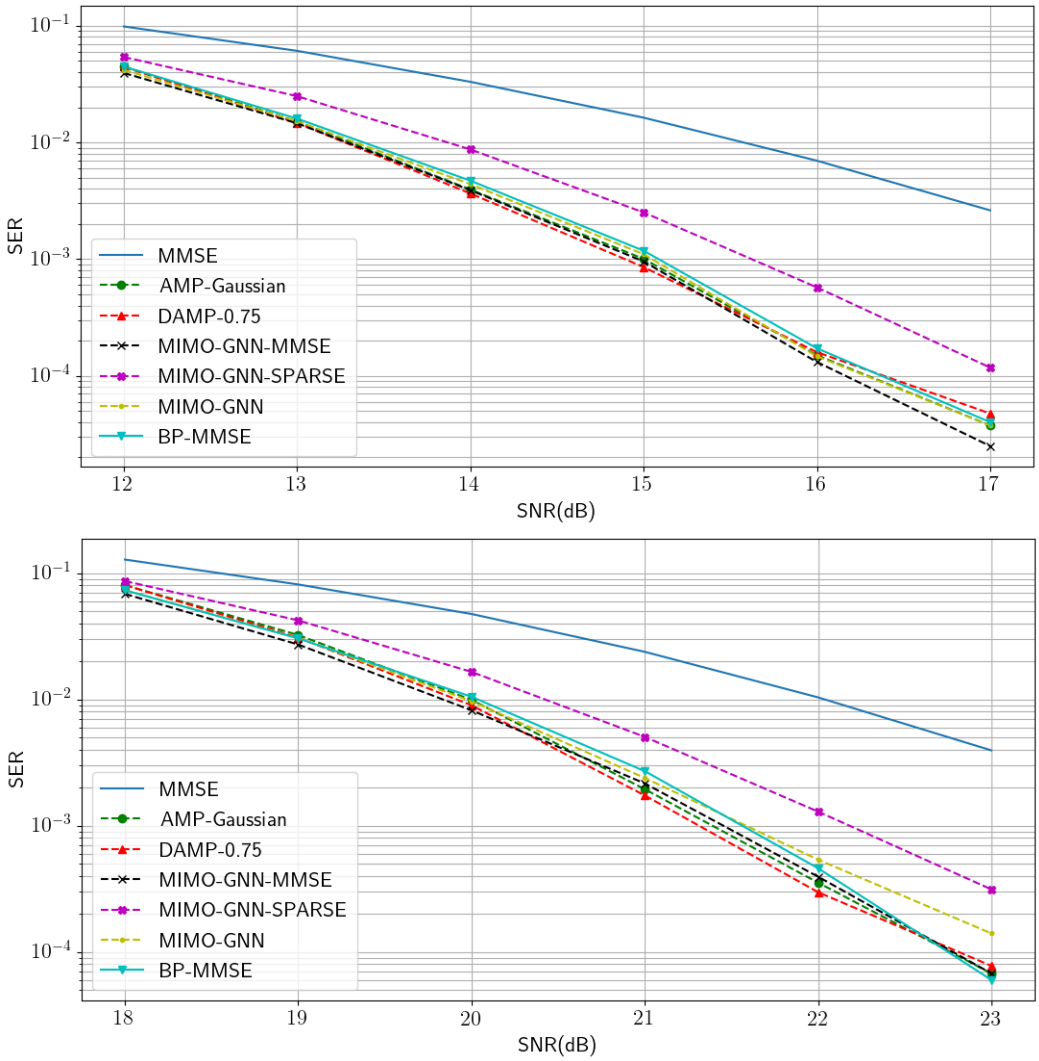


Figure 5.6: SER vs. SNR of different schemes for QAM-16 (top) and QAM-64 (bottom) modulations, MIMO system with 16 transmitters and 32 receivers with randomly sampled i.i.d. Gaussian channels.

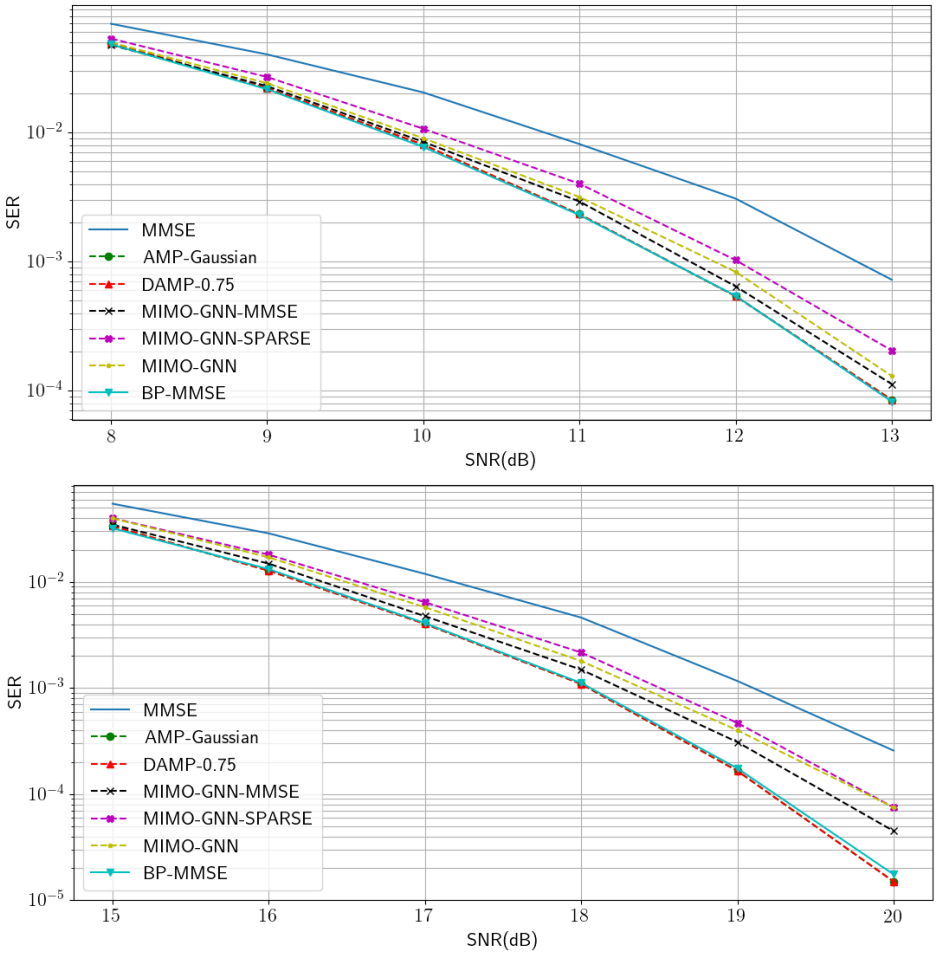


Figure 5.7: SER vs. SNR of different schemes for QAM-16 (top) and QAM-64 (bottom) modulations, MIMO system with 16 transmitters and 64 receivers with randomly sampled i.i.d. Gaussian channels.

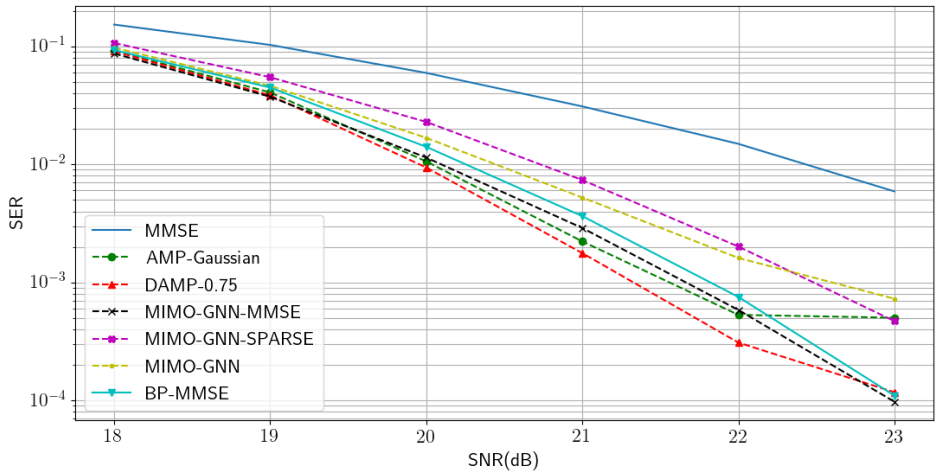


Figure 5.8: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system with 16 transmitters and 32 receivers with channels drawn from the Kronecker model with correlation $\rho_r = \rho_t = 0.3$.

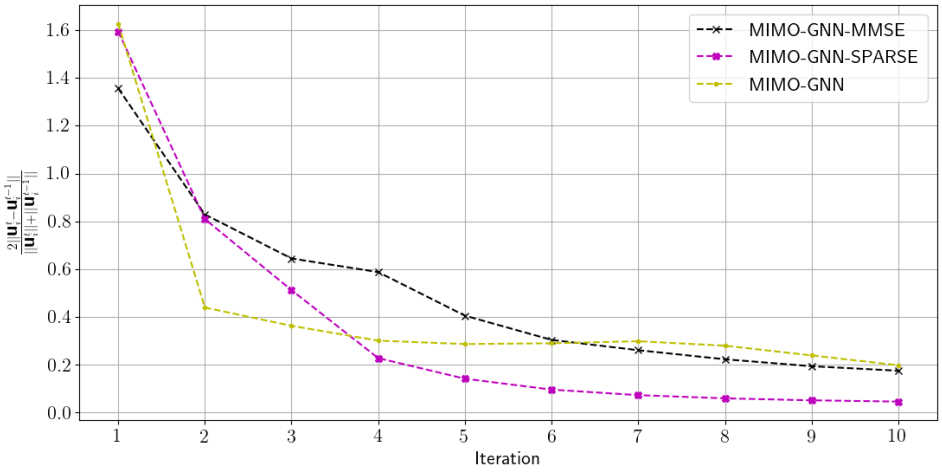


Figure 5.9: Convergence of GNN-based algorithms for QAM-64 modulation, MIMO system with 16 transmitters and 32 receivers with randomly sampled i.i.d. Gaussian channels.

5.3 Online training

In this section, the algorithms **Pseudo-MMNet** and **MMNet** are trained online over a single randomly sampled Kronecker channel matrix with correlation coefficient $\rho_r = 0.3$ at receiver side and $\rho_t = 0.5$ at transmitter side. In each simulation the data is generated with the same channel matrix which **Pseudo-MMNet** and **MMNet** have been trained on.

5.3.1 Experiment 5

The experiment compares the **MMSE**, **DAMP-0.75**, **Pseudo-MMNet** algorithms for the MIMO configuration of 32 transmitters and 64 receivers with modulation scheme QAM-64. Then we provide the results of **Pseudo-MMNet** by considering when:

- the network runs on different numbers of layers (3, 5, 7, 9);
- the network is trained on different numbers of iterations (15, 20, 25);
- the network is trained on different batch sizes (100, 300, 500).

The goal of the experiment is to understand the improvements brought by **Pseudo-MMNet** for medium/high correlation in the channels at receiver and transmitter side. Moreover, we are interested in determining how it is possible to deal with the trade-off between performance and computational complexity of the online training in **Pseudo-MMNet**.

When channels are strongly correlated (Figure 5.10), **Pseudo-MMNet** and **MMNet** achieve around 1-1.5 dB gain over **MMSE** when SER is between $10^{-2} - 10^{-3}$ at the expense of an online training that is not required by the alternative algorithms. Then, we notice that **DAMP** needs more running iterations to outperforms **MMSE**.

In Figure 5.10, we observe that **Pseudo-MMNet** outperforms **MMNet** in performance even by reducing the number of training epochs by a factor 80. In the same experiment we show how it is possible to control the trade-off between performance and complexity by changing the number of epochs e, the number of layers T and the batch size b.

We observe that for low and medium SNR values (≤ 24), there is no perceptible difference in performance among the network with 5, 7 and 9 layers. Instead, the network with 3 layers maintains around 0.5dB degradation with respect to the network with 9 layers for a significant part of the considered SNR range. Then, we notice a small degradation for the network with 5 layers at higher SNR (≥ 25) while the gap between the network of 7 and 9 layers is hardly noticeable for the same SNR values. Therefore, it is possible to adjust the number of layers to achieve the desired complexity both during training and prediction time by having at least 1dB gain over **MMSE** at SER of $10^{-2} - 10^{-3}$. For instance, by reducing the number of

layers from 9 to 3 would decrease the complexity by a factor of 3. Moreover, to further reduce the complexity at training time we show that is possible to reduce the number of training epochs e or the batch size b while keeping an improvement over **MMSE**.

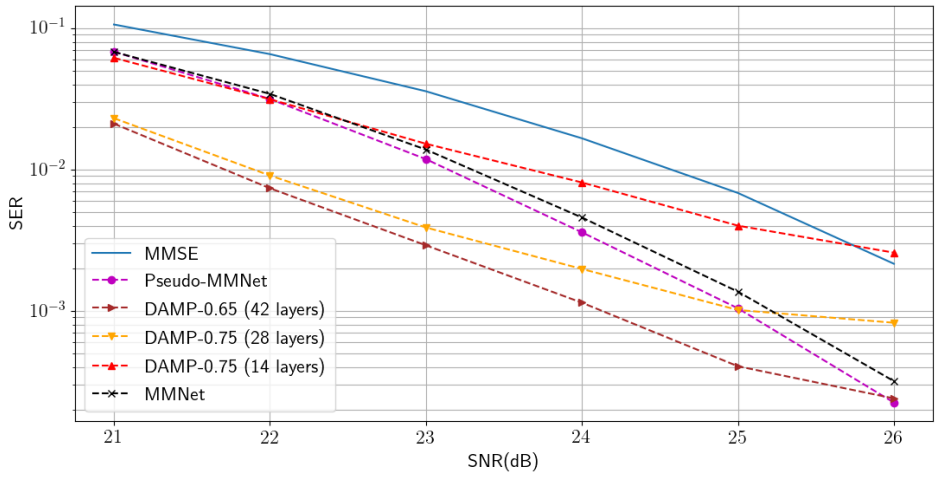


Figure 5.10: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system of 32 transmitters, 64 receivers on a randomly sampled Kronecker MIMO channel with correlation $\rho_r = 0.3$ at receiver side and $\rho_t = 0.5$ at transmitter side.

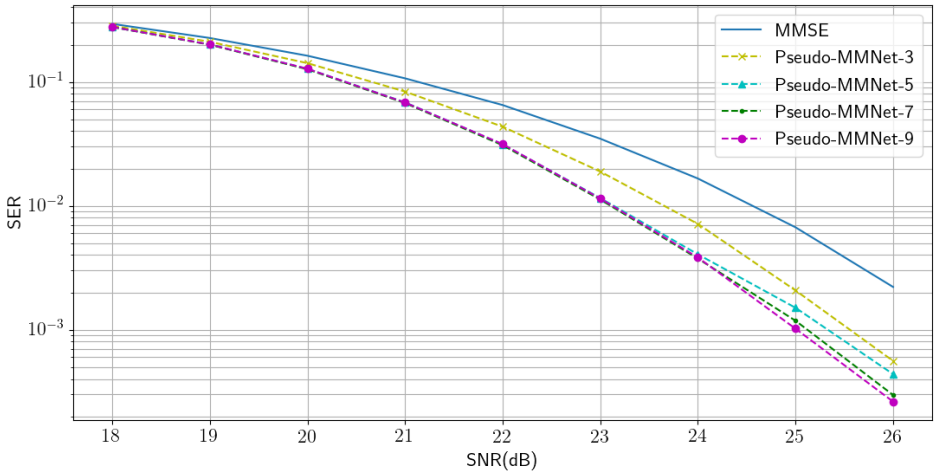


Figure 5.11: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system of 32 transmitters, 64 receivers on a randomly sampled Kronecker MIMO channel with correlation $\rho_r = 0.3$ at receiver side and $\rho_t = 0.5$ at transmitter side. Pseudo-MMNet is compared for different number of layers (3, 5, 7, 9).

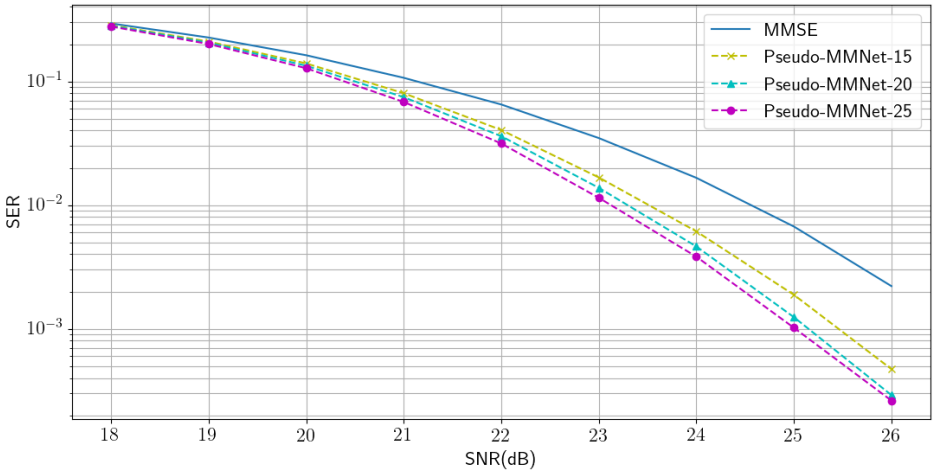


Figure 5.12: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system of 32 transmitters and 64 receivers on a randomly sampled Kronecker MIMO channel with correlation $\rho_r = 0.3$ at receiver side and $\rho_t = 0.5$ at transmitter side. Pseudo-MMNet is compared for different number of training epochs (15, 20, 25).

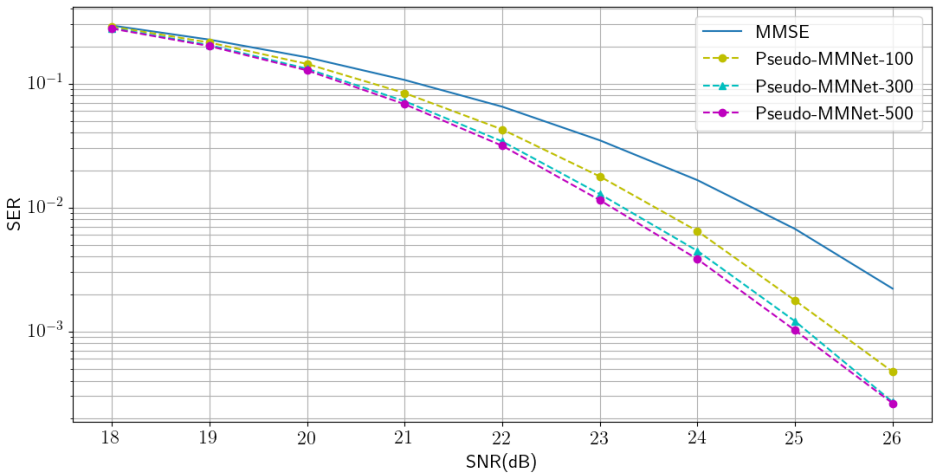


Figure 5.13: SER vs. SNR of different schemes for QAM-64 modulation, MIMO system of 32 transmitters and 64 receivers on a randomly sampled Kronecker MIMO channel with correlation $\rho_r = 0.3$ at receiver side and $\rho_t = 0.5$ at transmitter side. Pseudo-MMNet is compared for different batch size of training (100, 300, 500).

Chapter 6

Discussion

6.1 Performance analysis

Recall the derivation of the AMP-Gaussian algorithm in Section 2.4. It is evident the importance of the i.i.d. assumption on the channels for the correct functioning of the algorithm. The experiments confirm the outstanding performance of AMP-Gaussian on i.i.d. Gaussian channels and the expected lack of robustness for medium correlation in the channels at receiver side. On the other hand, the results show the robustness of DAMP even for medium correlation in the channels both at receiver and transmitter side. In this work we don't provide an analytic justification of the improvements led by damping. We notice that the presence of damping slows down the convergence of the algorithm. Meanwhile, offline training in MMNet-iid allows to speed up the convergence with respect to AMP-Gaussian. When we deal with high correlated channels, DAMP performance degrades at high SNR values while MMNet and Pseudo-MMNet prove to be a good alternative solutions.

With the aid of the prior information, BP-MMSE outperforms MMSE and it is robust to correlation. The same conclusions are valid for MIMO-GNN-MMSE when the system ratio satisfies $\frac{N_t}{N_r} = 0.5$. Instead, MIMO-GNN manages to outperform MMSE under the uniform prior assumption over the transmitted symbols. However, it exhibits slight degradation when there is correlation in the channel or when the modulation is QAM-64. Therefore, MIMO-GNN is sensitive to the prior information when tested on a channel model different from the training. For better results at higher-modulation, probably the model needs to be more complex to learn the underlying distribution of the dataset.

To conclude the performance analysis, we consider as the best schemes the ones that achieve the lowest SER when the channels exhibit low correlation, as it happens in practice. Our experiments demonstrate that DAMP, BP-MMSE and MIMO-GNN-MMSE are the most promising solutions for re-

alistic scenarios. In this work, we do not have simulation results to conclude the same for MMNet and Pseudo-MMNet. Since the scope of this thesis does not limit itself to a performance analysis of the proposed algorithms, in the Section 6.3 we analyze their complexity too.

6.2 MMNet-iid training strategy

In our experiment the original training strategy for MMNet-iid (**MMNet-iid-original**) in [24] doesn't lead to the same results obtained in the original work. The main issue regarding this approach is the chosen loss function described in Section 2.6.1. The loss function averages the loss error after each layer without applying any kind of discount from layer to layer. Therefore, the predominant contribution in the loss value comes from the first layers where the error is higher. In this way, the weights update in the last layer will be less significant compared to the previous ones, leading to bad predictions.

One possible solution to this issue is to modify the loss function by including only the error from the last layer. We observed that this approach leads to the desired result only after a long training, probably due to the vanishing gradient problem explained in Section 2.5.2. Therefore, we come up with the training strategy for **MMNet-iid** described in Section 4.3.1 that speeds up significantly the training. Then, we observe that by adding more training epochs to minimize the loss from the last layer (**MMNet-iid-extra**) doesn't really help.

6.3 Computational complexity

6.3.1 Low-complexity iterative algorithms

In order to recover the transmitted symbols, the iterative algorithms that follows the Scheme 7 have an overall complexity in the order of $\mathcal{O}(TN_r^2)$. For each layer, the dominant operation is the matrix multiplication between the matrix $\mathbf{A}^t$ and the residual $\mathbf{r}^t$. The online training complexity of MMNet and Pseudo-MMNet is $\mathcal{O}(ebTN_r^2)$ where e is the number of training epochs and b is the batch size. Moreover Pseudo-MMNet has an additional complexity of $\mathcal{O}(N_r^3)$ due to the computation of the pseudo inverse of the channel matrix $\mathbf{H}$. In MMNet, the number of parameters to learn is in the order of $\mathcal{O}(TN_rN_t)$ because there is a matrix $\mathbf{A}_t$ to learn for each layer. Instead, Pseudo-MMNet reduces the number of learnable variables by a factor T because the matrix $\mathbf{A}$ to learn is tied among the layer.

On one hand, the experiments show that DAMP outperforms Pseudo-MMNet when we substantially increase the number of layers in DAMP. On the other hand, after introducing a latency for each channel realization, the

complexity of Pseudo-MMNet to recover the transmitted symbols is significantly lower than DAMP because Pseudo-MMNet works with less layers.

6.3.2 GNN-based algorithms

In GNNs the computational complexity is dominated by the computation of function M for all the edges in the graph during each iteration of the algorithm. By recalling that M is a two layers MLP where the first and second layers have outputs of sizes l and $\frac{l}{2}$ respectively, the overall complexity of dense GNNs is $\mathcal{O}(TN_t^2 l^2)$.

By applying the pruning strategy discussed in Section 3.3.1, we can keep only 33% of the edges for i.i.d. Gaussian channels and reduce of 67% the overall complexity of GNNs. However, to compute the MMSE prior and covariance matrix we need to invert a square matrix of dimension N_t. Therefore, the preprocessing complexity of GNNs with MMSE prior is $\mathcal{O}(N_t^3)$.

Apparently the complexity of GNNs seems to be prohibitive for large systems. However, the advancement in distributed processing can be leveraged to use GNNs in realistic scenarios. Indeed, we can notice that, in each iteration, the function M can be computed independently for each edge. Therefore, the algorithm can be easily parallelized.

Chapter 7

Conclusions and future work

In this thesis, we started by formulating MIMO detection as a MAP inference problem that can be approximately solved by running LBP on the corresponding graphical model. Since, under the assumption of uniform prior over transmitted symbols, standard LBP yields poor results, we combine LBP and MMSE in a unique solution that we call BP-MMSE. The performance gain over MMSE is significant but the complexity of the algorithm is dominated by the inversion of a matrix in the computation of MMSE statistics, which makes the complexity prohibitive for large systems. Therefore, we use GNNs to learn an iterative message-passing algorithm, MIMO-GNN, that solves the same task even under the uniform prior assumption. The efficiency of the algorithm and its predisposition to be parallelized make MIMO-GNN a promising solution. Then we propose another class of algorithms, inspired by AMP and derived from LBP. Instead of exchanging messages on a dense graphical model, these algorithms reduce the complexity by working with matrix multiplications. We recall how to derive AMP for MIMO from LBP and we slightly modify the algorithm to achieve better results at high SNR values. The same algorithm is then improved by adding damping. The novel scheme is called DAMP and it outperforms AMP on correlated channels. When the correlation is too high, even DAMP performance degrades at high SNR values. Therefore we propose a deep neural scheme called Pseudo-MMNet that solves the issue with an online training for each channel realization.

We leave to future work an analytical justification for DAMP and an evaluation of the impact of the channel matrix condition number on all the proposed algorithms. Regarding DAMP, we suggest to check through the Anderson-Darling test if damping really helps to have decoupled Gaussian noisy predictions after the linear operator. Moreover, we suggest to test Pseudo-MMNet on multiple channel realization in order to provide a more reliable analysis of its performance. To conclude, we remember to the reader that all the simulations are conducted on column-normalized channel ma-

trices with a perfect knowledge of the channel at receiver side. Therefore, we encourage future research on the proposed algorithms where channel matrices are not column-normalized and with AWGN in the channel.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016.

[2] Aviad Aberdam, Alona Golts, and Michael Elad. Ada-LISTA: Learned solvers adaptive to varying models, 2020.

[3] M. A. Albreem, M. Juntti, and S. Shahabuddin. Massive MIMO detection techniques: A survey. *IEEE Communications Surveys Tutorials*, 21(4):3109–3132, 2019.

[4] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

[5] Mohsen Bayati and Andrea Montanari. The dynamics of message passing on dense graphs, with applications to compressed sensing. *IEEE Transactions on Information Theory*, 57(2):764–785, Feb 2011.

[6] Danny Bickson. Gaussian belief propagation: Theory and aplication, 2008.

[7] Thomas Blumensath and Mike Davies. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 14:629–654, 12 2008.

[8] Mark Borgerding and Philip Schniter. Onsager-corrected deep networks for sparse linear inverse problems. *ArXiv*, abs/1612.01183, 2016.

[9] Mark Borgerding, Philip Schniter, and Sundeep Rangan. AMP-inspired deep networks for sparse linear inverse problems. *IEEE Transactions on Signal Processing*, 65(16):4293–4308, Aug 2017.

[10] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoderdecoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[11] J. M. Cioffi. *Part I: signal processing and detection.*

[12] David L. Donoho, Arian Maleki, and Andrea Montanari. Message-passing algorithms for compressed sensing. *Proceedings of the National Academy of Sciences*, 106(45):18914–18919, Oct 2009.

[13] David L. Donoho, Arian Maleki, and Andrea Montanari. How to design message passing algorithms for compressed sensing. 2011.

[14] Gal Elidan, Ian McGraw, and Daphne Koller. Residual belief propagation: Informed scheduling for asynchronous message passing, 2012.

[15] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[16] J. Goldberger and A. Leshem. Pseudo prior belief propagation for densely connected discrete graphs. In *2010 IEEE Information Theory Workshop on Information Theory (ITW 2010, Cairo)*, pages 1–5, Jan 2010.

[17] Jacob Goldberger and Amir Leshem. A gaussian tree approximation for integer least-squares. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 638–645. Curran Associates, Inc., 2009.

[18] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 399–406, Madison, WI, USA, 2010. Omnipress.

[19] Dongning Guo and Sergio Verdú. Multiuser detection and statistical mechanics. 2003.

[20] Hengtao He, Chao-Kai Wen, Shi Jin, and Geoffrey Ye Li. A model-driven deep learning network for MIMO detection. *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Nov 2018.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[22] C. Jeon, R. Ghods, A. Maleki, and C. Studer. Optimality of large mimo detection via approximate message passing. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 1227–1231, 2015.

[23] Charles Jeon, Ramina Ghods, Arian Maleki, and Christoph Studer. Optimal data detection in large MIMO, 2018.

[24] Mehrdad Khani, Mohammad Alizadeh, Jakob Hoydis, and Phil Fleming. Adaptive neural signal detection for massive MIMO, 2019.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[26] Christian Knoll, Michael Rath, Sebastian Tschiatschek, and Franz Pernkopf. Message scheduling methods for belief propagation. In *ECML/PKDD*, 2015.

[27] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory*, 47:498–519, 2001.

[28] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2015.

[29] Dong Liu, Nima N. Moghadam, Lars K. Rasmussen, Jinliang Huang, and Saikat Chatterjee. α belief propagation as fully factorized approximation, 2019.

[30] S. L. Loyka. Channel capacity of mimo architecture using the exponential correlation matrix. *IEEE Communications Letters*, 5(9):369–371, 2001.

[31] Junjie Ma and Li Ping. Orthogonal amp, 2016.

[32] Arian Maleki. Approximate message passing algorithms for compressed sensing. 2011.

[33] Kevin Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study, 2013.

[34] United Nations. *The Sustainable Development Goals Report 2019*. 2019.

[35] Afif Osseiran, J.F. Monserrat, Patrick Marsch, Olav Queseth, Hugo Tullberg, Mikael Fallgren, Katsutoshi Kusume, Andreas Höglund, Heinz Droste, Icaro Silva, Peter Rost, Mauro Boldi, Joachim Sachs, Petar Popovski, David Gozalvez-Serrano, Peter Fertl, Zexian Li, Fernando Moya, Gabor Fodor, and Ji Lianghai. *5G Mobile and Wireless Communications Technology*. 06 2016.

[36] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[37] M.J. Wainwright, T.S. Jaakkola, and A.S. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, Jul 2005.

[38] Z. Xie, R. T. Short, and C. K. Rushforth. A family of suboptimum detectors for coherent multiuser communications. *IEEE Journal on Selected Areas in Communications*, 8(4):683–690, 1990.

[39] S. Yang and L. Hanzo. Fifty years of mimo detection: The road to large-scale mimos. *IEEE Communications Surveys Tutorials*, 17(4):1941–1988, 2015.

[40] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. *Understanding Belief Propagation and Its Generalizations*, page 239–269. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[41] KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks, 2018.

[42] Seokhyun Yoon and Chan-Byoung Chae. Low-complexity MIMO detection based on belief propagation over pairwise graphs. *IEEE Transactions on Vehicular Technology*, 63(5):2363–2377, Jun 2014.

Appendix A

Proof 1

Given $G(z,\tau) = F^{(2)} - F^2$ such that

$$F = \frac{1}{Z}\sum_{s\in\mathcal{A}} s\exp(-\frac{(z-s)^2}{2\tau})$$

and

$$F^{(2)} = \frac{1}{Z}\sum_{s\in\mathcal{A}} s^2\exp(-\frac{(z-s)^2}{2\tau}),$$

we want to prove $G(z,\tau) = \tau\frac{\partial F(z,\tau)}{\partial z}$.
Recalling $Z = \sum_{s\in\mathcal{A}}\exp(-\frac{(z-s)^2}{2\tau})$ and defining $\tilde{F} = ZF,\ \tilde{F^{(2)}} = ZF^{(2)}$:

$$\frac{\partial\tilde{F}}{\partial z} = \frac{1}{\tau}(\tilde{F^{(2)}} - z\tilde{F}), \tag{A.1}$$

$$\frac{\partial\frac{1}{Z}}{\partial z} = -\frac{1}{\tau}\frac{1}{Z^2}(\tilde{F} - zZ), \tag{A.2}$$

$$\frac{\partial F(z,\tau)}{\partial z} = \frac{\partial\frac{1}{Z}\tilde{F}}{\partial z} = \frac{\frac{1}{Z}\partial\tilde{F}}{\partial z} + \frac{\tilde{F}\partial\frac{1}{Z}}{\partial z} = \frac{1}{\tau}\frac{1}{Z}(\tilde{F^{(2)}} - z\tilde{F} - \frac{\tilde{F}^2}{Z} + z\tilde{F}) = \frac{1}{\tau}(F^{(2)} - F^2). \tag{A.3}$$

Appendix B

Proof 2

Here we prove that the messages $m^t_{l\to k}(x_l)$ from variable node to factor node can be approximated by the Gaussian density $\phi^{t+1}_{l\to k}(x_l)$ (2.67).

$$m^t_{l\to k}(x_l) \propto \prod_{a\neq k} m^{t-1}_{a\to l}(x_l)p(x_l) \propto \exp(-\frac{\sum_{a\neq k}(h_{a,l}x_l - r^t_{a\to l})^2}{\sigma^2(1+\tau^t_{k\to l})})p(x_l) =$$
$$= \exp(-\frac{\sum_{a\neq k}(h_{a,l}x_l - r^t_{a\to l})^2}{\sigma^2(1+\tau^t_{k\to l})})p(x_l) =$$
$$= \exp(-\frac{\sum_{a\neq k}(h^2_{a,l}x^2_l - 2h_{a,l}x_l r^t_{a\to l} + (r^t_{a\to l})^2)}{\sigma^2(1+\tau^t_{k\to l})})p(x_l) \propto$$
$$\propto \exp(-\frac{\sum_{a\neq k}(h^2_{a,l}x^2_l - 2h_{a,l}x_l r^t_{a\to l})}{\sigma^2(1+\tau^t_{k\to l})})p(x_l).$$

We follow the derivation by reminding that the columns of $\mathbf{H}$ are normalized such that $||\mathbf{h}_a|| = 1$ and by approximating $h^2_{k,l}x^2_l = O(\frac{x^2_l}{N_r})$:

$$m^t_{l\to k}(x_l) \propto \exp(-\frac{x^2_l - h^2_{k,l}x^2_l - 2x_l\sum_{a\neq k} h_{a,l}r^t_{a\to l}}{\sigma^2(1+\tau^t_{k\to l})})p(x_l) \propto$$
$$\propto \phi^{t+1}_{l\to k}(x_l)(1+O(\frac{x^2_l}{N_r})),$$

where $\phi^{t+1}_{l\to k}(x_l)$ is defined in (2.67). The last step of the derivation required to complete the square at the numerator of the exponent in order to have a Gaussian density up to a normalization constant.