UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

CURSO DE GRADUAÇÃO EM ENGENHARIA DE TELECOMUNICAÇÕES

CAIO LUCAS MESQUITA DE MORAES

UMA AVALIAÇÃO DE APRENDIZADO PROFUNDO (*DEEP LEARNING*) APLICADO À COMUNICAÇÃO SEM FIO PONTA-A-PONTA.

FORTALEZA

2022

CAIO LUCAS MESQUITA DE MORAES

UMA AVALIAÇÃO DE APRENDIZADO PROFUNDO (*DEEP LEARNING*) APLICADO À COMUNICAÇÃO SEM FIO PONTA-A-PONTA.

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Telecomunicações do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Telecomunicações.

Orientador: Prof. Dr. Charles Casimiro Cavalcante

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M819a Moraes, Caio Lucas Mesquita de.
Uma avaliação de aprendizado profundo (Deep Learning) aplicado à comunicação sem fio ponta-a-ponta. / Caio Lucas Mesquita de Moraes. – 2022.
68 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia de Telecomunicações, Fortaleza, 2022.
Orientação: Prof. Dr. Charles Casimiro Cavalcante.

1. Comunicação sem fio. 2. Aprendizado profundo. 3. Camada física. 4. Codificação de canal. I. Título.
CDD 621.382

CAIO LUCAS MESQUITA DE MORAES

UMA AVALIAÇÃO DE APRENDIZADO PROFUNDO (*DEEP LEARNING*) APLICADO À COMUNICAÇÃO SEM FIO PONTA-A-PONTA.

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Telecomunicações do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Telecomunicações.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Charles Casimiro Cavalcante (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Yuri Carvalho Barbosa Silva
Universidade Federal do Ceará (UFC)

Prof. Dr. Walter da Cruz Freitas Júnior
Universidade Federal do Ceará (UFC)

Dedico este trabalho aos meus pais Jeovan e Glaudene, por tudo que fizeram por mim durante todos esse anos de graduação e durante todo o transcurso da minha vida até aqui.

AGRADECIMENTOS

Ao Deus Triúno, criador de todas as coisas, por meio de quem tudo subsiste e a fonte absoluta de todo conhecimento e graça. Grato eternamente a Ele pela vida e tudo o mais.

À minha família, por todo o suporte e compreensão que foram e têm sido fundamentais na minha jornada. Em especial, meus pais, por me levarem a conhecer a Deus, pela educação que me deram e sem os quais nada disso teria sido possível.

À minha namorada e futura esposa, Maria Eduarda, por toda a paciência comigo, por compreender os meus objetivos e me ouvir sempre. Não só ela, mas sua preciosa família também.

Aos meus professores por todos os ensinamentos, em especial ao meu orientador Prof. Charles Casimiro por ter prontamente atendido o meu pedido de orientação, por todas as discussões técnicas, sugestões pertinentes e pelo grupo de orientação onde pude aprender bastante, mesmo em tão pouco tempo, com os seminários apresentados pelos seus orientandos. Tem sido uma oportunidade ímpar fazer parte do grupo.

Aos professores que compuseram a banca examinadora, Prof. Yuri Silva e Prof. Walter Freitas, pela leitura atenta que oportunizou diversas melhorias no trabalho.

A todos os colegas de curso, por todas as boas conversas e discussões técnicas que favoreceram meu crescimento profissional e pessoal.

A todos os colaboradores da UFC, em especial aos que fazem parte do Departamento de Engenharia de Teleinformática, que direta ou indiretamente contribuíram para minha trajetória.

”Ó profundidade da riqueza da sabedoria e do conhecimento de Deus! Quão insondáveis são os seus juízos e inescrutáveis os seus caminhos! Quem conheceu a mente do Senhor? Ou quem foi seu conselheiro? Quem primeiro lhe deu, para que ele o recompense? Pois dele, por ele e para ele são todas as coisas. A ele a glória para sempre. Amém.”

(Paulo de Tarso)

RESUMO

Neste trabalho propõe-se avaliar a possibilidade de usar técnicas de Aprendizado Profundo (*Deep Learning*) na camada física de um sistema de comunicação digital. A principal e única arquitetura utilizada neste trabalho consiste em uma estrutura *Autoencoder* composta por duas redes neurais profundas (DNN), uma como codificadora e outra como decodificadora. A partir disso, buscou-se remodelar a camada física de um sistema de comunicação sem fio clássico como um bloco único, substituindo especificamente os blocos de modulação e codificação, de tal forma a otimizar o processo de transmissão e recepção ponta-a-ponta. Os resultados do trabalho se resumem em simulações computacionais com comparações feitas com esquemas de modulação e codificação convencionais para investigar a possibilidade aludida. Os resultados utilizando taxa de erro de bit (BER) e taxa de erro de bloco (BLER) mostram que a abordagem baseada em Aprendizado Profundo é de certo modo promissora, apresentando desempenhos comparáveis aos convencionais ou até superiores em determinados casos. Alguns diagramas de constelação também demonstram a capacidade da arquitetura *Autoencoder* de aprender um esquema único de símbolos robusto em face aos efeitos do canal. Um cenário de usuário único com o modelo de canal AWGN (Ruído Aditivo Gaussiano Branco) foi adotado neste trabalho.

Palavras-chave: Comunicação sem fio. Aprendizado profundo. *Autoencoders*. Camada física. Modulação. Codificação de canal.

ABSTRACT

This work proposes to evaluate the possibility of using Deep Learning techniques in the physical layer of a digital communication system. The main and only architecture used in this work consists of an Autoencoder framework composed by two deep neural networks (DNN), one as an encoder and another as a decoder. Thus, we sought to remodel the physical layer of a classic wireless communication system as a single block, specifically replacing the modulation and encoding blocks, in such a way as to optimize the end-to-end transmission and reception process. The results of the work are obtained by computational simulations with comparisons made with conventional modulation and coding schemes to investigate the aforementioned possibility. The results using bit error rate (BER) and block error rate (BLER) show that the approach based on Deep Learning is somehow promising, presenting comparable performances to conventional ones or even superior in certain cases. Some constellation diagrams also demonstrate the ability of the Autoencoder architecture to learn a unique symbol scheme robust to the channel impairments. A single-user scenario with the AWGN (Additive White Gaussian Noise) channel model was adopted in this work.

Keywords: Wireless communication. Deep learning. Autoencoders. Physical layer. Modulation. Channel encoding.

LISTA DE FIGURAS

Figura 1 – Diagrama funcional de um sistema de comunicação digital. 21
Figura 2 – Modelo simplificado de canal gaussiano. 23
Figura 3 – Formas básicas de sinalização binária. 26
Figura 4 – Exemplos de constelações de vários esquemas de modulação. 27
Figura 5 – Quatro valores de fases da modulação QPSK com os bits correspondentes. . 28
Figura 6 – Constelação para modulação 16-QAM. 30
Figura 7 – Codificador convolucional com comprimento de restrição 3 e taxa 1/2. . . . 33
Figura 8 – Treliça para decodificação, por algoritmo de Viterbi, de Códigos Convolucionais (CC) (2,1,3). 34
Figura 9 – RNA de McCulloch e Pitts. 36
Figura 10 – Unidade linear com *threshold*. 37
Figura 11 – Modelo multi-perceptron como um classificador de múltiplas saídas. 37
Figura 12 – Modelo matemático do Perceptron. 38
Figura 13 – Perceptron multicamada. 39
Figura 14 – Funções de ativação (a) Unidade Linear Retificada (*Rectified Linear Unit*) (ReLU) e (b) *leaky*-ReLU. 42
Figura 15 – Arquitetura de um Autoencoder incompleto. 45
Figura 16 – Exemplo de grafo para cálculo de uma função. 47
Figura 17 – Sistema de comunicação em canal AWGN como um Autoencoder 50
Figura 18 – BLER para taxa R = 1 [bit/uso do canal] comparado com BPSK em canal AWGN. SNR_{treino} = 7dB. 53
Figura 19 – BLER para taxa R = 2 [bits/uso do canal] comparado com QPSK em canal AWGN. SNR_{treino} = 7dB. 54
Figura 20 – Constelação aprendida de (M=4, n=1) e (M=16, n=1), respectivamente. . . . 55
Figura 21 – Constelação aprendida de M=4 e n=2. 55
Figura 22 – BER para diferentes abordagens na escolha da SNR_{treino}. 56
Figura 23 – BER para diferentes valores de SNR_{treino}. Sistema com M=256 e n=8. . . . 57
Figura 24 – BER para diferentes valores de *batch*. 58
Figura 25 – BER para comparação de Autoencoder com M={16, 256} e CC com R=1/2. 60
Figura 26 – BER para comparação de Autoencoder com M={16, 256} e CC com R=1/3. 61

Figura 27 – BER para comparação de Autoencoder com M={16, 256} e CC com R=1/2 e K = {400,800,2000,6000}. 62
Figura 28 – BER para comparação de Autoencoder com M={16, 256} e CC com R=1/3 e K = {400,800,2000,6000}. 63
Figura 29 – BLER versus SNR para comparação de Autoencoder com vários *baselines*. . 64

LISTA DE TABELAS

Tabela 1 – Síndromes e padrões de erro corrigíveis para Hamming (7,4). 32
Tabela 2 – Tipos de função de perda. 40
Tabela 3 – Esboço do modelo do Autoencoder. 51
Tabela 4 – Diferentes parâmetros do Autoencoder e os sistemas convencionais respectivos. 52
Tabela 5 – Diferentes parâmetros do Autoencoder e os *baselines* respectivos. 59

LISTA DE ABREVIATURAS E SIGLAS

5G	5ª Geração da Telefonia Celular
API	Interface de Programação de Aplicações (*Application Programming Interface*)
ASK	Chaveamento por Deslocamento de Amplitude (*Amplitude Shift Keying*)
AWGN	Ruído Aditivo Gaussiano Branco (*Additive White Gaussian Noise*)
BER	Taxa de Erro de Bit (*Bit Error Rate*)
BLER	Taxa de Erro de Bloco (*Block Error Rate*)
BPA	Algoritmo de Retropropagação (*Backpropagation Algorithm*)
BPSK	Modulação por Deslocamento de Fase Binária (*Binary Phase Shift Keying*)
CC	Códigos Convolucionais
CNN	Redes Neurais Convolucionais (*Convolutional Neural Networks*)
CPU	Unidade Central de Processamento (*Central Processing Unit*)
DL	Aprendizado Profundo (*Deep Learning*)
DNN	Rede Neural Profunda (*Deep Neural Network*)
GAN	Redes Adversárias Generativas (*Generative Adversarial Networks*)
GPU	Unidades de Processamento Gráfico (*Graphics Processing Units*)
IoT	Internet das Coisas (*Internet of Things*)
LDPC	Checagem de Paridade de Baixa Densidade (*Low Density Parity Check*)
LTU	Unidade Linear com *Threshold* (*Linear Threshold Unit*)
MIMO	Múltiplas-Entradas Múltiplas-Saídas (*Multiple-Input Multiple-Output*)
ML	Aprendizado de Máquina (*Machine Learning*)
MLD	Decodificação por Máxima Verossimilhança (*Maximum Likelihood Decoding*)
MLP	Perceptron Multi-camadas (*Multi-layer Perceptron*)
MSE	Erro Quadrático Médio (*Mean Squared Error*)
PCA	Análise dos Componentes Principais (*Principal Component Analysis*)
PSK	Modulação por Deslocamento de Fase (*Phase Shift Keying*)
QPSK	Chaveamento por Deslocamento de Frequência em Quadratura (*Quadrature Phase-Shift Keying*)
ReLU	Unidade Linear Retificada (*Rectified Linear Unit*)
RNA	Redes Neurais Artificiais
SCD	Sistema de Comunicação Digital

SGD	Gradiente Descendente Estocástico (*Stochastic Gradient Descent*)
SNR	Relação Sinal-Ruído (*Signal-to-Noise Ratio*)
VQ	Quantização Vetorial (*Vector Quantization*)

LISTA DE SÍMBOLOS

$\mathbb{M}$	Conjunto de M mensagens
K	Tamanho do bloco de informação
R	Taxa de comunicação ou taxa de codificação (especificado no texto)
k	Número de bits de informação na mensagem s
n	Número de usos do canal
n_b	Número de bits por bloco de informação
s	Símbolo (mensagem) transmitido
$\mathbf{x}$	Vetor de informação transmitido
$\mathbf{y}$	Vetor de informação recebido
β	Variância do ruído para canal AWGN
$\mathbb{C}^n$	Conjunto dos n vetores complexos
$E\{.\}$	Operador esperança
E_b/N_0	Energia por bit por densidade espectral de potência de ruído
η	Taxa de aprendizado
∇_θ	Operador gradiente

SUMÁRIO

1	**INTRODUÇÃO**	16
1.1	**Motivação e objetivo**	16
1.2	**Trabalhos relacionados**	17
1.3	**Estrutura do trabalho**	18
2	**FUNDAMENTAÇÃO TEÓRICA**	20
2.1	**Sistema de comunicação ponta-a-ponta**	20
2.2	**Modelo de blocos clássico**	25
2.3	**Aprendizado profundo**	34
2.3.1	***Overview***	35
2.3.2	***Autoencoders***	44
2.3.3	***Potencial dos Autoencoders para comunicação ponta-a-ponta***	46
2.3.4	***Bibliotecas Tensorflow e Keras***	47
3	**RESULTADOS DE SIMULAÇÃO**	49
3.1	**Modelo do sistema**	49
3.2	**Sistemas sem codificação**	52
3.3	**Sistemas com codificação**	58
4	**CONCLUSÕES**	66
	REFERÊNCIAS	68

1 INTRODUÇÃO

O presente capítulo lança luz sobre a problemática que será tratada ao longo de todo o trabalho, à medida em que apresenta as principais motivações e objetivos deste trabalho na Seção 1.1, bem como expõe o leitor aos trabalhos relacionados na Seção 1.2. Por fim, a Seção 1.3 esclarece a estrutura deste trabalho.

1.1 Motivação e objetivo

O desafio mais fundamental de um sistema de comunicação é transmitir uma informação, uma mensagem, por um canal, de forma a recuperá-la na outra ponta com o máximo de confiabilidade, rapidez e segurança. Ao longo da evolução das comunicações, diversas técnicas foram sendo desenvolvidas e aperfeiçoadas, tornando os transmissores e receptores mais robustos e mais flexíveis.

Com o crescimento da microeletrônica e da computação, e as formalizações matemáticas do processo de transmissão e recepção de informação, os sistemas convencionais de comunicação foram sendo projetados para funcionarem de forma mais modularizada, em blocos, cada um com sua funcionalidade individual. Isso facilitou o controle dos sistemas, e mostrou-se ser bastante eficiente até hoje. As taxas de transmissão cada vez mais altas, a diminuição em latência, a eficiência energética e a massiva densidade de conexão são alguns pontos que têm se tornado uma realidade e que mostram que os atuais sistemas operam próximos à otimalidade.

Contudo, com a chegada do 5G, a quinta geração das comunicações móveis, tem surgido uma maior demanda por sistemas ainda mais eficientes. As variadas aplicações e serviços em IoT (Internet of Things), por exemplo, têm requerido uma maior adaptabilidade da rede ao canal e às imperfeições (diga-se não-linearidades) de hardware. O tráfego de dados cada vez mais volumoso por todas essas aplicações tem desafiado os atuais sistemas convencionais, exigindo um gerenciamento mais dinâmico e veloz.

Com esse cenário, tem crescido nos últimos anos o interesse em abordagens mais voltadas para técnicas de Aprendizado de Máquina (*Machine Learning*) e Aprendizado Profundo (*Deep Learning*, doravante DL) [1], em especial esta última, para lidar com os desafios inerentes às redes de comunicação móvel modernas e às aplicações que fazem uso delas.

A convencional separação em blocos nos Sistemas de Comunicação Digital (SCD)

[1] Em alguns contextos de mercado pode se chamar Computação Cognitiva também.

tem garantido ótimos desempenhos para prover os serviços atuais, no entanto, não há uma garantia de que a performance ponta-a-ponta, em sua totalidade, seja ótima. Além do mais, boa parte das formalizações matemáticas são feitas levando em consideração algumas premissas que muitas vezes não são encontradas na prática, nos canais de comunicação reais.

Com isso, pesquisas recentes têm sido feitas em diversas instâncias da cadeia de comunicação a fim de avaliar: (1) a possibilidade de substituir um ou mais blocos dos sistemas clássicos, principalmente na camada física; e (2) a abordagem instigante de uma eventual substituição de todos os blocos da camada física visando uma otimização ponta-a-ponta.

Dadas essas problemáticas, o objetivo deste trabalho é desenvolver uma avaliação comparativa entre os métodos clássicos de modulação e codificação de canal com o método de DL, utilizando especificamente a arquitetura de Autoencoder com redes neurais profundas (DNN) no transmissor e receptor, na referida abordagem de substituição dos principais blocos da camada física. Em relação aos artigos-base deste trabalho - (O'SHEA; HOYDIS, 2017) e (ERPEK *et al.*, 2020) - algumas configurações diferentes foram avaliadas, bem como os sistemas convencionais utilizados são de certo modo diferentes. Como parte da curva de aprendizado em relação ao tema, buscou-se também reproduzir um dos resultados de (O'SHEA; HOYDIS, 2017). Este trabalho tem seu escopo limitado a cenários com um único usuário, sem informação do estado do canal e utilizando o modelo de canal AWGN (*Additive White Gaussian Channel*) [2].

1.2 Trabalhos relacionados

Ao que se tem registro, as primeiras avaliações de um sistema de comunicação com bloco único utilizando DL foram feitas em (O'SHEA; HOYDIS, 2017) e (O'SHEA *et al.*, 2016). Os autores utilizam a arquitetura de Autoencoder para otimizar a transmissão e recepção como um proceso ponta-a-ponta, sem separação em blocos. A avaliação é feita basicamente com a BLER, onde os autores mostram que numa comparação com sistemas BPSK com código Hamming o desempenho com DL é bastante comparável e até melhor em determinadas configurações. Esses artigos são muito relevantes para este trabalho e serão tomados como base para os resultados aqui encontrados. No mesmo artigo os autores extendem o trabalho para o conceito de redes adversárias visando a avaliação em cenários multi-usuário e MIMO (Múltiplas-Entradas Múltiplas-Saídas) também. Isso foge do escopo deste presente trabalho, por isso é suficiente mencionar.

[2] Código-fonte utilizado disponível em https://github.com/caio-lucas07/Deep-Learning-PHY.

Dada a promissora arquitetura em Autoencoder, os autores em (DÖRNER *et al.*, 2018) lidam com os desafios da implementação em *hardware* de um sistema de comunicação totalmente baseado nessa arquitetura. Neste relevante trabalho os autores demonstram a possibilidade de modelar, construir e executar sistemas baseados em técnicas de DL, em especial as redes neurais. Algumas dificuldades inerentes a esse trabalho de implementação são contornadas, como a questão de sincronismo, a dificuldade com o modelo de canal a ser utilizado durante a fase de treinamento das redes, onde neste caso a técnica de Aprendizado por Transferência (*Transfer Learning*) é utilizada para condicionar as redes ao canal real; a questão dos efeitos do *offset* de frequência das portadoras, entre outras questões. E como os autores dizem, o artigo ergue mais perguntas do que provê respostas.

Em (ERPEK *et al.*, 2020) os autores reiteram o trabalho feito em (O'SHEA; HOYDIS, 2017) para comunicação ponta-a-ponta, bem como estendem a aplicação de DL para outras áreas como estimação de canal, gerenciamento de espectro e segurança em redes wireless.

Alguns outros trabalhos discutem novas arquiteturas, novos métodos, múltiplos cenários e tipos de informação que fogem do escopo relacionado a este trabalho, mas que de alguma forma contribuíram para a escrita do mesmo. Em especial, em (BOURTSOULATZE *et al.*, 2019) os autores propõem uma técnica de codificação conjunta fonte-canal para transmissão de imagens com redes neurais convolucionais (CNN) na arquitetura Autoencoder. Em (KURKA; GÜNDÜZ, 2019) o mesmo método, porém com uma arquitetura contendo *feedback*, é usado para transmissão de imagem. Esses trabalhos exploram bastante o desempenho da arquitetura Autoencoder em cenários como o de canal com desvanecimento Rayleigh, bem como fazem comparações com esquemas de modulação e codificação convencionais que operam satisfatoriamente, como os códigos Turbo e LDPC (Checagem de Paridade de Baixa Densidade). Os trabalhos constituem uma boa fonte para acesso a outras literaturas e materiais relevantes, além de serem um bom ponto de partida para futuros trabalhos em transmissão de imagens em canal sem fio.

1.3 Estrutura do trabalho

A cadeia clássica de um sistema de comunicação digital é apresentada na Seção 2.1. Os elementos da cadeia em questão são explanados brevemente para um melhor entendimento dos sistemas convencionais (que também chamaremos de *baselines*). A Seção 2.2 apresenta os esquemas de modulação e codificação clássicos utilizados no Capítulo 3.

Na Seção 2.3 são contemplados alguns conceitos do tópico de aprendizado profundo (*Deep Learning*), o campo da inteligência artificial, no contexto do qual algumas técnicas são utilizadas neste trabalho. Uma visão geral dos conceitos mais básicos é apresentada na Subseção 2.3.1, logo depois na Subseção 2.3.2 os *Autoencoders* são explanados em um nível um pouco maior de detalhes, pois são o núcleo deste trabalho; a Seção 2.3.3 discorre sobre o potencial dos *Autoencoders* como um modelo para viabilizar a comunicação digital ponta-a-ponta; por último, na Subseção 2.3.4 apresenta-se brevemente sobre duas das principais bibliotecas (ou *frameworks*, como queira) usadas no desenvolvimento de modelos de aprendizado profundo, *Keras* e *Tensorflow*.

O Capítulo 3 apresenta os resultados deste trabalho com as devidas discussões. E por fim, o Capítulo 4 reitera as principais conclusões dos resultados e propõe direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo trata de alguns conceitos fundamentais para o entendimento dos aspectos puramente téoricos concernentes aos campos de estudo em sistemas de comunicação digital e inteligência artificial. Tais aspectos são necessários para a construção da base de conhecimento que permite uma melhor compreensão do impacto da inteligência artificial nas comunicações digitais, especialmente, neste trabalho, na camada física da cadeia de comunicação.

O uso da palavra "alguns" no parágrafo anterior não é meramente acidental. O capítulo não pretende ser exaustivo, uma vez que os campos de estudo são, em si, muito vastos. Ainda assim, o leitor poderá notar ao longo deste trabalho uma série de referências que poderão lhe direcionar para trabalhos de cada tópico apresentado e poderá, portanto, aprofundar nos aspectos mais teóricos, caso seja de interesse.

2.1 Sistema de comunicação ponta-a-ponta

A Figura 1 ilustra um sistema de comunicação digital (doravante, SCD) em um aspecto mais funcional, através dos seus elementos mais básicos.

A fonte de informação (que pode ser analógica ou digital) consiste em uma ou várias mensagens que podem assumir qualquer natureza: texto, áudio, vídeo, entre outras. Em um SCD, essas mensagens provenientes da fonte de informação são convertidas em uma sequência de dígitos chamados *bits*. O intuito, portanto, é enviar os *bits* do transmissor, passando pelo canal e chegando ao receptor, a fim de que este consiga utilizar a informação como deseja (PROAKIS, 1995).

O grande desafio é, portanto, enviar os *bits* de informação para o receptor por um canal que, geralmente, é ruidoso, principalmente quando se fala em canal sem fio. Nesse momento, entra a grande utilidade dos outros blocos da cadeia do SCD, que têm seus objetivos e funcionalidades bem específicas para garantir que a informação chegue ao receptor de forma satisfatória (CAVALCANTI *et al.*, 2018).

O codificador de fonte recebe o conjunto de *bits* de mensagem e tenta representá-los da maneira mais eficiente possível, os comprimindo com ou sem perdas, retirando toda e qualquer redundância desnecessária da mensagem, a fim de que o canal seja usado de forma eficiente (PROAKIS, 1995; HAYKIN, 2014).

O fluxo de bits que sai do codificador de fonte alimenta o bloco do codificador de

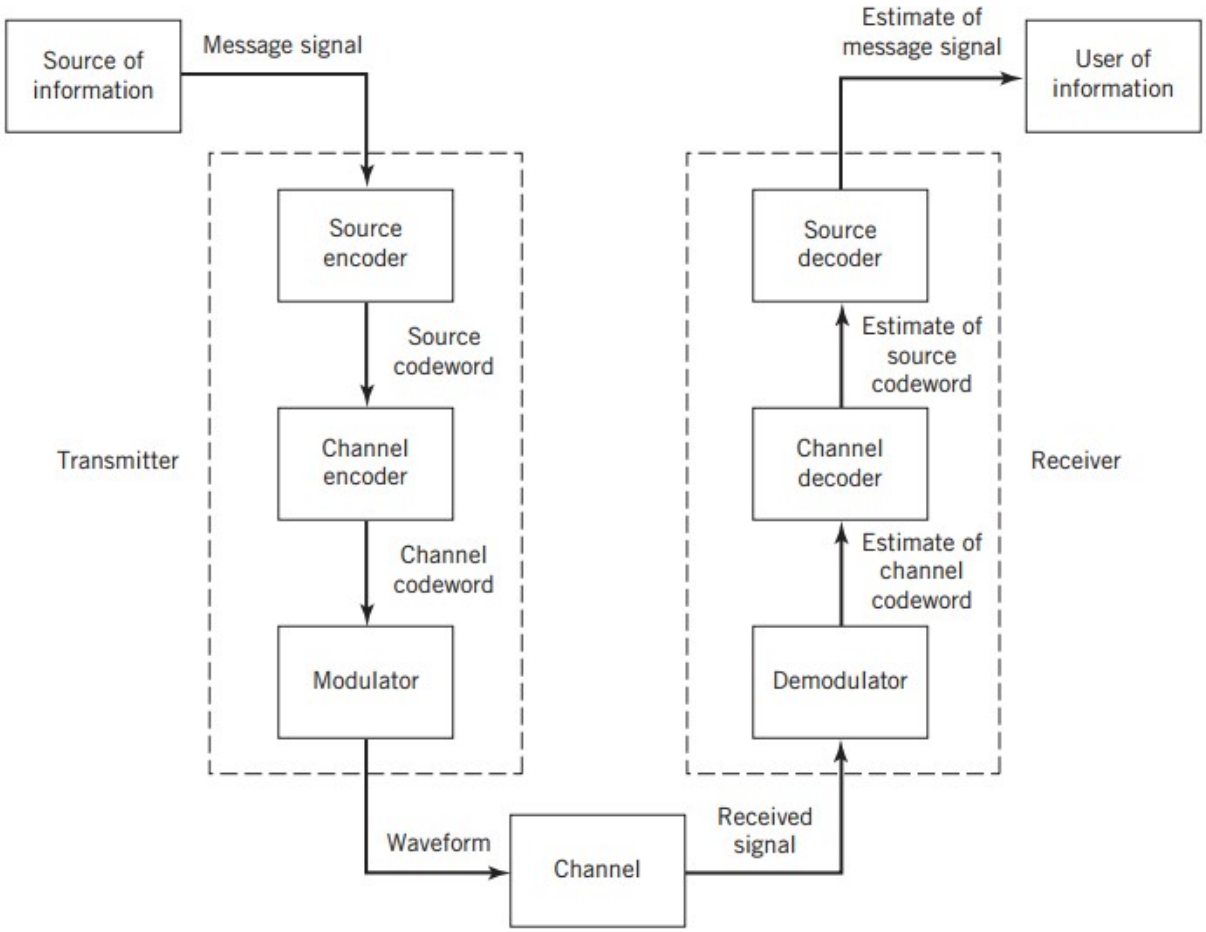


Figura 1 – Diagrama funcional de um sistema de comunicação digital.
Fonte: (HAYKIN, 2014)

canal, que é responsável por garantir a confiabilidade da comunicação, antes da informação ser enviada pelo canal. O objetivo é fazer com que a exata informação enviada seja identificada satisfatoriamente pelo receptor, mesmo após ter sido distorcida pelo canal. O codificador de canal faz isso geralmente adicionando redundância (bits extras) à mensagem, que é exatamente o contrário em relação ao que o codificador de fonte faz (CAVALCANTI *et al.*, 2018).

Logo após o codificador de canal, a informação ainda em formato binário entra no bloco de modulação, ou modulador. Este é responsável por mapear os bits em sinais como formas de onda eletromagnética, a serem enviados através do canal, em direção ao receptor. As formas de onda podem representar 1 bit apenas (bit 0 ou bit 1), no caso de modulação binária, ou podem representar um conjunto de bits, chamada de modulação *M-ária* (PROAKIS, 1995).

É importante mencionar que, dada uma sequência de m bits, cada sequência entra no modulador a cada m/R segundos, onde R é a taxa em bit/s. Logo, para modulação binária e uma taxa fixada, o tempo para transmitir uma das 2^m formas de onda é igual a m vezes o período T (ou tempo de bit), em segundos (PROAKIS, 1995).

Enviadas as formas de onda, estas passam pelo canal, o meio de tansmissão, que pode ser um fio, uma fibra óptica, o espaço livre ("ar"), ou mesmo uma combinação destes. O canal, por sua vez, geralmente causa distorções nos sinais, de maneira por vezes aleatória, fazendo com que o todo o processo de transmissão e recepção seja projetado para reverter esses efeitos. Os efeitos podem ser tanto da própria natureza do canal em si, bem como efeitos produzidos pelo próprio homem ou pelos dispositivos utilizados para viabilizar esse processo de transmissão. Além do mais, os efeitos de interferência também podem causar distorções, que é quando outros dispositivos executando o mesmo processo de transmissão acabam por atrapalhar os outros, em virtude dos fenômenos ondulatórios de interferência, massivamente presentes em grandes centros urbanos, por exemplo.

Quando as formas de onda chegam no receptor, todo o processo feito para enviar a informação é revertido e aplicado aos sinais, a fim de obter a mensagem mais próxima possível da que foi enviada. Esse processo consiste, a grosso modo, em três subprocessos também. São eles: a demodulação, a estimação dos sinais codificados para o canal e estimação dos sinais codificados na fonte.

A demodulação é o processo de reduzir as formas de onda em números que representam estimativas dos símbolos transmitidos. Com o conhecimento que o receptor tem do processo de codificação que foi feito nos bits transmitidos, o mesmo tenta reverter esse processo retirando os bits redundantes adicionados pelo codificador de canal e na sequência tenta reconstruir os símbolos que sofreram a compressão do codificador de fonte (PROAKIS, 1995).

Todo o processo é executado de forma a obter a réplica mais fidedigna da mensagem transmitida. No entanto, sempre haverá uma parcela de erro presente. A Taxa de Erro de Bit (doravante, BER, do inglês *Bit Error Rate*) é uma das formas mais comuns de medir esse erro. Ela consiste na probabilidade média de 1 bit estar errado dada a sequência recebida na saída do receptor, por unidade de tempo. Como um exemplo prático, para transmissão de mensagem de voz, a BER na entrada do decodificador de fonte precisa estar entre 10^{-3} e 10^{-7} para que a voz seja reconstruída eficientemente (CAVALCANTI *et al.*, 2018).

Outra métrica equivalente à BER também utilizada é a Taxa de Erro de Bloco (BLER, do inglês Block Error Rate), que inclusive é uma função da BER. Ela é dada pela seguinte equação:

$$BLER = 1 - (1 - BER)^{n_b}, \tag{2.1}$$

onde n_b é a quantidade de bits por bloco de informação.

Ela é definida como a taxa dos blocos recebidos em erro. Um bloco recebido em erro consiste em um bloco que contém pelo menos 1 bit em erro.

Essas duas métricas mencionadas serão utilizadas durante as avaliações dos resultados deste trabalho.

Canal AWGN

Outros tópicos importantes que explicaremos aqui em linhas gerais (deixamos o aprofundamento para o leitor) são os conceitos de modelo de canal com Ruído Aditivo Gaussiano Branco (doravante, AWGN) e a medida de desempenho Relação Sinal-Ruído (SNR, do inglês *Signal-to-noise Ratio*). O modelo em questão foi utilizado em todas as simulações deste trabalho, e será importante para a Seção 2.2, onde será tomado como um pressuposto de modelo de canal para a explicação dos esquemas de modulação e as formas de codificação utilizadas neste trabalho. A SNR também é utilizada para todas as simulações e a explicaremos em seguida.

O modelo de canal AWGN é retratado na Figura 2. Suponha a informação de entrada no somador sendo X_i, o ruído gaussiano definido por Z_i e a saída do sistema Y_i.

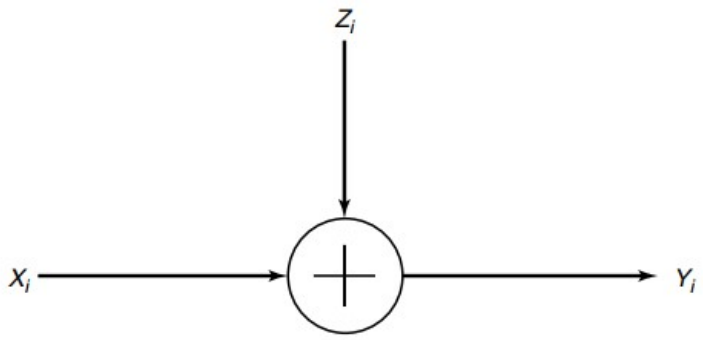


Figura 2 – Modelo simplificado de canal gaussiano.
Fonte: (COVER; THOMAS, 2006)

O ruído gaussiano Z_i é definido, em geral, como uma sequência de variáveis aleatórias independentes e identicamente distribuídas com média igual a zero e variância igual a σ^2. Ele é caracterizado por ser *aditivo*, pois é somado ao sinal; *branco*, pois possui componentes de energia em toda a banda de frequência do sinal; e *gaussiano*, por ter uma distribuição gaussiana no domínio do tempo.

O canal AWGN modela alguns dos vários canais de comunicação nas aplicações reais, como canais com fio, canais sem fio e canais de comunicação via satélite. Se sua variância é zero, o receptor recebe os símbolos transmitidos sem erro. Se a entrada do canal não contém

restrição alguma, a capacidade do canal se torna infinita, o que não é factível em termos práticos (COVER; THOMAS, 2006).

Por isso, quando se usa o canal AWGN, geralmente há uma restrição a ser satisfeita na sequência que vai ser transmitida, chamada de *restrição média de energia* ou *restrição média de potência*. Dada uma sequência de símbolos $(s_1, s_2, ..., s_n)$[1] esta restrição é dada por:

$$\frac{1}{n}\sum_{i=1}^{n} s_i^2 \leq P, \tag{2.2}$$

onde P é a máxima potência do canal.

Essa restrição é uma das que foram utilizadas durante o processo de geração dos resultados deste trabalho.

SNR

Claude Shannon, em (SHANNON, 1948), estabeleceu os fundamentos matemáticos para transmissão de informação e deduziu os limites fundamentais dos sistemas de comunicação digital. Nesse trabalho, Shannon formulou o problema da transmissão de informação com confiabilidade a partir de uma perspectiva estatística, usando modelos probabilísticos para fontes de informação e canais de comunicação. Com essa formulação estatística, ele adotou uma medida logarítmica para representar o conteúdo de informação de uma fonte (posteriormente mais conhecida como *entropia*). Com isso, ao desenvolver a teoria, ele demonstrou que o efeito de uma restrição de potência no transmissor, uma restrição de largura de banda e um ruído aditivo, pode ser associado com o canal de comunicação e associado a um parâmetro apenas, chamado *capacidade de canal* [2] (PROAKIS, 1995).

Para o caso do canal AWGN, por exemplo, a sua capacidade, assumindo que ele é limitado em banda, pode ser dada pela seguinte expressão:

$$C = B\log_2\left(1 + \frac{E}{BN_0}\right) \quad [\text{bit/s}], \tag{2.3}$$

onde B é a largura de banda do canal, E é a potência média de transmissão e N_0 é a densidade espectral de potência do ruído aditivo gaussiano (PROAKIS, 1995).

Assim, convencionou-se chamar o logaritmo (na base 10) da razão que está dentro do argumento do logaritmo de base 2 de *relação sinal-ruído*, ou SNR. A expressão abaixo mostra

[1] Para mais detalhes sobre os conceitos de codificação, cf. Seção 2.2.

[2] Capacidade de canal pode ser descrita em linhas gerais como a taxa mais alta em bits por uso de canal na qual as informações podem ser enviadas com probabilidade de erro arbitrariamente baixa (COVER; THOMAS, 2006).

a definição da SNR em decibel[3]:

$$SNR = 10\log_{10}\left(\frac{E}{N_0}\right) \quad [\text{dB}]. \tag{2.4}$$

Ela pode ser descrita como a relação entre a potência média da energia do sinal transmitido e a potência do ruído. É considerada uma medida de desempenho, pois nos diz qual a influência do ruído no sinal de informação. Essa medida é avaliada geralmente em decibéis (dB) e é obtida preferencialmente no receptor, a fim de estimar o estado do canal de comunicação no que diz respeito à presença de ruído.

2.2 Modelo de blocos clássico

Nesta sub-seção são explanados em linhas gerais alguns dos esquemas de modulação convencionais utilizados em SCD, bem como o tópico de codificação de canal.

O tópico de codificação de canal será brevemente abordado, em uma perspectiva mais *en passant*, discutindo as motivações principais dessa teoria, bem como alguns aspectos dos códigos lineares de Hamming e os códigos Convolucionais (CC). O que será visto pelo leitor tem como base majoritária as literaturas introdutórias de (HAYKIN; MOHER, 2009) (capítulo 10), (HAYKIN, 2014) (capítulo 10) e (CAVALCANTI *et al.*, 2018) (capítulo 6). Para leituras com uma maior riqueza de detalhes, pode-se consultar (SWEENEY, 2002) (capítulos 2 e 3) e (SKLAR, 2001) (capítulos 6 a 8).

Esquemas de modulação BPSK, QPSK e QAM

Os processos de modulação se dão quando, ao receber os bits do codificador de canal, estes são mapeados em sinais que são modelados como ondas eletromagnéticas. Como todo modelamento de onda, estas possuem uma amplitude A, uma fase ϕ e um comprimento de onda λ. Com o comprimento de onda calcula-se a frequência f da onda com $f = c/\lambda$, onde c é a velocidade da luz no vácuo = $3 \cdot 10^8$ m/s.

A mágica da modulação ocorre de fato quando, dependendo do número de bits na sequência vinda do codificador de canal, os sinais [4] têm os seus parâmetros variados (comutados, chaveados) para representar algum símbolo de informação. Um símbolo pode conter 1 bit ou uma sequência de m bits. A quantidade de símbolos define qual o tipo de modulação. Se binária,

[3] A definição da SNR pode ser dada em escala linear também como a razão simples entre a potência média do sinal transmitido e a potência média do ruído.

[4] Entenda *sinais*, doravante, como um modelamento matemático das ondas eletromagnéticas.

a modulação possui 2 símbolos, com 1 bit em cada um deles. Se *M-ária*, possui *M* símbolos, com uma quantidade definida de *m* bits em cada símbolo.

BPSK

O esquema de modulação BPSK (do inglês, *Binary Phase Shift Keying*), por exemplo, é um tipo de modulação binária. Nesse esquema, um sinal (chamado comumente de *portadora senoidal*) tem sua amplitude e frequência fixas, e sua fase é deslocada de 180 graus para representar o símbolo 0 (ou s_0) [5]. A Figura 3 mostra as três principais formas de sinalização para transmissão de informação binária: (a) ilustra um chaveamento de amplitude, (b) um chaveamento de frequência e (c) um chaveamento de fase. O caso BPSK está ilustrado na Figura 3c (HAYKIN; MOHER, 2009).

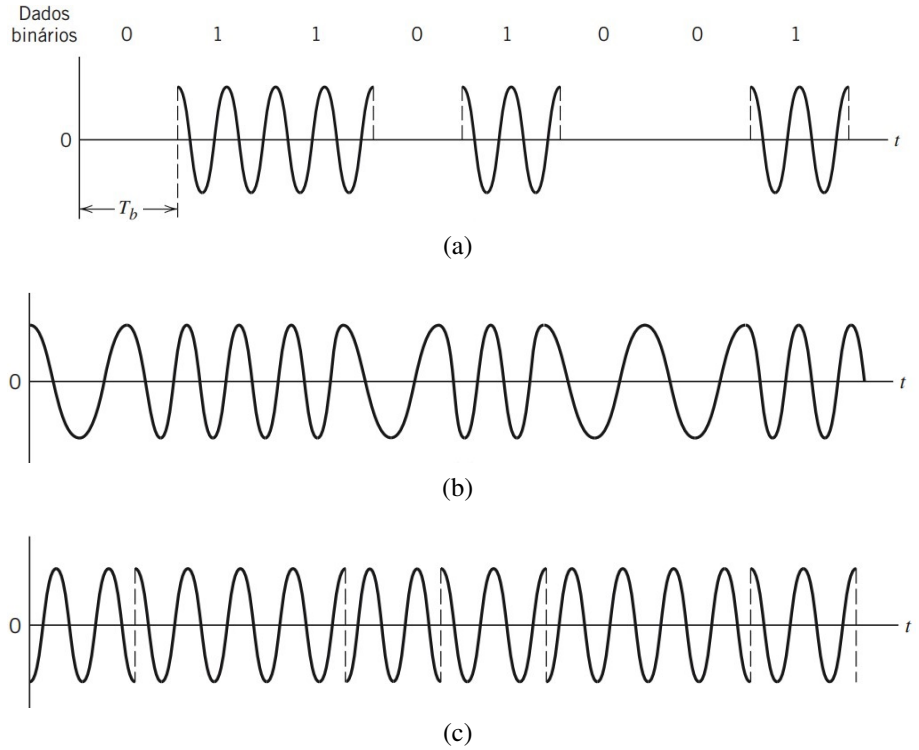


Figura 3 – Formas básicas de sinalização binária.
Fonte: (HAYKIN; MOHER, 2009).

Matematicamente, um sinal BPSK é dado pelas seguintes equações:

$$s_1 = A\,cos(2\pi ft) \tag{2.5}$$

$$s_0 = -A\,cos(2\pi ft) = A\,cos(2\pi ft + \pi) \tag{2.6}$$

[5] Adotaremos a notação s_i para representar um símbolo, onde i simboliza o i-ésimo símbolo.

A Equação (2.5), para s_1, representa o símbolo 1 e a Equação (2.6(, para s_0, representa o símbolo 0, que é deslocado de 180 graus em sua fase (ao argumento é somado π).

Os símbolos de uma modulação geralmente são retratados geometricamente numa estrutura chamada *constelação*. Alguns exemplos de constelação podem ser vistos na Figura 4. Da esquerda para a direita, na Figura 4a, tem-se as modulações ASK (Chaveamento por Deslocamento de Amplitude, do inglês *Amplitude Shift Keying*) binária, BPSK e 4-ASK; da mesma forma, na Figura 4b tem-se 16-QAM (Modulação de Amplitude em Quadratura, do inglês *Quadrature Amplitude Modulation*); 4-QAM e QPSK (Chaveamento por Deslocamento de Fase em Quadratura, do inglês *Quadrature Phase Shift Keying*); e 8-PSK.

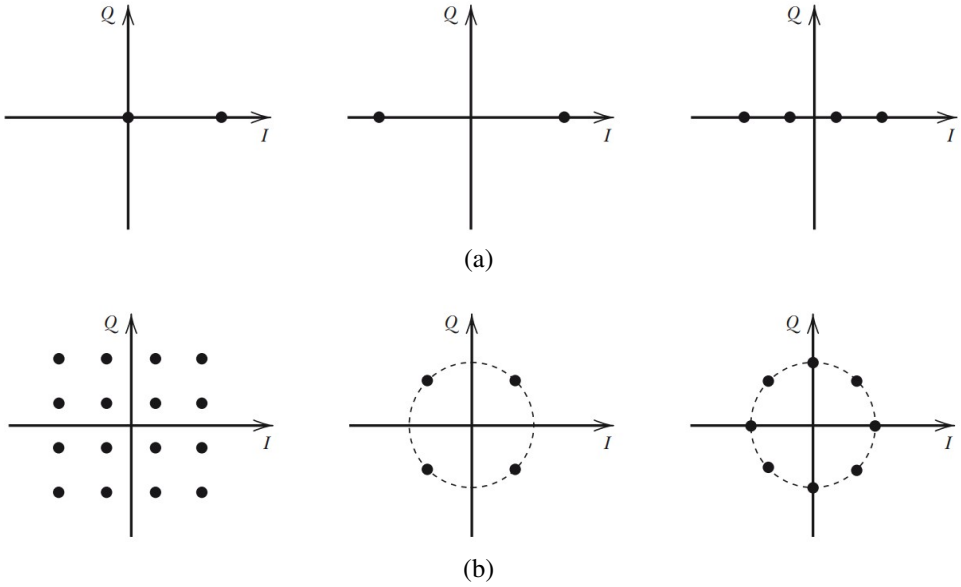


Figura 4 – Exemplos de constelações de vários esquemas de modulação.
Fonte: (HAYKIN; MOHER, 2009).

Comparado com outros esquemas M-PSK, os símbolos BPSK são mais tolerantes a ruído, pois eles estão mais separados na constelação, como se vê na Figura 4a, o que diminui erros de interpretação no receptor quando este detecta um símbolo. Por esse motivo essa é a modulação PSK mais robusta. Em contrapartida, o BPSK perde em taxa de comunicação, uma vez que se tem poucos bits transmitidos por unidade de tempo.

No receptor, a demodulação pode se dar, dentre outras formas, com detecção coerente ou não coerente. A explicação com detalhes deixamos como consulta para o leitor, em (HAYKIN; MOHER, 2009), no Capítulo 9. Aqui, só apresentamos a BER média P_b da modulação BPSK para detecção coerente, dada por:

$$P_b = \frac{1}{2}\text{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right), \tag{2.7}$$

onde a função erro erfc$(.)$ é dada por:

$$\text{erfc} = \frac{2}{\sqrt{\pi}} \int_{x=0}^{\infty} e^{-t^2}\, dt. \tag{2.8}$$

Essas equações podem ser calculadas para a ratificação de que uma simulação de BPSK esteja coerente com a teoria.

QPSK

A modulação QPSK tem constelação semelhante à da modulação 4-QAM, ambas usadas neste trabalho como sistemas *baseline*. A informação dos símbolos da modulação QPSK, assim como na BPSK, está na fase ϕ da portadora. A fase da portadora pode assumir um dos 4 valores a seguir: $\pi/4$, $3\pi/4$, $5\pi/4$ e $7\pi/4$. Para esses valores, os símbolos da modulação podem ser da seguinte forma:

$$s_i = \begin{cases} \sqrt{\frac{2E}{T}} \cos\left(2\pi f t + (2i-1)\frac{\pi}{4}\right), & 0 \leq t \leq T,\ i = 1,2,3,4 \\ 0, & \text{c.c.}, \end{cases} \tag{2.9}$$

onde E é a energia por símbolo transmitido e T a duração do símbolo. A Figura 5 ilustra os bits correspondentes aos quatro símbolos da modulação QPSK.

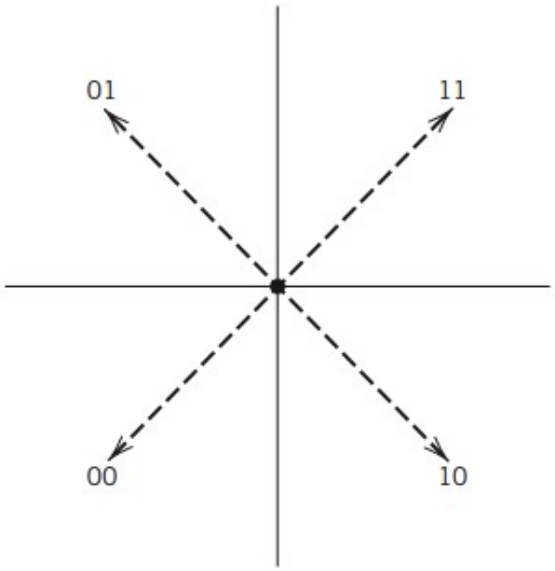


Figura 5 – Quatro valores de fases da modulação QPSK com os bits correspondentes.
Fonte: (HAYKIN; MOHER, 2009).

Assim como para o esquema BPSK, a demodulação do QPSK é deixada para o leitor consultar em (HAYKIN; MOHER, 2009), no Capítulo 9 da referência.

A BER média da modulação QPSK é dada por:

$$P_b = \frac{1}{2}\text{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right). \tag{2.10}$$

A expressão é a mesma para a modulação BPSK. Isso significa que na modulação QPSK, numa dada largura de banda de frequência, pode-se transmitir o dobro de bits em relação ao BPSK coerente, com o mesmo desempenho de BER média. Contudo, vale ressaltar que isso não significa dizer que a modulação QPSK é sempre melhor que a modulação BPSK. Outros fatores estão envolvidos nessa comparação. Um dos tais é o fato de que o demodulador QPSK exige um circuito de recuperação de portadora mais sofisticado do que o do BPSK (HAYKIN, 2014).

QAM

Os sistemas QAM são esquemas utilizados tanto na modulação de sinais analógicos quanto de sinais digitais. Podem ser vistos como uma generalização do QPSK (HAYKIN, 2014). Algumas aplicações da modulação QAM são: TV digital, rádio digital, modem ADSL, entre outras.

Na modulação QAM, tanto a fase como a amplitude da portadora variam de acordo com a informação a ser transmitida. É um tipo de modulação M-ária (assim como a QPSK), portanto, possui mais de dois símbolos. Em geral, $M = \{4, 16, 64, 256\}$.

O sinal transmitido para o símbolo s_i é representado como se segue (PROAKIS, 1995):

$$s_i(t) = \sqrt{\frac{2E}{T}}\, a_i \cos(2\pi f t) - \sqrt{\frac{2E}{T}}\, b_i \,\text{sen}(2\pi f t), \tag{2.11}$$

onde a_i e b_i são elementos de um par ordenado da constelação QAM. No caso de $M = 16$, tem-se a constelação ilustrada na Figura 4b, primeira figura.

Para o caso em que a constelação é quadrada, o número de bits por símbolo é par. A depender de M, pode-se ter diferentes tipos de constelação, onde $M = 2^m$ e m é o número de bits em cada símbolo. Na Figura 6 é ilustrada a constelação da modulação 16-QAM com os respectivos bits que representam cada símbolo, na codificação Gray.

A taxa de erro de símbolo [6] da modulação QAM é dada pela seguinte equação (HAYKIN, 2014):

$$P_s \approx 2\left(1 - \frac{1}{\sqrt{M}}\right)\text{erfc}\left(\sqrt{\frac{E_0}{N_0}}\right), \tag{2.12}$$

[6] Taxa de erro de símbolo é definida como o número de símbolos em erro sobre o número de símbolos recebidos.

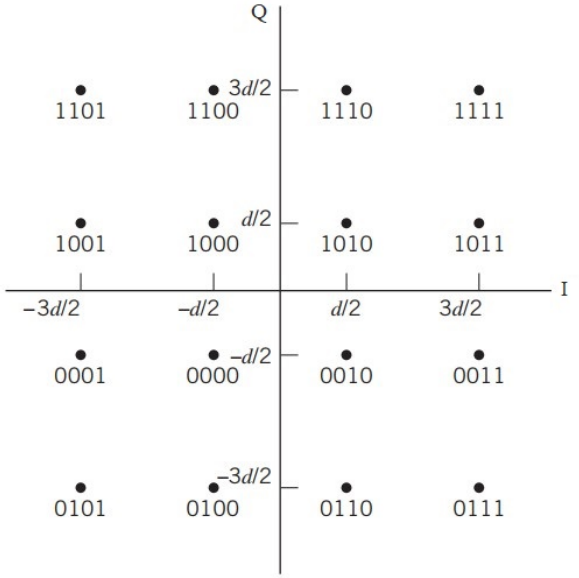


Figura 6 – Constelação para modulação 16-QAM.
Fonte: Adaptado de (HAYKIN, 2014).

onde E_0 é a energia do símbolo com mais baixo valor de amplitude.

Codificação de Canal

Em suma, a codificação de canal tem por objetivo tornar o sinal transmitido mais resiliente aos efeitos do ruído do canal. Todo o processo realizado com este objetivo permite que o receptor consiga detectar e até corrigir os erros que surgiram durante a transmissão dos sinais modulados. Como mencionado na Seção 2.1, em termos mais claros, o que ocorre é uma adição de bits extras (redundantes) aos bits do sinal, mapeando esse sinal para um espaço de maior dimensão, de tal forma que a distância entre os pontos da constelação seja maior e, assim, mesmo na presença de ruído, a informação (mensagem) consiga ser reconhecida.

Pensando em conceitos preliminares da teoria de codificação que ficarão patentes na seção de resultados com codificação deste trabalho, dentre eles tem-se o de *ganho de codificação*. Ele consiste na redução de SNR que se atinge ao adicionar um codificador de canal em um SCD, para uma dada BER ou BLER como alvo. Esse conceito é simples, mas importante para uma avaliação rápida de um gráfico de BER por SNR. É importante mencionar que ganhos de codificação podem ser até negativos. Isso ocorre geralmente quando os códigos são projetados para regimes de alta SNR: em altas SNRs (canal com melhor condição) o desempenho é satisfatório, mas em regimes de baixa SNR (canal mais ruidoso) o desempenho é mais degradante.

Outro ponto relevante para uma introdução à teoria de codificação de canal trata-se

do *teorema da codificação de canal*. Shannon, em (SHANNON, 1948), identifica e formula a capacidade de canal como o limite teórico para que uma transmissão com codificação possa ser feita com confiabilidade, de tal maneira que o sinal transmitido seja recebido e a mensagem reconhecida com uma probabilidade de erro arbitrariamente pequena. Não é um teorema onde se apresenta qual é o melhor e mais adequado esquema de codificação que pode ser usado, nem toca na questão do cálculo de probabilidade de erro na sequência decodificada. O teorema discute apenas os limites teóricos e postula que a probabilidade de erro diminui com o aumento do tamanho do código. O teorema, além disso, diz que a capacidade corretora de erros aumenta com códigos mais longos. Apesar disso, contudo, pode-se verificar que códigos longos exigem sistemas com decodificação mais complexa, que incorrem em maior latência e atraso de processamento (CAVALCANTI *et al.*, 2018). Esses são alguns aspectos importantes a serem levados em consideração nos sistemas clássicos e em eventuais sistemas baseados em DL também.

Código Hamming

O código Hamming faz parte da família dos códigos lineares em bloco e foi inventado por Richard W. Hamming em 1950 [7]. Um código em bloco é dado por um conjunto de palavras-código de n bits que é gerado a partir de k bits de informação, onde $n > k$. Os $n-k$ bits extras adicionados são chamados de bits de paridade. Um código em bloco é dito linear se a soma de uma palavra-código com outra levar a uma palavra-código válida. A aritmética da soma é definida pelo alfabeto em questão: se binário, a soma será em aritmética módulo-2. Esses códigos geralmente são representados pela notação (n,k). A taxa de codificação é dada, portanto, por $R = k/n$. Ela expressa uma métrica quantitativa do impacto da adição de bits extras em relação à estratégia de codificação adotada.

Dentre outras formulações usadas para construir palavras-código tem-se uma das mais usadas que é a matricial. Essa formulação gira em torno das matrizes **H** (chamada matriz de paridade) e **G** (chamada matriz geradora). Para o caso do código (7,4), por exemplo, tem-se que:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

[7] Artigo original em (HAMMING, 1950).

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}, \tag{2.14}$$

Para $k = 4$ bits, é possível gerar $2^k = 16$ palavras distintas e que são obtidas a partir de $\mathbf{v} = \mathbf{uG}$, onde $\mathbf{u}$ é a mensagem enviada. Para a mensagem $\mathbf{u} = [1011]$, por exemplo, a palavra-código é dada por $\mathbf{v} = \mathbf{uG} = [1011000]$. A decodificação pode ser feita através do que se chama de *síndrome* e os padrões de erro associados a cada uma das síndromes. Essas são informações que o receptor tem conhecimento. De posse desse conhecimento, o decodificador pode identificar o erro na palavra que ele recebeu, e então reverter o bit em erro [8]. A Tabela 1 mostra as síndromes e os padrões de erro corrigíveis para o código Hamming (7,4).

Tabela 1 – Síndromes e padrões de erro corrigíveis para Hamming (7,4).

Síndrome	Padrão de erro
0 0 0	0 0 0 0 0 0 0
1 1 0	1 0 0 0 0 0 0
1 1 1	0 1 0 0 0 0 0
1 0 1	0 0 1 0 0 0 0
0 1 1	0 0 0 1 0 0 0
1 0 0	0 0 0 0 1 0 0
0 1 0	0 0 0 0 0 1 0
0 0 1	0 0 0 0 0 0 1

Suponha que o decodificador recebeu a palavra-código $\mathbf{c} = [1111000]$. Isso significa que o segundo bit foi recebido em erro (é evidente que essa informação o receptor ainda não tem). A síndrome será dada por $\mathbf{s} = \mathbf{cH}^T = [111]$. Olhando para a Tabela 1, vê-se que essa síndrome corresponde ao padrão de erro [0100000]. Logo, a palavra-código estimada será [1011000] que decorre de reverter o segundo bit da palavra-código recebida, e que é exatamente a palavra-código correta.

Códigos Convolucionais

Em teoria, os códigos convolucionais atuam sobre um fluxo contínuo de dados. Contudo, na prática, são códigos em bloco assim como Hamming, com a diferença importante

[8] É importante mencionar que o código Hamming só é capaz de corrigir padrões de erro de 1 bit apenas.

que são códigos com memória, ou seja, as palavras-código geradas dependem não somente das entradas atuais do codificador, mas também das entradas passadas.

Como mencionam os autores em (CAVALCANTI *et al.*, 2018), um código convolucional executa o processo de geração de palavras-código ao fazer passar os bits de informação por um registrador de deslocamento de estado linear e finito. O registrador de deslocamento consiste em K estágios com k bits cada. Uma palavra-código de comprimento n é gerada para cada sequência de k bits de informação e $m-1$ mensagens anteriores. Portanto, um código (n,k,K) relaciona a mensagem atual com $m-1$ mensagens anteriores. O parâmetro K representa a profundidade de memória (comprimento de restrição) do código e especifica o número de estágios de memória do codificador (flip-flops) utilizados para armazenar informação de estado. Ou seja, ele é definido como o número de deslocamentos ao longo dos quais um único bit de mensagem pode influenciar a saída do codificador. A Figura 7 mostra um codificador convolucional com $n = 2$ e $K = 3$, em que opera um bit por vez ($k = 1$). Note que para um bit na entrada chegar ao final do codificador precisa ser deslocado $K = 3$ vezes.

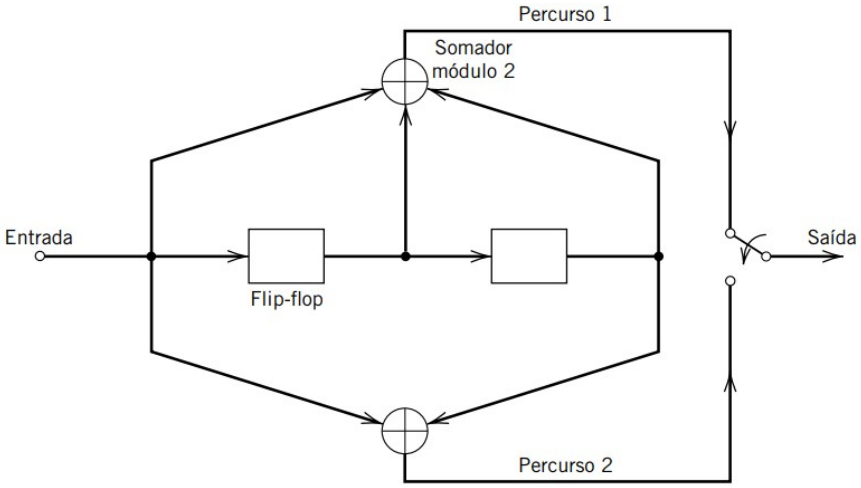


Figura 7 – Codificador convolucional com comprimento de restrição 3 e taxa 1/2.
Fonte: (HAYKIN; MOHER, 2009).

Para o caso da Figura 7 o processo de codificação pode ser pensado da seguinte forma: imagine a entrada, o primeiro flip-flop (caixas vazias) e o segundo flip-flop como iguais a zero ("0"); agora, suponha uma mensagem $\mathbf{u} = [101001]$; como se vê na figura, a primeira saída do codificador (o percurso 1) é o resultado da soma módulo-2 entre a entrada, a saída do primeiro flip-flop e a saída do segundo flip-flop (note os somadores nas extremidades de cima e de baixo, bem como as setas que apontam para eles); agora pegue a mensagem e a desloque para dentro do codificador, executando as somas a cada único bit da mensagem que você deslocar;

lembre-se de que o primeiro passo, por exemplo, consiste em somar "1 $\oplus$ 0 $\oplus$ 0", onde o bit 1 corresponde ao primeiro bit mais à direita da mensagem **u**. Assim, desloque e some até não conter mais nenhum bit da mensagem dentro do codificador. Quando passarem a não restar mais os bits da mensagem na entrada do codificador, anexe zeros (sempre à esquerda). O resultado, para este caso, será uma palavra-código **c** = [11 01 00 01 11 11 01 11].

A decodificação de um código convolucional pode ser feita por máxima verossimilhança, através do *algoritmo de Viterbi*, que é um método baseado em treliça. A Figura 8 mostra uma treliça que ilustra o processo de decodificação do CC (2,1,3).

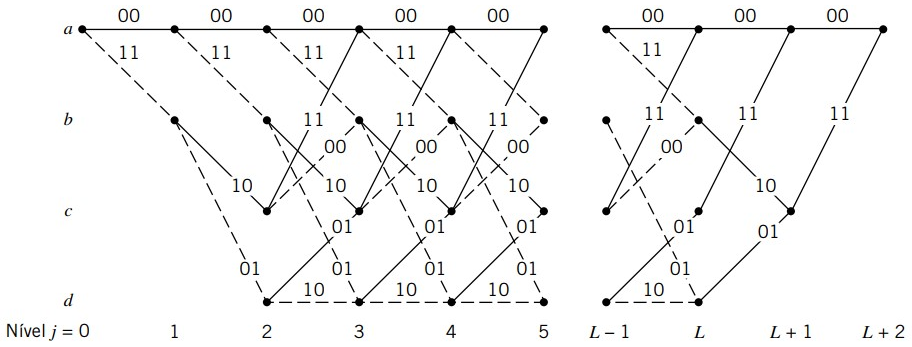


Figura 8 – Treliça para decodificação, por algoritmo de Viterbi, de CC (2,1,3).
Fonte: (HAYKIN; MOHER, 2009).

Como explicam os autores em (HAYKIN; MOHER, 2009), o algoritmo opera calculando uma métrica ou discrepância para cada percurso possível na treliça. A métrica para um percurso particular é definida como a distância de Hamming [9] entre a sequência codificada representada pelo percurso e a sequência recebida. Dessa forma, para cada nó (estado) na treliça da Figura 8, o algoritmo compara os dois percursos que chegam ao nó. O percurso com a menor métrica é mantido, e o outro percurso é descartado. O cálculo é repetido para cada nível j da treliça na faixa $M \leq j \leq L$, em que $M = K - 1$ é a memória do codificador e L é o comprimento da sequência de mensagem de entrada.

2.3 Aprendizado profundo

Ao longo da história dos sistemas de comunicação, surgiram muitas abordagens aos problemas inerentes da transmissão de informação à distância. Na seção anterior foi apresentada uma pequena parte da forma clássica de resolver alguns dos principais desafios e problemas dessa área que possui tantos bons sabores (e dissabores também).

[9] A distância de Hamming é dada pela quantidade de vezes em que duas mensagens (ou palavras-código) se diferenciam, elemento a elemento. E.g. a medida entre [101] e [111] é igual a 1.

Nesta presente seção, será delineado o conhecimento basilar para mais uma abordagem, em crescente expansão nos últimos anos, do desafio de comunicação sem fio ponta-a-ponta. Reitera-se que o que será delineado nas próximas subseções servirá somente para situar o leitor no tópico (vasto e extenso) de Aprendizado Profundo aplicado à comunicação ponta-a-ponta. Todas as referências usadas constituem as principais fontes de consulta do leitor para explanações mais detalhadas.

Um ponto importante consiste em que os assuntos apresentados serão, por vezes, direcionados para a área de comunicação sem fio ponta-a-ponta. Isso significa que algumas terminologias específicas serão adotadas. Para um melhor entendimento destas, recomenda-se as referências usadas nas Seções 2.1 e 2.2.

2.3.1 Overview

Aprendizado profundo é um termo não tão sugestivo, mas que tem impactado o mundo de forma impressionante há algum tempo. O campo de DL [10] é, na verdade, um subramo do campo de Aprendizado de Máquina [11] que, por sua vez, é um campo de estudo que essencialmente permite um algoritmo fazer predições, classificações, ou decisões baseadas na tentativa de reconhecer padrões e regularidades em dados observados, sem estar explicitamente programado para fazer isso (ZHANG *et al.*, 2019).

Alguns exemplos mais clássicos de ML incluem: regressão linear e logística, classificação por vizinhos mais próximos (*nearest neighbors*) e por mínimos quadrados (BISHOP, 2006), e *Q-learning*. São técnicas que têm forte apelo matemático e estatístico, e que, portanto, se provaram no tempo.

Os algoritmos de ML foram e ainda são muito salutares na resolução de diversos problemas do mundo real. Contudo, para algumas tarefas esses algoritmos falham em encontrar soluções ótimas. Uma limitação bastante pungente que os algoritmos de ML encontram é, por exemplo, a dificuldade de generalizar as predições (classificações ou regressões) para dados dispostos em espaços de alta dimensão, a chamada *maldição da dimensionalidade* (em inglês, *curse of dimensionality*), comumente apresentada nas literaturas de ML e DL (cf. (GOODFELLOW *et al.*, 2016), Seção 5.11.1).

Os algoritmos de DL têm se distinguido justamente por apresentarem desempenhos

[10] Doravante, adotar-se-á a abreviatura DL, do inglês *Deep Learning*, para se referir a *Aprendizado Profundo*.

[11] Doravante, adotar-se-á a abreviatura ML, do inglês *Machine Learning*, para se referir a *Aprendizado de Máquina*.

impressionantes em problemas que possuem essa natureza, bem como em diversos outros problemas de regressão, classificação, reconhecimento de padrões em geral, que exigem uma maior escalabilidade, modularidade e flexibilidade.

O tópico de DL tem sua origem há muitos anos. Sobretudo, às chamadas redes neurais artificiais (doravante, RNA) é que geralmente se atribui, na literatura, o início desse campo de estudo. Portanto, não será diferente neste trabalho. As redes neurais foram introduzidas pela primeira vez em 1943 pelo neurofisiologista Warren McCulloch e pelo matemático Walter Pitts, em seu artigo em (MCCULLOCH; PITTS, 1943). Neste trabalho, os autores apresentaram um modelo computacional simplificado utilizando lógica proposicional de como os neurônios podem trabalhar juntos em cérebros de animais na realização de cálculos complexos. Esta foi a primeira arquitetura de RNA. A partir desta, muitas outras foram sendo propostas (GÉRON, 2017). O modelo de McCulloch e Pitts está ilustrado na Figura 9.

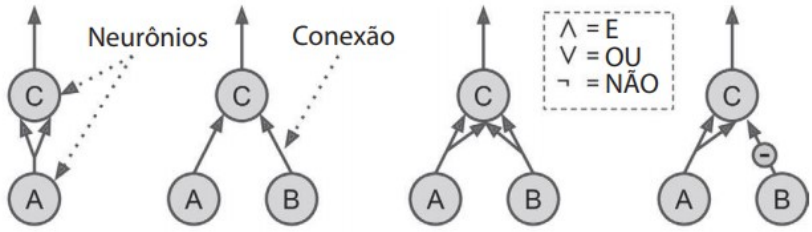


Figura 9 – RNA de McCulloch e Pitts.
Fonte: (GÉRON, 2017).

A figura mostra como os neurônios podem ser usados para calcular expressões lógicas e, se combinados, calcular expressões bem mais complexas.

Em 1957, Frank Rosenblatt propõe a arquitetura de RNA cunhada de *Perceptron*, e que foi inspirada no modelo de McCulloch e Pitts. Era uma arquitetura um pouco diferente, e que tinha um elemento central chamado de unidade linear com *threshold* (LTU, do inglês), que está ilustrada na Figura 10.

Agora, ao invés de valores binários, como no modelo de McCulloch e Pitts, as entradas e saídas são números reais, e cada conexão de entrada está associada a um peso. A LTU calcula uma soma ponderada de suas entradas ($z = w_1 x_1 + w_2 x_2 ... + w_n x_n = \mathbf{w}^{\mathrm{T}} \mathbf{x}$), depois aplica uma função degrau (*step*) a esta soma e exibe o resultado, como na figura (GÉRON, 2017). Esta função degrau, bem como todas as outras que ao longo do tempo a substituíram nas diversas pesquisas que se sucederam, são comumente chamadas de funções de ativação, e serão melhor explanadas mais à frente.

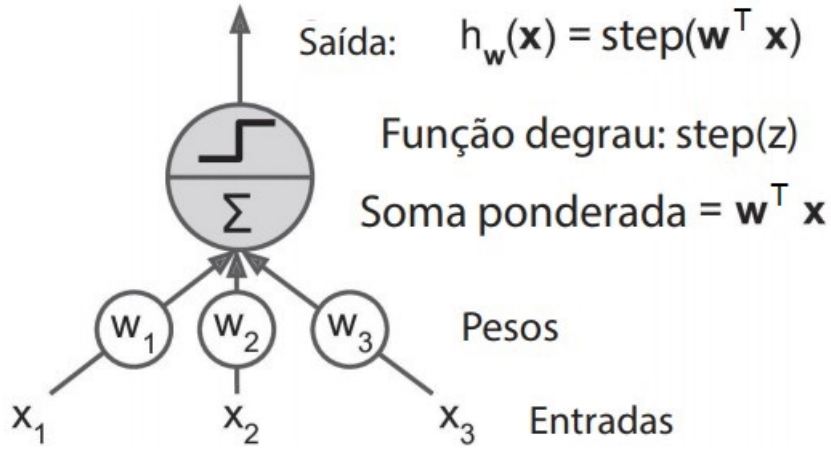


Figura 10 – Unidade linear com *threshold*.
Fonte: (GÉRON, 2017).

Um conjunto de três LTUs, como mostra a Figura 11, pode ser chamado de Perceptron (GÉRON, 2017). No entanto, adotaremos a ideia de que somente uma LTU pode também ser chamada de Perceptron, como é mais comum ser chamado.

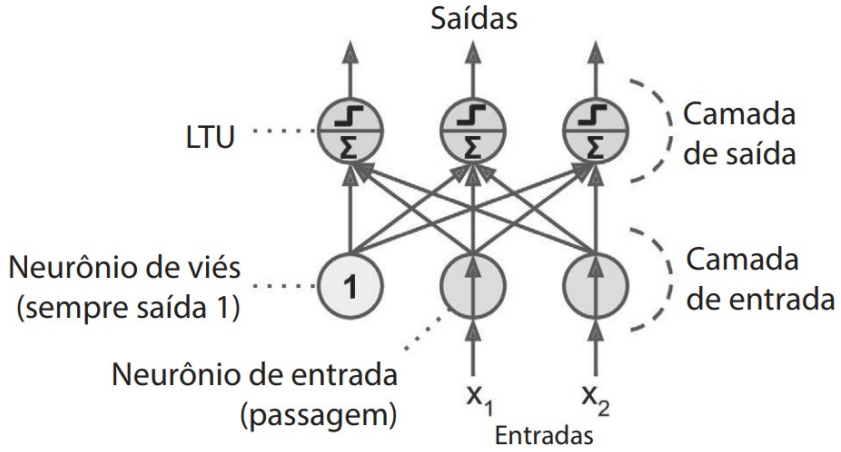


Figura 11 – Modelo multi-perceptron como um classificador de múltiplas saídas.
Fonte: (GÉRON, 2017).

Para um modelo matematicamente mais interessante, será usada a notação x_i para a i-ésima entrada do neurônio ($i \in \mathbb{N}$), w_i para o i-ésimo peso que pondera as entradas i, b para a polarização (ou *bias*, termo usado neste trabalho), $\sigma_a(.)$ para as funções de ativação [12] e y_k para a saída do k-ésimo neurônio. A Figura 12 ilustra, a grosso modo, a notação adotada.

Uma grande contribuição de Rosenblatt consistiu também no algoritmo de treinamento do Perceptron, que segue a lógica da regra *hebbiana* (ou aprendizado hebbiano). [13] Essa regra sugere que *quando um neurônio biológico desencadeia outro neurônio com frequência, a*

[12] O subscrito a em σ_a será usado para diferenciar da notação da variância σ.
[13] Refere-se ao neurofisiologista Donald Hebb.

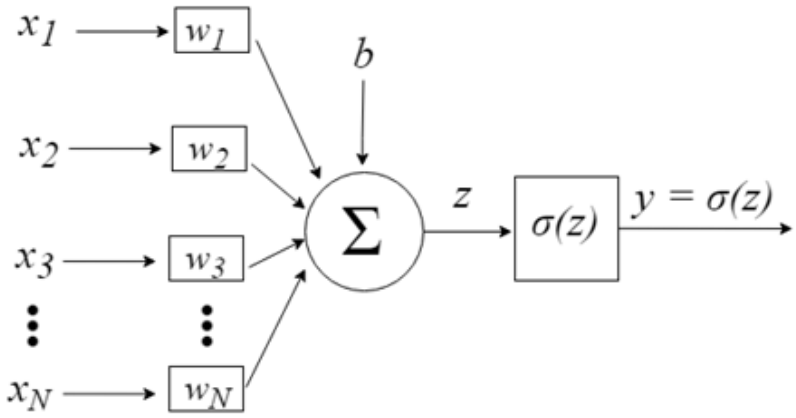


Figura 12 – Modelo matemático do Perceptron.
Fonte: https://www.mql5.com/pt/articles/8908.

conexão entre esses dois neurônios fica mais forte. Com uma variante dessa ideia, Rosenblatt propõe o treinamento do Perceptron baseado no cálculo dos erros produzidos pela rede, ao fazê-la buscar a melhor combinação dos valores dos pesos que a tornem apta a executar uma decisão. A cada previsão incorreta, os pesos que contribuíram para a saída correta é que serão reforçados. A Equação (2.15) mostra a regra matematicamente:

$$w_{i,k}^{j} = w_{i,k}^{j-1} + \eta\,(y_i - \hat{y}_k)\,x_i, \tag{2.15}$$

em que $w_{i,k}^{j}$ é o peso de conexão entre o i-ésimo neurônio de entrada e o k-ésimo neurônio de saída na j-ésima iteração, e η a taxa de aprendizado (GÉRON, 2017).

Como se vê, a decisão de cada neurônio, na busca por uma melhor combinação dos valores dos pesos, tem um caráter linear. Se as instâncias de treinamento forem linearmente separáveis, Rosenblatt demonstrou que esse algoritmo convergiria para uma solução. Contudo, por esse motivo, o Perceptron tinha uma limitação quanto a tarefas de aprendizado em contextos não-lineares. Isso gerou desapontamento nos pesquisadores, que descartaram por um tempo o estudo das redes neurais.

Foi nos anos 80 que as redes neurais atraíram um maior interesse novamente, quando Rumelhart *et al.* mostraram que redes neurais com múltiplas camadas poderiam ser treinadas efetivamente por meio da retropropagação dos erros de aprendizado (RUMELHART *et al.*, 1986).

Os perceptrons multi-camadas (MLP) começaram a ganhar muita atenção, principalmente porque naquele momento estes poderiam ser treinados de forma mais eficiente com o algoritmo de retropropagação (*backpropagation algorithm*, ou BPA) [14]. A Figura 13 retrata, a

[14] O algoritmo não será discutido em detalhes. Para o leitor interessado nos detalhes, cf. (GOODFELLOW *et al.*,

grosso modo, a arquitetura do MLP, que possui sempre uma camada chamada *camada oculta*, uma camada de entrada e uma camada de saída. Cada neurônio da camada oculta e da camada de saída o leitor pode considerar como sendo aquele apresentado na Figura 12. Quando um MLP possui duas ou mais camadas ocultas é chamado de *rede neural profunda* (DNN, do inglês *Deep Neural Network*). Daí o nome aprendizado profundo.

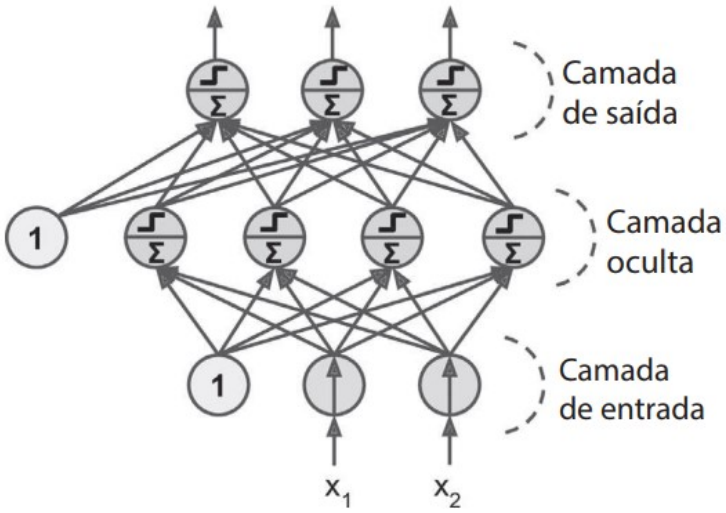


Figura 13 – Perceptron multicamada.
Fonte: (GÉRON, 2017).

A arquitetura das redes neurais profundas é usualmente diferenciável, logo, os pesos (ou parâmetros) do modelo podem ser aprendidos na medida em que se busca minimizar uma função de perda usando métodos de gradiente descendente através do BPA, seguindo a tão fundamental regra da cadeia do Cálculo (ZHANG *et al.*, 2019).

Para o caso de aprendizado supervisionado, o treinamento das DNNs se divide em dois processos, um de caminho direto (*forward pass*) e um de caminho reverso (*backward pass*). Se a DNN possuir l camadas e não for configurada com realimentações [15] o caminho direto pode ser descrito pela equação abaixo:

$$f_l(\mathbf{x}_{l-1};\theta_l) = \sigma(\mathbf{W}_l\mathbf{x}_{l-1} + \mathbf{b}_l), \tag{2.16}$$

onde $\mathbf{W}_l \in \mathbb{R}^{N_l \times N_{l-1}}$, $\mathbf{b}_l \in \mathbb{R}^{N_l}$ e $\sigma(.)$ é a função de ativação. O conjunto de parâmetros, que denotamos por $\theta_l = \mathbf{W}_l,\ \mathbf{b}_l$, é formado pela matriz de pesos e pelos *bias* de cada camada. Cada saída, de cada neurônio, é acumulada e alimenta os neurônios da camada seguinte, onde serão mais uma vez alvos de uma função de ativação num encadeamento de composição de funções. Esse caminho se chama direto em razão de não haver nenhuma retroalimentação.

2016), capítulo 6, seção 6.5.

[15] Mais conhecida como *feedforward networks* ou redes de propagação direta.

No fim do caminho direto, tem-se a saída y, que é resultado de uma combinação dos parâmetros de pesos e *bias*. Essa saída é comparada com o resultado esperado e através de uma função-custo, que é pretendida ser minimizada, os parâmetros são atualizados no caminho reverso, pelo BPA que, por sua vez, faz o trabalho de propagar os gradientes da função-custo em relação aos parâmetros para a camada de entrada da rede. Abaixo pode-se ver a perda da rede que pretende-se ser minimizada:

$$L(\theta) = \frac{1}{S} \sum_{i=1}^{S} l(\hat{y}, y). \tag{2.17}$$

A função $l(\hat{y}, y) : \mathbb{R}^{N_l} \times \mathbb{R}^{N_l} \mapsto \mathbb{R}$ é a função de perda, que pode ser alguma das que estão na Tabela 2.

Tabela 2 – Tipos de função de perda.

Função de Perda	l(**u**, **v**)
Erro quadrático médio (MSE)	$\|\mathrm{u}-\mathrm{v}\|_2^2$
Entropia cruzada categórica	$-\sum_j u_j log(v_j)$
Entropia cruzada binária	$-\frac{1}{N}\sum_j^N (u_j log(v_j) + (1-u_j log(1-v_j)))$

O processo de treinamento anteriormente descrito de forma sucinta, de modo amplo, é o que se chama na literatura de *otimização*, que é basicamente a tarefa de encontrar o melhor conjunto de parâmetros que minimizem (ou maximizem) uma função. Grande parte dos otimizadores utilizados em treinamento de DNN são baseados no cálculo dos gradientes, que são as derivadas parciais da função-custo em relação aos parâmetros da rede. Um otimizador clássico, largamente utilizado em DL, é o chamado Gradiente Descendente Estocástico (SGD), que atualiza os parâmetros da rede da seguinte forma:

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta; x^{(i)}, y^{(i)}), \tag{2.18}$$

onde ∇_θ é o operador gradiente, $x^{(i)}$ e $y^{(i)}$ são as amostras de treino, e θ é uma notação resumida para os parâmetros da rede. Note que a atualização pode ser feita para uma ou um pequeno conjunto de instâncias de treino apenas (geralmente chamado de *minibatch*). Essas instâncias são escolhidas aleatoriamente, daí o nome "estocástico" do método. Isso acelera o processo de treinamento, uma vez que o algoritmo não é exposto a todos os dados de treino. Em contrapartida, esse método acaba por não ser tão estável próximo aos mínimos locais, pois fica rebatendo a cada iteração, principalmente se a função-custo for muito irregular. Uma forma de contornar isso é adotar a prática de diminuir gradativamente a taxa de aprendizado a cada época [16] a fim de

[16] Uma passagem inteira pelos dados de treino, no modo direto e reverso.

que o algoritmo se estabeleça no mínimo global. Um outro fato a ser observado é que a taxa de aprendizado no caso clássico do SGD é fixa. Isso motiva uma taxa de aprendizado adaptativa. E é exatamente isso que faz o algoritmo Adam, usado neste trabalho.

O algoritmo Adam (Estimativa de Momento Adaptativo) atualiza os parâmetros de acordo com a seguinte equação:

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}} + \varepsilon}\hat{\mathbf{m}}, \tag{2.19}$$

onde:

$$\begin{aligned} \hat{\mathbf{m}} &= \frac{\mathbf{m}}{1-\beta_1} \\ \hat{\mathbf{v}} &= \frac{\mathbf{v}}{1-\beta_2}, \end{aligned} \tag{2.20}$$

em que:

$$\begin{aligned} \mathbf{m} &= \beta_1 \mathbf{m} + (1-\beta_1)\nabla_\theta L(\theta) \\ \mathbf{v} &= \beta_2 \mathbf{v} + (1-\beta_2)\nabla^2_\theta L(\theta). \end{aligned} \tag{2.21}$$

Neste caso, o algoritmo atualiza as médias móveis exponenciais do gradiente (**m**) e do gradiente quadrático (**v**) onde os hiperparâmetros β_1 e β_2 controlam as taxas de decaimento exponencial dessas médias móveis e são inicializados em 0,9 e 0,999, respectivamente; e o termo de suavização ε é iniciado com 10^{-8}. As próprias médias móveis são estimativas do 1º momento (a média) e do 2º momento (variância não centrada) do gradiente. O método Adam tem sido largamente utilizado por alcançar resultados muito bons de forma muito rápida, pois leva em consideração os gradientes das iterações mais recentes, em vez de todos os gradientes desde o início do treinamento. Para mais detalhes do método, o leitor pode consultar o trabalho original dos autores em (KINGMA; BA, 2015).

As funções de ativação, que também fazem parte do processo de treinamento e que assumem um papel muito importante são agora consideradas. Algumas das mais utilizadas estão nas Equações (2.22)-(2.26).

$$\text{Linear} \longrightarrow u_i \tag{2.22}$$

$$\text{Tangente Hiperbólica} \longrightarrow tanh(u_i) \tag{2.23}$$

$$\text{Sigmoid} \longrightarrow \frac{1}{1+e^{-u_i}} \tag{2.24}$$

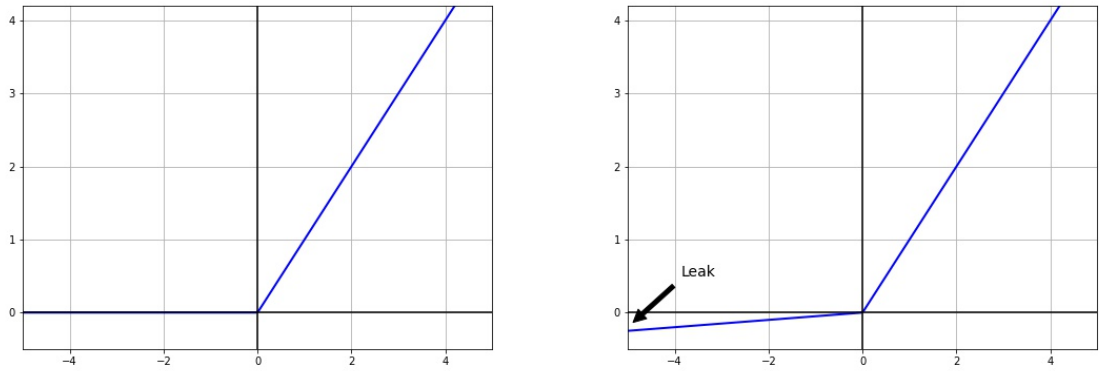
$$\text{Softmax} \longrightarrow \frac{e^{u_i}}{\sum_i e^{u_i}} \tag{2.25}$$

$$\text{ReLU} \longrightarrow max(0,\, u_i) \tag{2.26}$$

Essas funções, com exceção da função linear, adicionam não-linearidade à rede e fazem com que a rede tenha a habilidade de representar funções mais complexas dos dados e parâmetros. Assim, usando uma ativação não linear é possível gerar funções não lineares de entradas para saídas. Em especial, a função *softmax* é geralmente usada na saída das redes neurais, para problemas de classificação, onde a saída de cada neurônio equivale à probabilidade estimada da classe correspondente.

Um problema encontrado pelos pesquisadores com a função de ativação ReLU (Unidade Linear Retificada) consiste no chamado problema de *morte de neurônio*. Esse problema ocorre quando os pesos dos neurônios começam a ter valor zero, à medida em que o BPA ocorre. Quando a soma ponderada das entradas é negativa, o resultado da aplicação da função ReLU é igual a zero, como se vê na Figura 14a. Quando isso ocorre, é improvável que o neurônio volte à vida útil, uma vez que o BPA irá propagar sempre um gradiente igual a zero (GÉRON, 2017).

Como uma solução para esse problema, usa-se geralmente uma variante da função ReLU chamada *leaky*-ReLU = $max(\alpha z, z)$. O hiperparâmetro [17] α define o "vazamento" (daí o nome *leaky*, do inglês) da função. Quando $z < 0$, o resultado da aplicação da função assume um valor irrisório, dependendo da magnitude de α, e assim garante que neurônios ainda acordem caso cheguem a ficar perto de morrer. Na Figura 14b é apresentado o gráfico da função *leaky*-ReLU, onde o *leak* indica o vazamento mencionado da função.



(a) (b)

Figura 14 – Funções de ativação (a) ReLU e (b) *leaky*-ReLU.

Além disso, a função ReLU não é diferenciável em $z = 0$. Contudo, ela funciona muito bem na prática, e tem a vatagem de o cálculo ser rápido. Sobretudo, outro aspecto

[17] Hiperparâmetro é todo parâmetro manualmente ajustável em um modelo de ML ou DL.

importante dessa função e sua variante *leaky*-ReLU é que não possuem um valor máximo de saída (não saturam para grandes valores de entrada), o que ajuda na redução do problema dos platôs durante o SGD.

Vantagens e Desvantagens de DL para comunicação ponto-a-ponto (ZHANG et al., 2019)

Algumas boas razões nos fazem acreditar que as técnicas de DL podem ter um impacto cada vez maior na sociedade moderna, e principalmente nas comunicações digitais ponta-a-ponta. Entre elas estão:

1. Algoritmos de DL podem extrair diversas características de dados que possuem uma estrutura complexa e correlações internas. O processo de aprendizagem não precisa ser planejado por um humano, o que simplifica tremendamente o trabalho de pré-processamento dos dados. No contexto de comunicações móveis sabe-se que os dados são geralmente ruidosos, vindos de fontes bem heterogêneas, por isso não apresentam padrões espaço-temporais triviais.
2. DL é capaz de lidar com grandes quantidades de dados, sem sofrer substancialmente de sobre-ajuste aos dados (quando os modelos "decoram"os dados e perdem a capacidade de generali zação). As arquiteturas de implementação dos algoritmos de DL, geralmente construídas de forma a possibilitar a distribuição e paralelismo dos cálculos, podem garantir velocidade e capacidade de processamento frente à massiva quantidade de dados gerada pelos sistemas de comunicação.
3. As representações aprendidas pelas DNN podem ser compartilhadas para várias tarefas diferentes, enquanto em ML isso é limitado e um pouco mais difícil. As técnicas de Transfer Learning (Aprendizado Transferido) [18] podem ser utilizadas para otimizar o tempo de resolução dos problemas em comunicação móvel digital.
4. DL é uma grande e potencial solução para o problema de dados não rotulados, que é bastante comum em sistemas de comunicação móvel. Algumas arquiteturas como as Redes Generativas Adversárias (GANs) [19] são muito úteis para explorar dados não rotulados de maneira não-supervisionada.

Embora existam muitas vantagens significativas na aplicação de DL em comunicação digital ponta-a-ponta, é necessário mencionar algumas desvantagens também, a fim de esclarecer

[18] Cf. (ZHUANG *et al.*, 2019) para mais detalhes sobre essas técnicas.
[19] Cf. (GOODFELLOW *et al.*, 2014).

as limitações.

1. Em geral, DL é vulnerável a dados adversários (usados para "enganar"os algoritmos de DL). Isso afeta a aplicação de DL nas comunicações digitais, pois acaba possibilitando ataques *hackers*, explorando as vulnerabilidades.
2. DL é comumente chamada de abordagem "caixa preta", o que significa que são técnicas que possuem baixa interpretabilidade. Isso se torna um problema mais crítico pricipalmente quando se lida com aplicações que envolvam recursos econômicos.
3. Algoritmos de DL podem demandar bastante poder computacional (energia), o que não é tão fácil de contornar assim quando se fala em dispositivos móveis e embarcados, que possuem restrições de energia e capacidade.
4. DNNs e outras arquiteturas de DL geralmente possuem muitos hiperparâmetros e encontrar uma configuração ótima pode ser difícil.

2.3.2 Autoencoders

Os Autoencoders são uma técnica de aprendizado não supervisionado, na qual se usa as redes neurais para a tarefa de aprendizado de representação. Nesse tipo de arquitetura, impõe-se um gargalo na rede que força uma representação de conhecimento compactada da entrada original. Se os recursos de entrada fossem independentes um do outro, essa compressão e reconstrução subsequente seria uma tarefa muito difícil. No entanto, se houver algum tipo de estrutura nos dados (ou seja, correlações entre os recursos de entrada), essa estrutura poderá ser aprendida e consequentemente aproveitada ao forçar a entrada através do gargalo da rede.

As primeiras referências aos Autoencoders já datam de 1993, quando em (HINTON; ZEMEL, 1993) os autores mencionam a ideia de que os Autoencoders são uma variação de duas técnicas bem conhecidas: a Análise de Componentes Principais (PCA, do inglês *Principal Component Analysis*) e a Quantização Vetorial (VQ, do inglês *Vector Quantization*). De fato, essas duas técnicas podem ser implementadas a partir de uma arquitetura Autoencoder, como eles mencionam. A primeira é basicamente um Autoencoder com ativações lineares e o objetivo sendo a minimização do erro quadrático médio (MSE), para citar; é possível mostrar que o resultado é o mesmo (GOODFELLOW *et al.*, 2016).

A junção da característica não-linear da VQ com a representação distribuída e fatorial da PCA leva à técnica de Autoencoder conhecida hoje. O Autoencoder sempre é composto de duas partes: um codificador (ou rede de conhecimento), que converte a entrada

para uma representação interna, seguido de um decodificador (ou rede geradora) que converte a representação interna para as saídas. As saídas, portanto, são versões reconstruídas da entrada, a partir da representação interna.

Matematicamente, as duas partes de um Autoencoder podem ser vistas como duas transições f e g da seguinte forma, considerando $\mathbf{x} \in \mathbb{R}^N$:

$$\begin{aligned} &f : \mathbb{R}^N \to \mathbb{R}^n,\ \mathbf{h} = f(\mathbf{x}) \\ &g : \mathbb{R}^n \to \mathbb{R}^N,\ \mathbf{x}' = g(h). \end{aligned} \tag{2.27}$$

A variável **h** é geralmente chamada de código, representação latente ou variável latente. A dimensão dessa variável classifica os Autoencoders em incompletos ou supercompletos. Quando $n < N$ tem-se um Autoencoder incompleto. Caso contrário, tem-se um supercompleto. As saídas x' são chamadas de reconstruções, e a função de custo contém uma perda de reconstrução que penaliza o modelo quando as reconstruções são diferentes das entradas, já que o Autoencoder tenta reconstruí-las.

A Figura 15 mostra a arquitetura genérica de um Autoencoder incompleto. Note a dimensão do código como sendo menor que a dimensão da entrada e da saída, representado pelo retângulo de menor tamanho. Outra observação é que, como já mencionado, pode-se implementar o codificador e o decodificador como sendo cada um uma rede neural (um MLP, por exemplo). Isso é bem comum na literatura, e é essa possibilidade que é explorada neste trabalho.

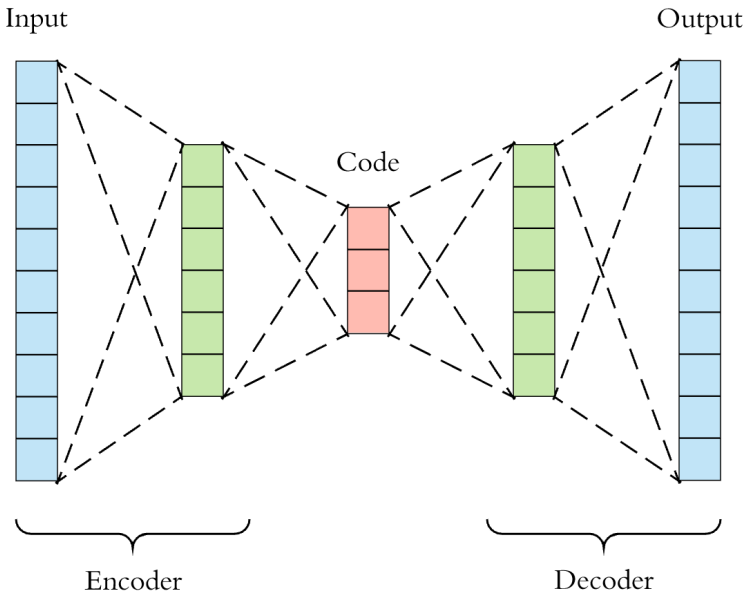


Figura 15 – Arquitetura de um Autoencoder incompleto.

Para uma rede de apenas 1 camada, por exemplo, um autoencoder pode ser treinado

para a minimização do MSE. A perda de reconstrução é expressa a seguir:

$$L(x,x') = \|x - x'\|_2^2 = \|x - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x}+b))+b')\|_2^2, \tag{2.28}$$

onde σ', $\mathbf{W}'$ e b' são a função de ativação, a matriz de pesos e o *bias* da rede decodificadora, respectivamente. O treinamento ocorre da mesma maneira como no MLP explicado na Seção 2.3.1, através do BPA. A diferença está no objetivo, onde deseja-se obter uma réplica muito verossímil das entradas da rede. Um questionamento natural que se faz é sobre onde está a utilidade de reproduzir réplicas de um conjunto de dados trivialmente. A resposta, na verdade, está no fato de que essa tarefa não é feita sem uma dificuldade imposta ao Autoencoder. O que se faz geralmente é impor algumas restrições que dificultem essa tarefa. Por exemplo, pode-se limitar o tamanho da representação interna ou adicionar ruído às entradas originais. Essas restrições acabam impedindo o Autoencoder de apenas copiar os dados de entrada, obrigando-o a aprender maneiras eficientes de representar os dados (GÉRON, 2017).

2.3.3 Potencial dos Autoencoders para comunicação ponta-a-ponta

Considerando o que foi apresentado na seção anterior sobre os Autoencoders, pode-se verificar que essa técnica tem um potencial de eventualmente substituir a cadeia clássica de um sistema de comunicação digital.

Como visto na Seção 2.1, todo o processo clássico é feito com base na separação em blocos, onde cada um deles individualmente prepara a informação para algum tipo de dano, principalmente os danos causados pelo canal de comunicação, e para um melhor uso dos recursos de energia e frequência. No entanto, é sabido que, por exemplo, a separação em codificação de fonte e canal não é ótima (GOLDSMITH, 1995). Isso motiva a utilização dos Autoencoders para uma comunicação ponta-a-ponta.

Como foi mencionado na Seção 2.3.2, os Autoencoders possuem a característica de buscarem a representação mais eficiente dos dados de entrada, e assim o fazem de uma forma mais global e distribuída. Isso pode ser explorado para flexibilizar e acelerar o processo de comunicação. Os autores em (O'SHEA *et al.*, 2016), (O'SHEA; HOYDIS, 2017) e mais recentemente em (ERPEK *et al.*, 2020) fazem essa análise mostrando que é possível utilizar os Autoencoders para uma comunicação ponta-a-ponta.

Neste trabalho, pretendeu-se estender o trabalho realizado por esses autores, adotando outras configurações e comparando com outros sistemas clássicos, a fim de construir uma base de

conhecimento dos aspectos práticos que tangem esse desafio, que possibilite futuras discussões do tópico.

Vale ressaltar que o modelo de Autoencoder, como uma técnica de DL, já de antemão tem suas desvantagens conhecidas; as mesmas que foram citadas na Seção 2.3.1. Esses pontos dificultam o uso dos Autoencoders na prática. Para mais detalhes desse ponto, o leitor pode consultar a referência (DÖRNER *et al.*, 2018), melhor descrita na Seção 1.2 deste trabalho.

2.3.4 Bibliotecas Tensorflow e Keras

Com a grande popularização das técnicas de ML e DL nos últimos anos, devido ao grande avanço na capacidade de processamento dos computadores, foi surgindo a necessidade de ter à disposição bibliotecas mais robustas, flexíveis e otimizadas para construir e treinar colossais redes neurais, que são complexas e exigem muito esforço na tarefa de escolher os melhores parâmetros.

É com essa motivação que aqui apresenta-se uma breve descrição da biblioteca Tensorflow (ABADI *et al.*, 2016) e a sua implementação em mais alto nível, Keras (CHOLLET *et al.*, 2015) [20].

A biblioteca Tensorflow é uma poderosa biblioteca de software para cálculo numérico de código aberto ajustada especialmente para ML e DL em larga escala. O princípio da biblioteca é básico: define-se um grafo de cálculos para executar em Python e, em seguida, o Tensorflow pega esse grafo e o executa de forma eficiente utilizando um código C++ otimizado. Um exemplo de grafo para cálculo de uma função pode ser visto na Figura 16:

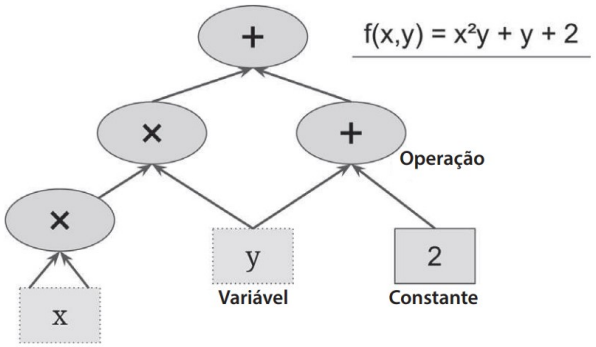


Figura 16 – Exemplo de grafo para cálculo de uma função.
Fonte: (GÉRON, 2017)

Um aspecto que a torna bastante útil é a possibilidade de, com ela, poder dividir

[20] Grande parte do que é apresentado nesta subseção baseia-se no Capítulo 9 de (GÉRON, 2017).

o grafo em pedaços e executá-los em várias CPUs (Unidades Centrais de Processamento) e GPUs (Unidades de Processamento Gráfico). Isso traz um ganho de velocidade considerável no treinamento de grandes redes neurais.

Alguns pontos de destaque da biblioteca incluem:

1. Pode ser executada nos principais sistemas operacionais (Windows, Linux e MacOS) bem como em dispositivos móveis com iOS e Android.
2. Inclui implementações C++ altamente eficientes, além de possuir uma API (Interface de Programação de Aplicações) C++ para definir suas próprias operações de alto desempenho.
3. Fornece vários nós de otimização, que implementam algoritmos de diferenciação automática, e que facilitam bastante a implementação do BPA, por exemplo. A biblioteca calcula os gradientes das funções definidas pelo usuário.
4. Várias outras APIs de alto nível foram construídas independentemente com base no Tensorflow, otimizando ainda mais o processo de construção e treinamento de redes neurais. Entre elas, inclui-se a Keras, uma das mais usadas atualmente e que foi utilizada neste trabalho.

3 RESULTADOS DE SIMULAÇÃO

Neste capítulo, apresentamos os resultados obtidos por simulação analisando a performance dos Autoencoders frente ao modelo clássico prévia e teoricamente discutido na Seção 2.1 e na Seção 2.2.

Como se segue, na Seção 3.1 discutimos o modelo utilizado nas simulações, suas particularidades, parâmetros e alguns problemas de cunho prático que surgiram nas consideráveis tentativas de replicar alguns resultados dos principais artigos utilizados como ponto de partida deste trabalho. A Seção 3.2 trata da análise dos Autoencoders em comparação com sistemas de comunicação sem a presença de codificação (seja de fonte ou canal), com os convencionais esquemas de modulação BPSK e QPSK, em um canal AWGN, bem como discutimos o impacto da mudança de alguns hiperparâmetros nos resultados durante a fase de treinamento do modelo. Já na Seção 3.3 apresentamos comparações do método de Autoencoders com sistemas convencionais codificados. Nesta seção, adotamos os códigos Convolucionais (doravante, o nome completo ou CC) e o código Hamming (7,4) como *baselines*. Adotamos as métricas de BER, BLER e tempo de execução ao longo das simulações que se seguem, sempre optando por aquela que melhor garante uma base justa de comparação no momento.

3.1 Modelo do sistema

Um sistema de comunicação baseado na arquitetura dos Autoencoders e que se propõe a substituir as "caixas"convencionais do modelo clássico de comunicação foi proposto em (O'SHEA; HOYDIS, 2017) com a configuração que pode ser vista na Figura 17.

No presente trabalho, adotamos o mesmo modelo proposto pelos autores, porém com algumas diferenças que foram sendo percebidas resultarem em uma semelhante ou melhor performance dentro do ambiente de simulação utilizado que, obviamente, é diferente daquele que foi usado pelos referidos autores.

Optou-se por utilizar um ambiente de simulação baseado na liguagem de programação Python (versão 3.7.12 no *Google Colab*) com as bibliotecas *Tensorflow* e *Keras*, brevemente apresentadas na Seção 2.3.4 deste trabalho. A versão utilizada de ambas as bibliotecas foi a 2.7.0.

O modelo do sistema consiste em que o transmissor pretende enviar uma mensagem $s \in \mathbb{M} = \{1, 2, ..., M\}$, fazendo n usos do canal. Ele transmite um vetor $\mathbf{x}$ de n símbolos complexos

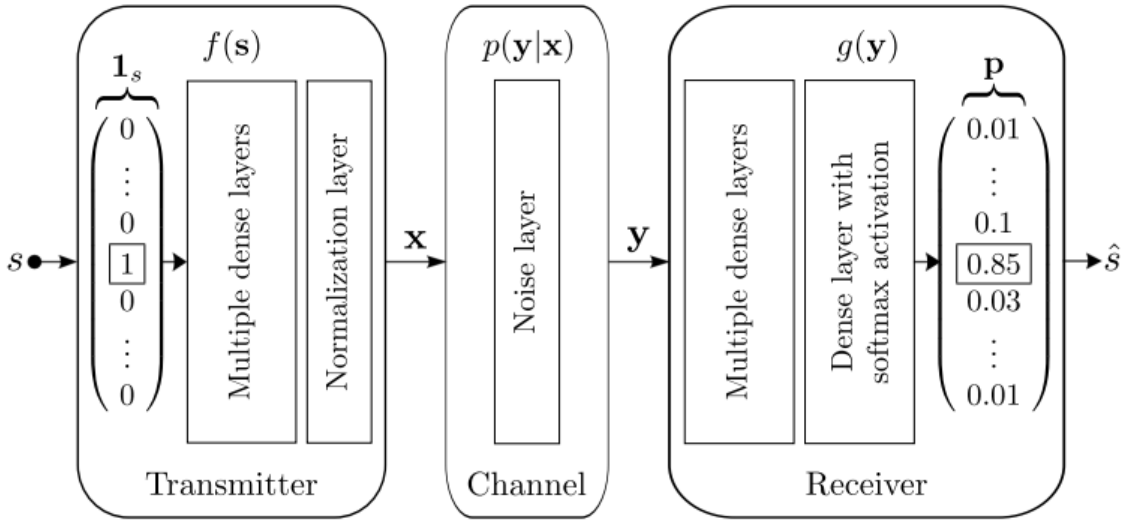


Figura 17 – Sistema de comunicação em canal AWGN como um Autoencoder
Fonte: (O'SHEA; HOYDIS, 2017).

pelo canal para enviar a mensagem s. Além disso, dentro do modelo considera-se algumas restrições impostas pela implementação física do transmissor como a restrição de energia $\|\mathbf{x}\|_2^2 \leq n$ ou a restrição de potência média $\mathbb{E}[x_i^2] \leq 1 \forall i$.

Uma mensagem s pode ser representada em $k = log_2(M)$ bits de informação, fazendo com que o sistema opere, portanto, com uma taxa de comunicação $R = k/n$ [bits/uso do canal]. O canal, modelado como AWGN, distorce a informação antes desta chegar ao receptor que, por sua vez, ao receber o sinal $\mathbf{y} \in \mathbb{C}^n$ do canal, estima a mensagem mais provável $\hat{s}$.

Neste modelo, o Autoencoder implementado é composto por um transmissor que é dado por uma rede neural de propagação direta com múltiplas camadas densas seguidas por uma camada de normalização que garante que as restrições físicas serão atendidas. A entrada da rede é um vetor M-dimensional que tem em sua composição somente um bit "1"e todos os outros elementos iguais a zero (comumente chamado na literatura de *one-hot encoded vector*). A saída do transmissor é um vetor $2n$-dimensional que corresponde aos n símbolos complexos transmitidos em n usos do canal, onde consideramos os elementos de índice par do vetor como a parte real do símbolo, e os elementos de índice ímpar a parte complexa do símbolo [1]. O canal é representado por uma camada *não-treinável* de ruído gaussiano (camada *Gaussian Noise* da *Keras*) logo antes da camada latente de código do Autoencoder, que só adiciona o ruído durante a fase de treinamento. A variância do ruído é dada por $\beta = (2RE_b/N_0)^{-1}$, onde E_b/N_0 denota a

[1] Aqui, entenda "símbolos complexos"como uma adaptação meramente numérica a vetores com dimensão duplicada, dada a forma como as bibliotecas atuais de DL funcionam.

relação sinal-ruído em termos da energia por bit (E_b) e da densidade espectral de potência do ruído (N_0). O receptor é também implementado como uma rede neural de propagação direta, formada por uma camada densa seguida de uma camada com ativação *softmax* cujo vetor de saída $\mathbf{p} \in (0,1)^M$ é composto pelas probabilidades da mensagem mais provável. A mensagem decodificada é aquela que corresponde ao índice de $\mathbf{p}$ com maior valor de probabilidade. O Autoencoder foi treinado utilizando o método de otimização *Adam* com uma taxa de aprendizado $\eta = 0.001$ e com a função de custo de entropia cruzada categórica, explicitados na Seção 2.3.1.

O esboço das dimensões das camadas que compõem o transmissor e o receptor, os respectivos nomes, bem como as funções de ativação estão listadas na Tabela 3.

Tabela 3 – Esboço do modelo do Autoencoder.

Camadas	Dimensões
Entrada	M
Densa-LeakyReLU	M
Densa-Linear	2n
Lambda	2n
Ruído Gaussiano	2n
Densa-LeakyReLU	M
Densa-Softmax	M

As camadas de 1-4 (Entrada-Lambda) fazem parte do transmissor, onde a camada Lambda (camada customizável do *Keras*) garante que a restrição de potência seja atendida. Logo após a camada Lambda tem-se a camada que age como ruído. Como já foi citado, utilizamos a camada *Gaussian Noise* da biblioteca *Keras* para agir como o canal ruidoso. O detalhe que notamos é que esta camada é uma camada de regularização no *Keras* e, portanto, não é ativada na fase de teste do Autoencoder. A camada só adiciona ruído durante o período de treinamento. Para resolver isso, duas saídas são possíveis: (1) criar uma classe customizada do *Keras*, idêntica à camada padrão da biblioteca, e que herda um objeto da classe *Layer*, sendo utilizada normalmente no modelo; ou (2) criar os modelos de transmissor e receptor logo após o treinamento das redes e usá-los na fase de teste, acrescentando um ruído AWGN na saída do modelo do transmissor. Adotamos o método (2), pois o método (1) gerou alguns problemas de integração entre as versões das bibliotecas, uma vez que o sistema utilizado na simulação foi o *Google Colab*, um ambiente que tem alguns limites de customização.

O Autoencoder implementado foi treinado para diferentes tamanhos de alfabeto (M) e para diferentes taxas de codificação (R), visando investigar o impacto desses parâmetros na performance do modelo, em comparação com: (1) sistemas convencionais não codificados

como BPSK e QPSK, buscando ter uma base justa de comparação a cada taxa de comunicação utilizada; e (2) com sistemas convencionais codificados, buscando também uma base justa de comparação.

Na fase de treinamento adotamos pelo menos três abordagens diferentes na escolha da SNR de treino (SNR_{treino}), tendo como objetivo simular, ainda que de forma sintética, a exposição do modelo às alterações de qualidade do canal, comuns no mundo real. As abordagens são as seguintes:

- Treinamento com um valor fixo de SNR (5dB ou 7dB).
- Treinamento com um valor escolhido aleatoriamente dentro de um intervalo de valores (1dB a 8dB).
- Iniciando com um valor de SNR (8dB) e decrescendo o valor de 1dB a cada 10 épocas, até o valor final de 1dB.

O dataset utilizado para treinamento, validação e teste do modelo foi gerado de forma simulada, com um total de 1.000.000 de bits, para ambos os estágios de comparação: sistemas codificados e não codificados.

3.2 Sistemas sem codificação

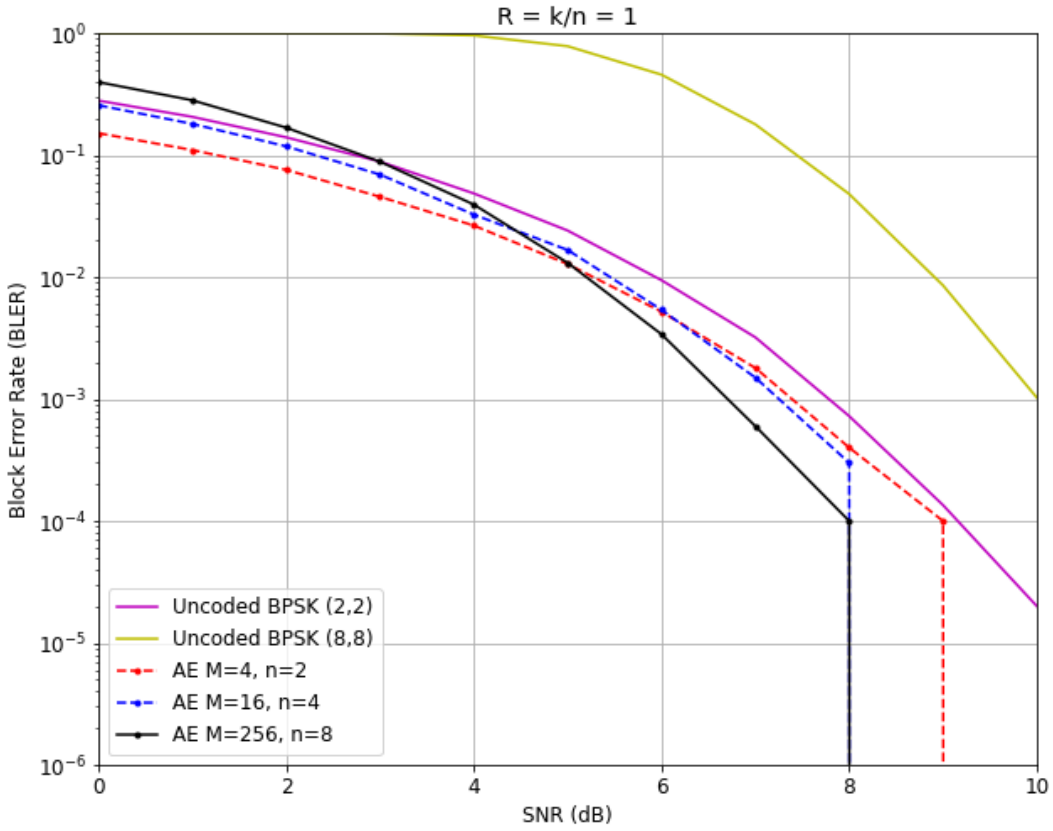
Para uma investigação da performance do modelo de Autoencoder frente aos modelos clássicos BPSK e QPSK, adotamos as configurações de tamanho da mensagem e taxa de comunicação sumarizadas na Tabela 4.

Tabela 4 – Diferentes parâmetros do Autoencoder e os sistemas convencionais respectivos.

Número de mensagens (M)	Usos do canal (n)	Taxa de Comunicação (R)	*Baseline*
4	2	1 [bit/uso do canal]	BPSK
16	4		
256	8		
4	1	2 [bits/uso do canal]	QPSK
16	2		
256	4		

A Figura 18 mostra a performance do Autoencoder para uma taxa $R = 1$ [bit/uso do canal] comparado com esquemas de modulação BPSK não codificados e simulados com duas configurações diferentes de bits por bloco de informação. Utilizamos a métrica BLER para avaliar.

Figura 18 – BLER para taxa $R = 1$ [bit/uso do canal] comparado com BPSK em canal AWGN. $SNR_{treino} = 7$dB.

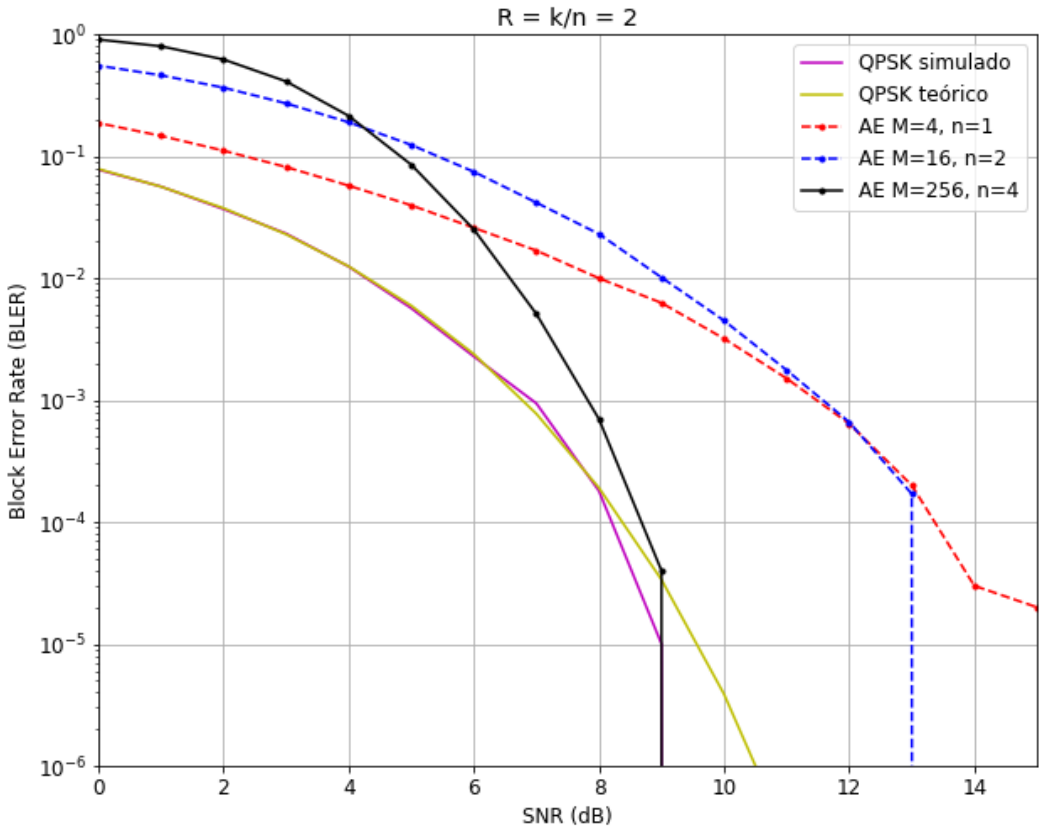


Pode-se notar pelo gráfico que o Autoencoder supera a performance dos dois esquemas BPSK utilizados, com exceção da configuração para $M = 256$ no intervalo de 0dB a 3dB, quando a BLER fica um pouco mais alta que o BPSK (2,2). No entanto, para valores ainda baixos, onde as condições do canal ainda não são satisfatórias, o Autoencoder se portou bem, sobretudo as configurações para $M = 4$ e $M = 16$.

Nota-se também o fato de que, para uma mesma taxa de comunicação, com o aumento do número de mensagens, bem como do número de usos do canal, a BLER melhora gradativamente, alcançando melhores resultados em níveis um pouco mais altos de SNR (quando o canal está com melhor qualidade). Ao que parece, o modelo aprende uma codificação temporal, à medida que vai ocorrendo os múltiplos usos do canal. Com uma dimensão maior na camada latente do modelo (canal), há mais graus de liberdade para o modelo se ajustar às distorções impostas pelo canal. No entanto, um cuidado que deve ser tomado em relação à escolha dos parâmetros de tamanho de mensagem e taxa de comunicação é que valores mais elevados podem

acarretar um maior atraso de decodificação no receptor, logo, para uma implementação em *hardware* essas questões devem ser levadas em consideração. Em (DÖRNER *et al.*, 2018), sobre o qual é comentado no Capítulo 1, os autores fazem uma excelente discussão sobre isso.

Figura 19 – BLER para taxa $R = 2$ [bits/uso do canal] comparado com QPSK em canal AWGN. $SNR_{treino} = 7$dB.



Na Figura 19 pode-se ver o resultado para $R = 2$ [bits/uso do canal] e tendo o esquema QPSK como *baseline*. Para melhores condições do canal (alta SNR), o modelo de Autoencoder para $M = 256$ tem uma performance comparável ao QPSK, muito embora para valores de SNR no intervalo de 0dB a 9dB não apresenta uma boa performance. É curioso ver também que as configurações do Autoencoder, neste caso, diferem mais entre si do que quando se usa $R = 1$.

Logo abaixo, nas Figuras 20 e 21, vemos as constelações aprendidas para alguns sistemas com taxas de comunicação e tamanho de mensagem propositadamente escolhidas. As figuras foram geradas após a fase de treinamento, usando o modelo do transmissor.

O procedimento consiste em salvar os pesos do modelo integralmente, e ao carregá-lo para realizar os testes, faz-se o carregamento somente das camadas que se tem interesse em ver a

Figura 20 – Constelação aprendida de (M=4, n=1) e (M=16, n=1), respectivamente.

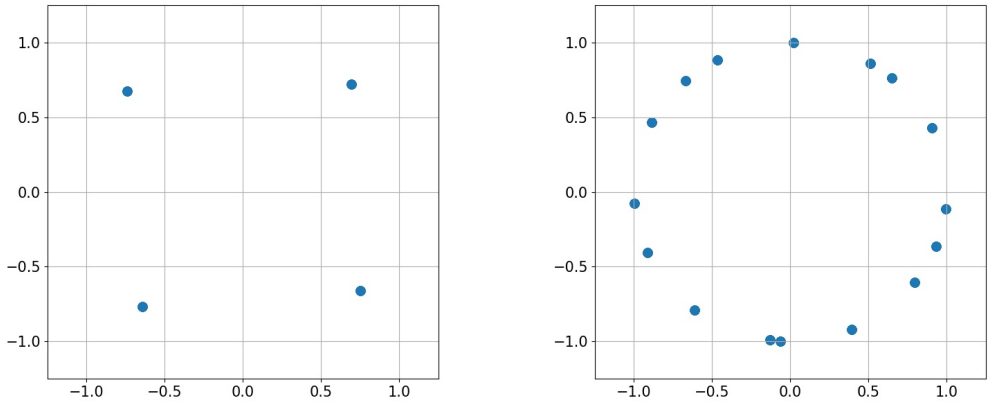
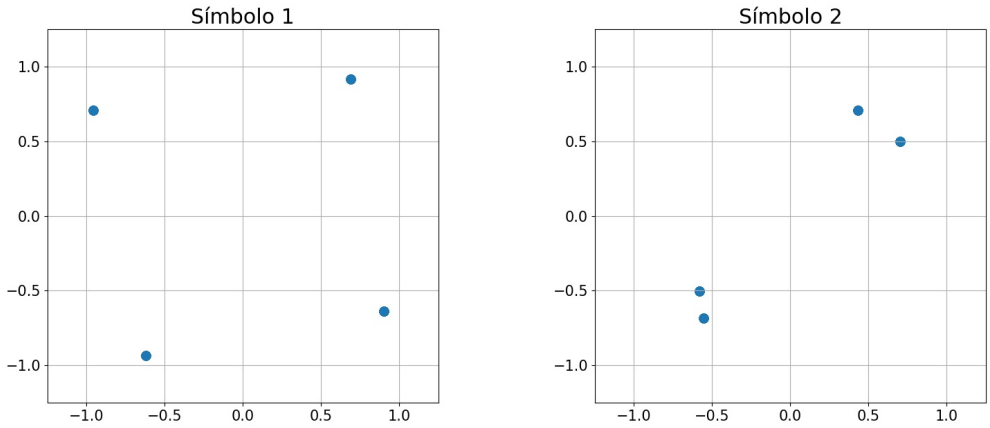


Figura 21 – Constelação aprendida de M=4 e n=2.



saída. Neste caso, carrega-se aquelas camadas que pertencem somente ao modelo do transmissor, submete-se essas camadas ao conjunto apropriado de bits de informação e recupera-se na saída o que foi aprendido pelo modelo, antes de passar pelo canal ruidoso.

Como anteriormente citado, o modelo do transmissor tem em sua saída um vetor de dimensão $2n$ que é mapeado para a dimensão complexa n. Para gerar o gráfico, adotou-se os índices pares como sendo as componentes em quadratura (Q) dos símbolos, e os índices ímpares como sendo as componentes em fase dos símbolos (I).

Pode ser visto pelas figuras das constelações, especialmente a Figura 20, que para M=4 e M=16 a disposição dos símbolos em muito se assemelha às constelações dos esquemas de modulação clássicos QPSK e 16-PSK, a menos de alguma rotação no primeiro caso e uma distância mais reduzida entre dois símbolos no segundo caso.

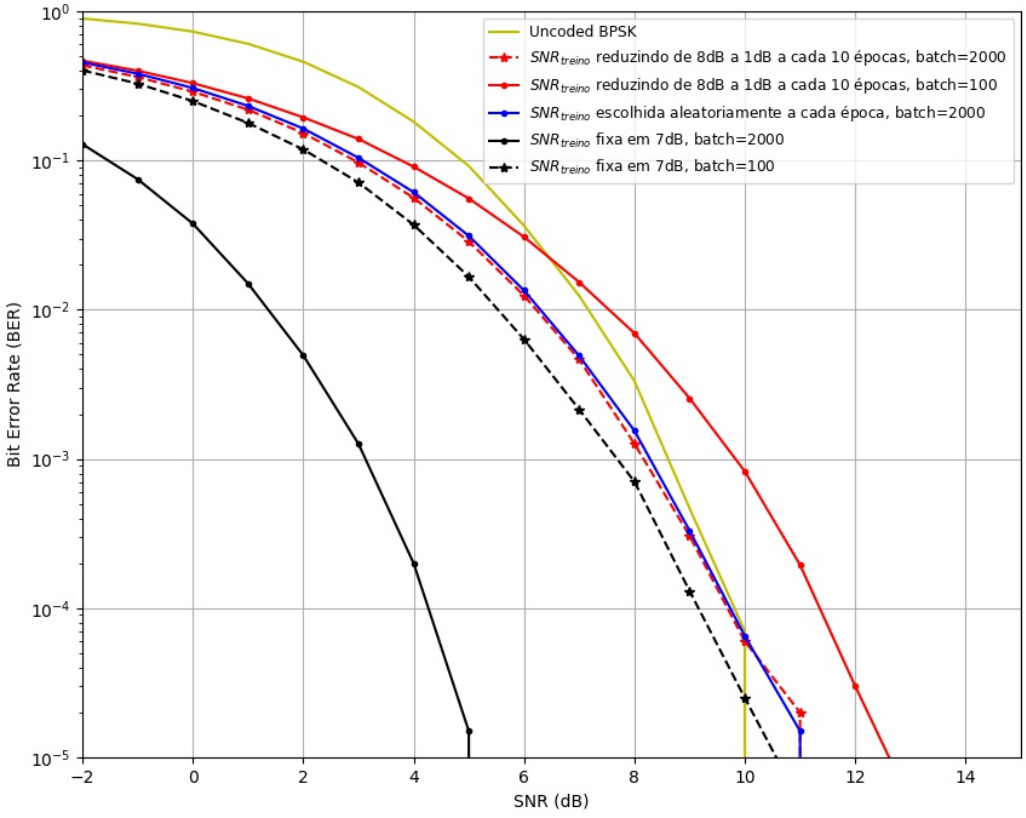
Nos símbolos para M=4 e n=2 o modelo aprendeu duas configurações distintas, mas ainda com algum padrão observável, o que é bastante interessante. Isso indica que o Autoencoder

consegue aprender alguma codificação que o torna tolerante às distorções do canal.

Isso é diferente de manter as constelações fixas para uma transmissão, como ocorre em sistemas convencionais. Os Autoencoders, neste caso, aprendem M conjuntos únicos de mensagem para qualquer configuração de n. Se e.g. n = 1, a dimensão do símbolo complexo é 2, mas independente disso, para cada vetor $s \in \mathbb{R}^{t \times 2n}$ recuperado após a operação do transmissor, onde t é o numero de amostras de treino/teste, são sempre obtidos M conjuntos de valores únicos, que representam cada mensagem/símbolo.

Agora, nas Figuras 22-24 vemos uma avaliação do impacto da mudança de alguns hiperparâmetros na BER. Para essas avaliações, usamos a configuração de sistema (M=16, n=1), com exceção da Figura 23, onde usamos (M=256, n=8).

Figura 22 – BER para diferentes abordagens na escolha da SNR_{treino}.



A motivação da Figura 22 foi observar como o modelo de Autoencoder se comporta em face de algumas alterações nas condições do canal, na fase em que está sendo treinado.

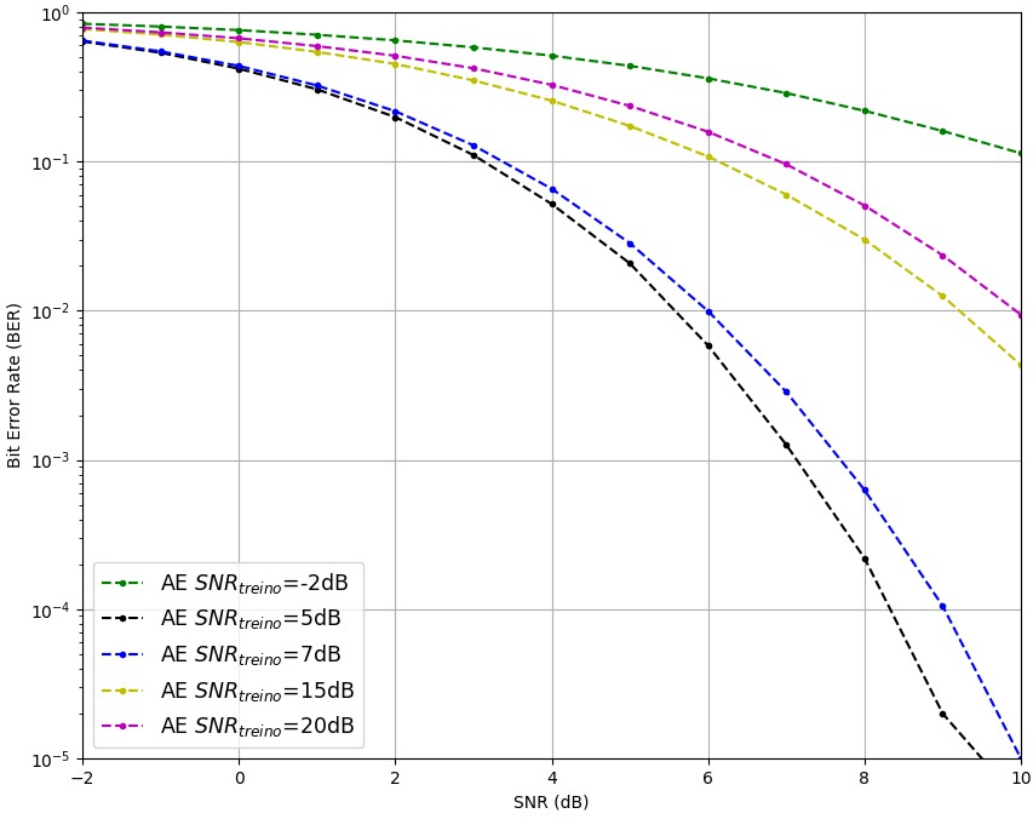
Algumas coisas são notáveis nesse gráfico. A primeira mais patente é a melhor

performance do modelo treinado com uma SNR fixa e um tamanho de *batch* igual a 2000, em detrimento daquele treinado com a mesma SNR, porém com um *batch* menor.

É interessante notar que com SNR fixa o modelo tem a melhor performance em comparação com as outras duas abordagens utilizadas. Isso provavelmente pode trazer alguma vantagem para a implementação em *hardware*, uma vez que não traz a necessidade de acrescentar alguma heurística que cuide da variação da SNR durante a fase de treino, poupando um pouco de memória.

Na Figura 23 vemos um outro resultado interessante, que diz respeito ao treinamento do modelo em valores fixos de SNR previamente definidos.

Figura 23 – BER para diferentes valores de SNR_{treino}. Sistema com M=256 e n=8.

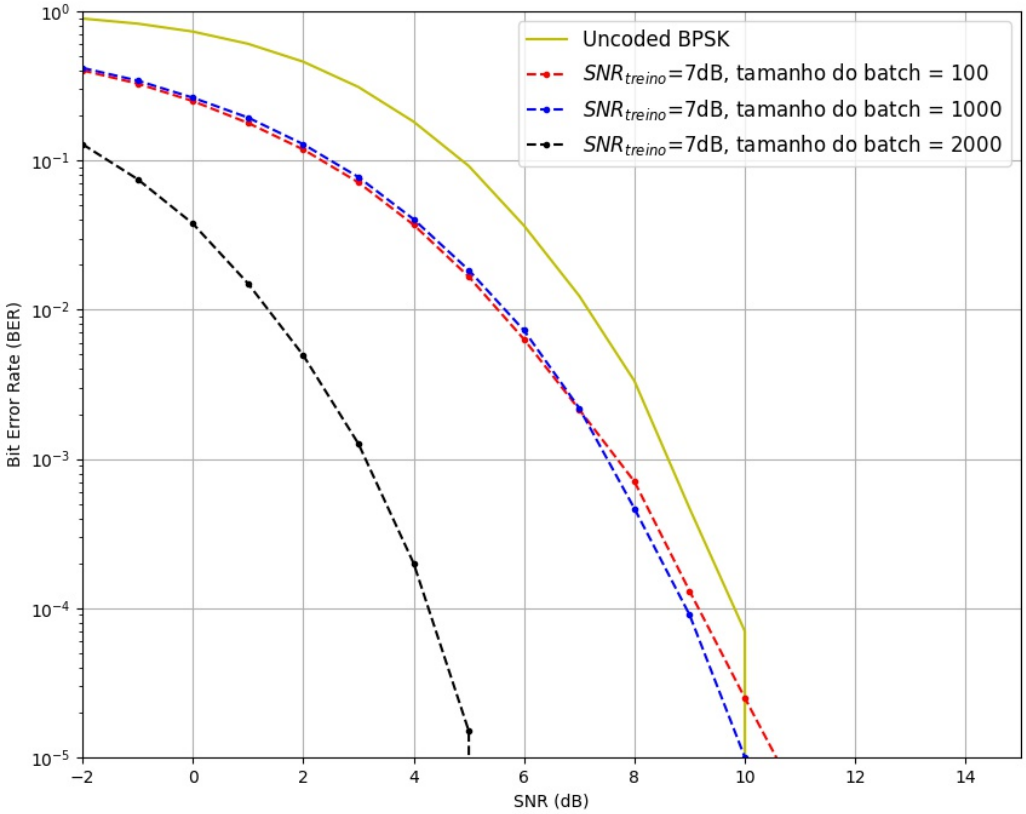


Pode-se observar que para baixos valores de SNR o modelo tem uma melhor performance. Isso se deve, provavelmente, ao fato de que valores baixos de SNR submetem as camadas do modelo a uma condição mais crítica, fazendo com que sejam obrigadas a representar o máximo de informação que conseguem. Em valor negativo, a informação está muito distorcida,

e em valores mais altos, o modelo não tem muito o que aprender, pois emprega um esforço consideravelmente menor na reconstrução da informação.

Antes de qualquer outro resultado ser gerado, para checar o impacto do hiperparâmetro de tamanho de *batch* foi construído o resultado mostrado na Figura 24.

Figura 24 – BER para diferentes valores de *batch*.



Sabe-se que os valores de *batch* e a taxa de aprendizado são hiperparâmetros críticos, pois impactam bastante na performance de qualquer modelo de aprendizado profundo. A partir da Figura 24 podemos nos certificar de que um tamanho de *batch* igual a 2000 é uma escolha acertada, e que por isso, em todo este trabalho o referido valor foi usado.

3.3 Sistemas com codificação

Nesta subseção são apresentados os resultados das comparações entre o modelo de Autoencoder e sistemas convencionais codificados e transmitidos em canal AWGN. Os Autoencoders não possuem uma divisão em blocos, como já explicado na sub-seção 2.3.2. Estes,

na verdade, são otimizados para buscarem reproduzir na outra ponta o que foi enviado pelo transmissor, onde, para atingir esse objetivo, o modelo primeiro tenta representar os dados de entrada da maneira mais eficiente possível de acordo com, principalmente, o número de usos do canal, bem como outros parâmetros de sistema que são definidos.

Visando testar a equivalência dos Autoencoders com os SCD codificados, usou-se alguns cenários com diferentes parâmetros de tamanho de mensagem e taxa de codificação para obter os resultados.

Para implementar os resultados desta subseção utilizou-se o mesmo modelo de Autoencoder apresentado na subseção 3.2, com o mesmo número de camadas, e número de amostras de treino, validação e teste. A diferença é que o modelo foi treinado na SNR_{treino} = 5dB, pois foi percebida uma melhor performance, segundo se vê na Figura 23. Como métrica para avaliação foi dado preferência à taxa de erro de bit. Na Tabela 5 pode-se ver as configurações utilizadas nas simulações, onde *Soft* e *Hard*, bem como *Hard* e MLD (Decodificação por Máxima Verossimilhança) referem-se às formas de decodificação.

Tabela 5 – Diferentes parâmetros do Autoencoder e os *baselines* respectivos.

Número de mensagens (M)	$k = log_2(M)$	Usos do canal (n)	Taxa de Comunicação (R)	*Baseline*
16	4	8	$1/2$	CC 1/2 + 4-QAM
256	8	16		Soft e Hard
16	4	12	$1/3$	CC 1/3 + 4-QAM
256	8	24		Soft e Hard
16	4	7	4/7	Hamming (7,4)
				Hard e MLD

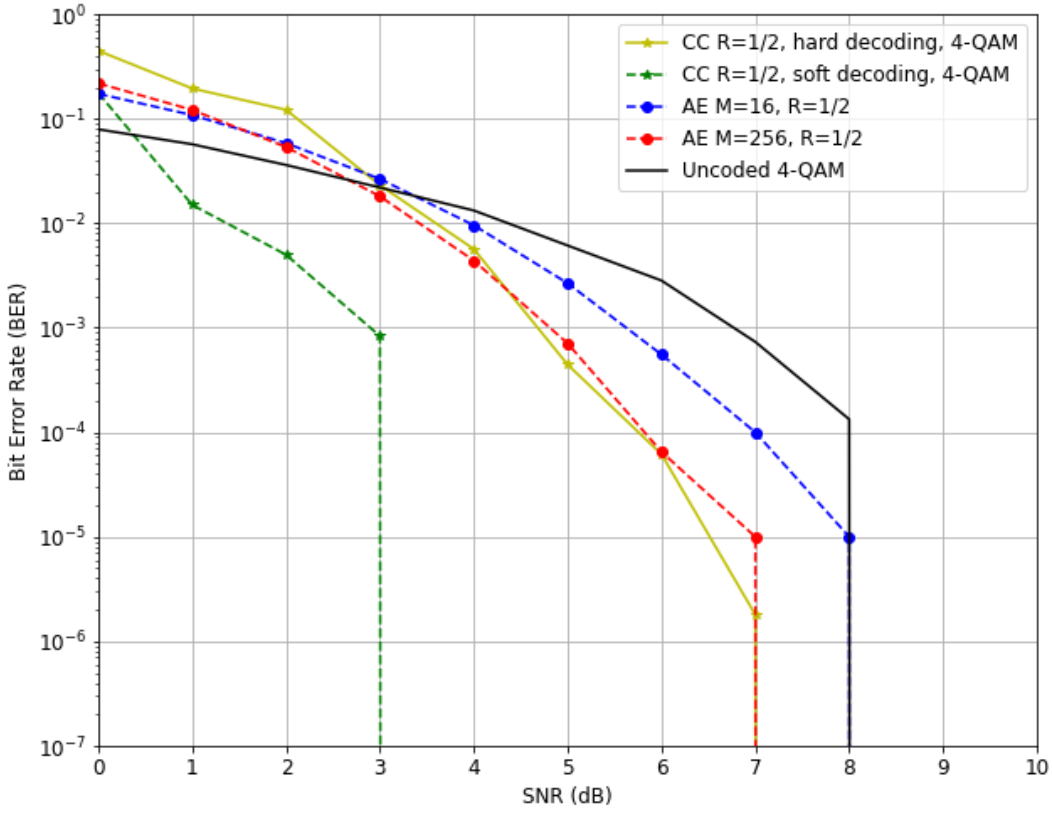
Os sistemas *baseline* utilizados foram os códigos Convolucionais, nas devidas taxas usadas também nos Autoencoders ($1/2$, $1/3$), com o esquema de modulação 4-QAM, bem como o código Hamming (7,4). A energia para transmitir uma mensagem foi mantida a mesma para o Autoencoder e para o *baseline*. Para o resultado com o esquema (7,4) (R=4/7) a intenção foi reproduzir a Figura 3.a de (O'SHEA; HOYDIS, 2017), um dos artigos-base desse projeto. Com isso, algumas lições foram aprendidas e serviram de inspiração para futuros trabalhos.

Os sistemas convencionais de codificação e modulação para esta subseção foram implementados na versão de licença acadêmica pela UFC do (MATLAB, 2021), com as funções nativas do *Communications toolbox*, com exceção do código Hamming (7,4), que foi implementado em Python, inclusive ambas as formas de decodificação.

Agora, passamos a ver os resultados obtidos. Na Figura 25 vemos a comparação

entre o modelo Autoencoder e o código convolucional para a taxa R = 1/2, com ambas as formas de decodificação *hard* e *soft*. O esquema de modulação 4-QAM foi usado na transmissão. O bloco de bits de informação selecionado para os CC foi K = 800 e o comprimento de restrição utilizado foi igual a 7.

Figura 25 – BER para comparação de Autoencoder com M={16, 256} e CC com R=1/2.



O que se pode notar é que, como esperado, a performance BER melhora à medida que se aumenta o tamanho da mensagem. A diferença de 1dB de ganho entre as duas performances pode ser explicada pelos graus de liberdade que o modelo possui quando o valor de M aumenta, fazendo com que consiga representar de modo mais fidedigno a quantidade de informação vinda do transmissor.

Outra importante observação é a performance comparável do Autoencoder em relação aos CC para decodificação *hard*, especialmente no intervalo de 0dB a 6dB. Em média, o Autoencoder teve desempenho muito semelhante aos CC, o que é bastante interessante quando se tem em mente que os códigos convolucionais são bastante otimizados. Em contrapartida, o

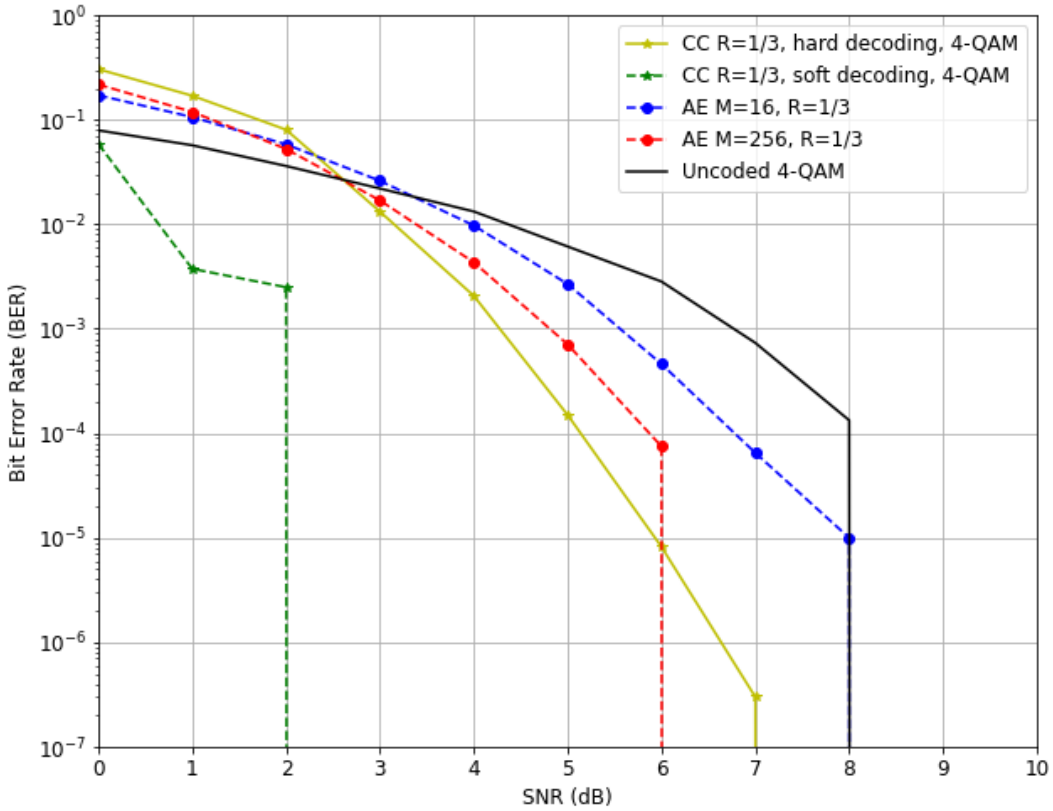
Autoencoder não possui um desempenho satisfatório em relação à decodificação *soft*, chegando a uma diferença de cerca de 2dB para quase todos os valores de BER.

Além disso, para $M = 256$, no intervalo de 0dB a aproximadamente 4,5dB o Autoencoder possui uma melhor performance que o CC com decodificação *hard*.

Na Figura 26 podemos ver agora a mesma comparação, mas com a taxa $R = 1/3$.

Nesse caso, o Autoencoder obteve um melhor resultado somente no intervalo de 0dB a quase 3dB. Contudo, é notável uma diferença de somente 1dB entre os CC para decodificação *hard* e o Autoencoder.

Figura 26 – BER para comparação de Autoencoder com M={16, 256} e CC com R=1/3.



Tendo em vista a avaliação dos Autoencoders em comparação com os códigos convolucionais, vislumbramos a possibilidade de comparar os dois métodos sendo os códigos convolucionais implementados com vários valores de bloco de bits de informação. Isso é o que está mostrado na Figura 27, para o caso em que $R = 1/2$.

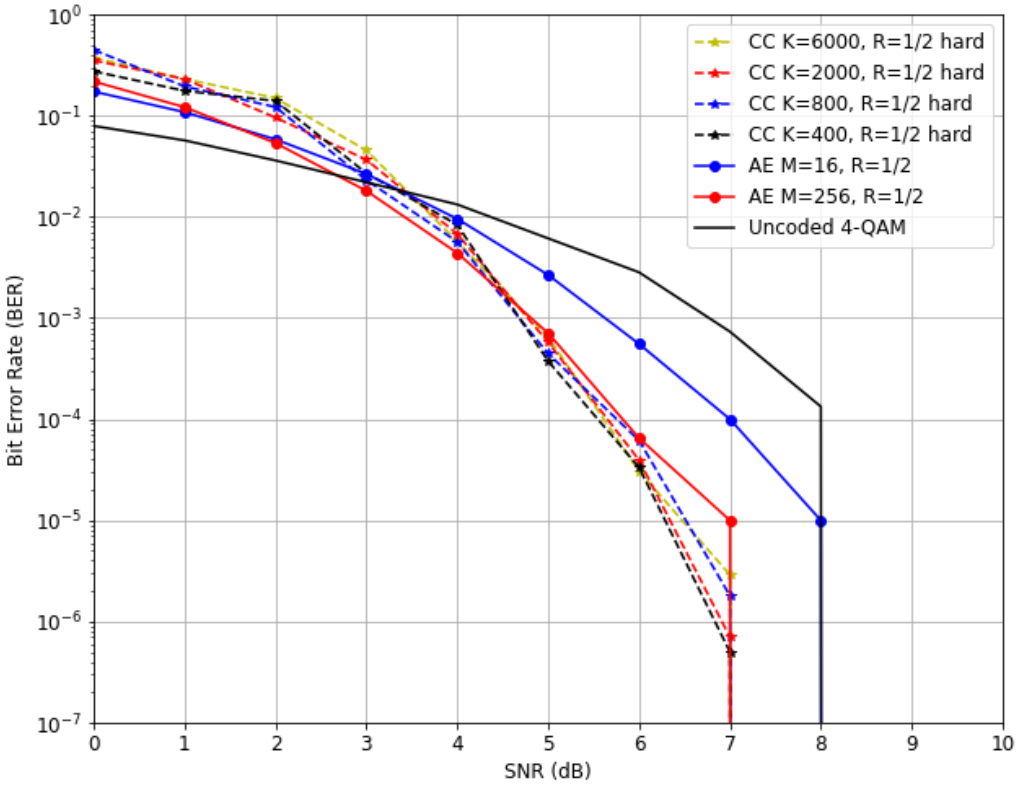
A motivação se encontra no fato já apresentado na Seção 2.2 de que os códigos

convolucionais têm alguma dependência no número de blocos de informação, que nos leva à hipótese de que os Autoencoders podem ser independentes do número de blocos de informação.

A princípio, nota-se no gráfico um desempenho um tanto semelhante, ou ao menos comparável, do Autoencoder com os códigos convolucionais para vários valores de tamanho de bloco. Para o valor de bloco $K = 400$ percebe-se um melhor desempenho dos CC.

No entanto, para outros valores de bloco e a partir de 5dB de SNR, em média a diferença diminui um pouco mais entre os dois métodos.

Figura 27 – BER para comparação de Autoencoder com M={16, 256} e CC com R=1/2 e K = {400,800,2000,6000}.



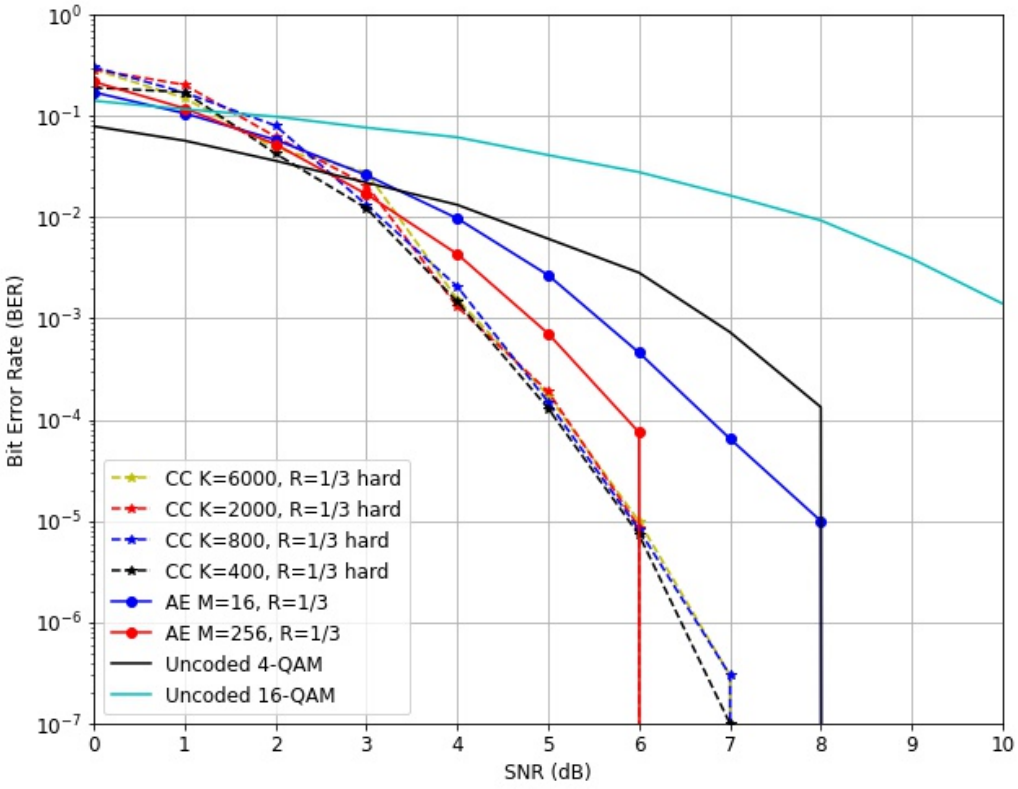
Esse resultado está um tanto diferente do esperado, principalmente no que diz respeito ao desempenho dos códigos convolucionais. É sabido que os códigos convolucionais possuem uma melhor performance para valores intermediários de bloco de informação, como K = {400, 800}, e que possuem um desempenho um pouco mais degradante para valores mais elevados de bloco. Dado que a implementação foi feita a partir de funções prontas do Matlab, não se pôde ter muito controle nesse quesito. Algumas horas foram gastas para tentar consertar,

mas não foi obtido êxito. Logo, em virtude do tempo, deixamos para passos futuros uma possível implementação própria dos códigos convolucionais.

Outra verificação consiste no melhor desempenho do Autoencoder para o intervalo de 0dB a 4dB. Isso já foi verificado na Figura 23, onde foi mostrado que o Autoencoder tende a ter um melhor desempenho em baixos valores de SNR, devido ao esforço das camadas escondidas para reconstruir a informação ser maior nesses valores.

Semelhante processo de comparação foi feito na Figura 28, mas agora para uma taxa $R = 1/3$.

Figura 28 – BER para comparação de Autoencoder com M={16, 256} e CC com R=1/3 e K = {400,800,2000,6000}.



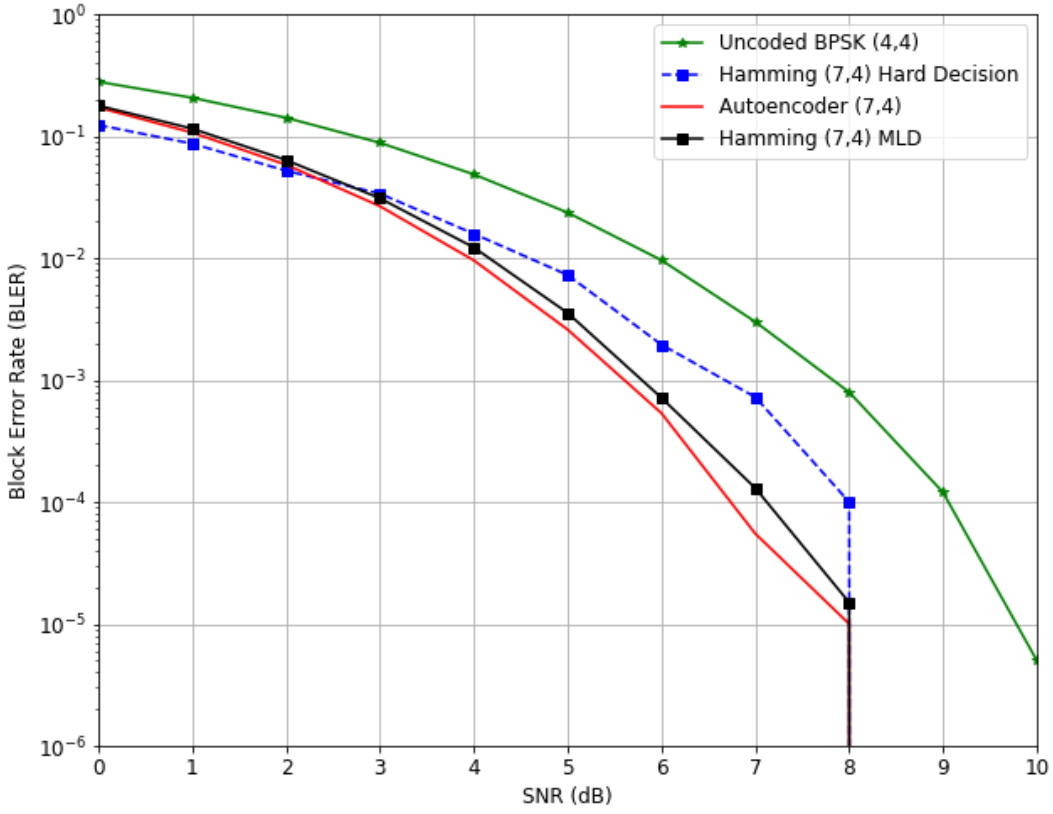
Dessa vez, não há nenhuma diferença significativa para os valores de tamanho de bloco dos CC. O desempenho do Autoencoder se mostra mais degradante do que o dos CC, principalmente para $M = 16$. Por outro lado, a princípio é possível observar uma certa independência nos resultados do Autoencoder em relação ao tamanho de bloco de bits de informação, uma vez que o modelo é alimentado a cada bloco de k bits. Isso é uma verificação

importante em relação ao desempenho dos CC, que possuem por natureza um desempenho mais degradante à medida que o bloco de bits aumenta consideravelmente.

Logo, a partir dos resultados encontrados até aqui para sistemas com codificação, pode-se ver que os Autoencoders têm o potencial de serem mais úteis em cenários onde se é necessário aumentar o bloco de bits de informação, cenário este onde os CC têm um desempenho mais degradante. Da mesma forma, pode-se concluir pelos resultados que em cenários com taxas de transmissão maiores, os Autoencoders podem vir a ter uma maior utilidade, o que é bastante interessante.

Agora, na Figura 29 vemos uma reprodução da Figura 3.a de (O'SHEA; HOYDIS, 2017). A implementação dos códigos Hamming para os dois tipos de decodificação, *hard* e MLD, foi feita utilizando a coleção de classes e funções que estão no repositório em (WICKERT, 2017).

Figura 29 – BLER versus SNR para comparação de Autoencoder com vários *baselines*.



O que se pode perceber é que o resultado encontrado aqui é bastante semelhante ao

resultado do artigo supracitado. A primeira diferença se encontra no desempenho do Autoencoder, que está visivelmente melhor que o código Hamming (7,4) para decodificação MLD. Isso se deve ao fato de que o treinamento do Autoencoder foi feito na SNR_{treino} = 5dB, já no artigo os autores usaram a SNR_{treino} = 7dB. Como foi obtido um desempenho melhor para esse valor de SNR_{treino} durante todas as outras simulações, foi mantido esse valor.

É importante destacar também que após uma implementação idêntica à que os autores mostram no artigo, os resultados não saíram iguais aos que foram obtidos por eles. Foi usado o mesmo número de camadas, de taxa de aprendizado, o mesmo otimizador, mesma taxa de código, bem como a mesma restrição de energia por bit, mas ainda assim os resultados apresentaram um desempenho inferior. Após alguns dias testando outras configurações, chegou-se a este último resultado. Todos essas configurações podem ser vistas na Seção 3.1.

Algumas diferenças, como a utilização da função de ativação *leaky-ReLU* ao invés da ReLU usada pelos autores, foram importantes para que o desempenho chegasse ao mesmo obtido pelos autores. A função de ativação *leaky-ReLU* é sabida ser uma melhor solução para o problema de *morte de neurônios*, explicado na seção 2.3.1, o que sugere que por alguma outra inconsistência do modelo deste trabalho, a função ReLU não estava contribuindo para uma solução ótima.

Isso sugere algumas das duas hipóteses: (1) o modelo utilizado neste presente trabalho possuía alguma inconsistência em relação ao modelo usado pelos autores no artigo ou, menos provável, (2) os autores não otimizaram seus hiperparâmetros e funções de ativação de forma mais cautelosa.

4 CONCLUSÕES

Este trabalho apresentou uma avaliação comparativa entre um sistema baseado na arquitetura Autoencoder com redes neurais profundas e sistemas convencionais de comunicação, mais especificamente modulação e codificação. O trabalho foi desenvolvido com forte influência de outros trabalhos como o de (O'SHEA; HOYDIS, 2017) e (ERPEK *et al.*, 2020). Alguns sistemas clássicos e configurações diferentes dos Autoencoders foram adotados na geração dos resultados.

O principal foco deste trabalho consiste em analisar o desempenho dos Autoencoders como um sistema de transmissão e recepção otimizado para comunicação ponta-a-ponta, num cenário de usuário único, sem informação do estado do canal e para o modelo de canal AWGN.

Para sistemas não codificados, os Autoencoders foram comparados com os esquemas clássicos de modulação BPSK e QPSK, sendo configurados para terem a mesma taxa de comunicação desses sistemas. Foi visto através da BLER que para o sistema em que $R = 1$ [bit/uso do canal] os Autoencoders possuem um desempenho bastante satisfatório e comparável com os convencionais. O modelo é ligeiramente melhor para baixos valores de SNR e para a configuração com $M = 256$ possui um desempenho bem superior em valores de SNR acima de 3dB. Observou-se também que com o aumento do número de mensagens e de usos do canal o desempenho dos Autoencoders melhora gradativamente. Para $R = 2$ [bits/uso do canal], o sistema QPSK se sobressai com uma melhor performance no intervalo de 0-9dB, ainda que para $M = 256$ o Autoencoder se comporte bem. Além disso, os resultados de constelação mostraram que os Autoencoders conseguem otimizar e aprender padrões de símbolos que se ajustam temporalmente para compensar os erros causados pelo canal.

Em sistemas com codificação, os Autoencoders foram comparados nas taxas $R = 1/2$ e $R = 1/3$ com sistemas convencionais, onde para estes foram considerados os cenários com Código Convolucional (codificação de canal) nas devidas taxas equivalentes aos Autoencoders e com o esquema de modulação 4-QAM. Ambas as formas de decodificação - *soft* e *hard* - foram usadas nos CC. Foi observado que para uma decodificação *hard* dos CC os Autoencoders (para $M = 256$) possuem um desempenho um tanto comparável, especialmente para a taxa $R = 1/2$. Além disso, foi concluído a partir dos resultados que em cenários com blocos de informação e taxas de transmissão maiores os Autoencoders podem vir a ter uma maior utilidade. Isso mostra a capacidade promissora dos Autoencoders frente a sistemas otimizados como os CC.

Como futuras direções e/ou extensões desse trabalho pode-se citar os seguintes

pontos:

1. Avaliar cenários de canais com desvanecimento, uma vez que aqui só foi abordado o modelo AWGN, bem como cenários multi-usuário, tendo em vista principalmente a importância dos sistemas MIMO para o 5G.
2. Estender a discussão em comunicação ponta-a-ponta com DL para outras arquiteturas e configurações. Tem surgido mais recentemente um maior interesse em arquiteturas como a de GANs, por exemplo, que se mostram também promissoras nessa área.
3. Explorar a adoção de outras fontes de informação como imagem, áudio e texto ao invés de apenas um *stream* de bits, seria uma outra direção possível, tendo em vista a natureza dos resultados ser mais interessante.
4. Investigar formas de obter modelos capazes de se adaptar de maneira satisfatória aos mais variados cenários e condições de canal, em tempo real, é um aspecto muito relevante e que pode ser uma futura direção, mais desafiadora diga-se de passagem.
5. Estender as simulações para implementações em hardware é outra possível direção que expõe aos inúmeros desafios de toda e qualquer implementação prática de sistemas de comunicação.

REFERÊNCIAS

ABADI, M. *et al.* Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016. Disponível em: http://arxiv.org/abs/1603.04467.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. Springer Street, NYC, NY: Springer, 2006.

BOURTSOULATZE, E.; KURKA, D. B.; GÜNDÜZ, D. Deep joint source-channel coding for wireless image transmission. In: **ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. [*S. l.*: *s. n.*], 2019. p. 4774–4778.

CAVALCANTI, R.; MACIEL, T.; FREITAS, W.; SILVA, Y. **Comunicação Móvel Celular**. Rio de Janeiro: Elsevier, 2018.

CHOLLET, F. *et al.* Keras. GitHub, 2015. Disponível em: https://github.com/fchollet/keras.

COVER, T. M.; THOMAS, J. A. **Elements of Information Theory**. River Street, Hoboken, New Jersey: John Wiley & Sons, Inc., 2006.

DÖRNER, S.; CAMMERER, S.; HOYDIS, J.; BRINK, S. t. Deep learning based communication over the air. **IEEE Journal of Selected Topics in Signal Processing**, v. 12, n. 1, p. 132–143, 2018. Disponível em: 10.1109/JSTSP.2017.2784180.

ERPEK, T.; O'SHEA, T. J.; SAGDUYU, Y. E.; SHI, Y.; CLANCY, T. C. Deep learning for wireless communications. abs/2005.06068, 2020. Disponível em: https://arxiv.org/abs/2005.06068.

GOLDSMITH, A. Joint source/channel coding for wireless channels. In: **1995 IEEE 45th Vehicular Technology Conference. Countdown to the Wireless Twenty-First Century**. Chicago, IL, USA: IEEE, 1995. v. 2, p. 614–618.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. The MIT Press, 2016. Disponível em: http://www.deeplearningbook.org.

GOODFELLOW, I. J.; POUGET-ABADIE, J.; MIRZA, M.; XU, B.; WARDE-FARLEY, D.; OZAIR, S.; COURVILLE, A.; BENGIO, Y. Generative adversarial nets. In: **Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2**. Cambridge, MA, USA: MIT Press, 2014. p. 2672–2680.

GÉRON, A. **Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. Sebastopol, California: O'Reilly Media, Inc., 2017.

HAMMING, R. W. Error detecting and error correcting codes. **The Bell System Technical Journal**, v. 29, n. 2, p. 147–160, 1950.

HAYKIN, S. **Digital Communication Systems**. New York: John Wiley & Sons, 2014.

HAYKIN, S.; MOHER, M. **Sistemas de Comunicação**. New York: John Wiley & Sons, Inc., 2009.

HINTON, G. E.; ZEMEL, R. S. Autoencoders, minimum description length and helmholtz free energy. In: . San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. (NIPS'93), p. 3–10.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. 2015. Disponível em: http://arxiv.org/abs/1412.6980.

KURKA, D. B.; GÜNDÜZ, D. Deepjscc-f: Deep joint-source channel coding of images with feedback. abs/1911.11174, 2019. Disponível em: http://arxiv.org/abs/1911.11174.

MATLAB. **versão 9.11.0 (R2021b), licença acadêmica (UFC)**. Natick, Massachusetts: The MathWorks Inc., 2021.

MCCULLOCH, W.; PITTS, W. A logical calculus of ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, p. 127–147, 1943.

O'SHEA, T. J.; KARRA, K.; CLANCY, T. C. Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention. abs/1608.06409, 2016. Disponível em: http://arxiv.org/abs/1608.06409.

O'SHEA, T.; HOYDIS, J. An introduction to deep learning for the physical layer. **IEEE Transactions on Cognitive Communications and Networking**, v. 3, n. 4, p. 563–575, 2017.

PROAKIS, J. G. **Digital Communications**. NYC, USA: McGraw-Hill, 1995.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning Representations by Back-propagating Errors. **Nature**, v. 323, p. 533–536, 1986. Disponível em: http://www.nature.com/articles/323533a0.

SHANNON, C. E. A mathematical theory of communication. v. 27, 1948.

SKLAR, B. **Digital Communications**. New Jersey: Prentice Hall, 2001.

SWEENEY, P. **Error Control Coding: From theory to practice**. Baffins Lane, Chichester, England: John Wiley & Sons, Inc., 2002.

WICKERT, M. **scikit-dsp-comm**. GitHub, 2017. Disponível em: https://github.com/charlespwd/project-title.

ZHANG, C.; PATRAS, P.; HADDADI, H. Deep learning in mobile and wireless networking: A survey. **IEEE Communications Surveys Tutorials**, v. 21, n. 3, p. 2224–2287, 2019.

ZHUANG, F.; QI, Z.; DUAN, K.; XI, D.; ZHU, Y.; ZHU, H.; XIONG, H.; HE, Q. A comprehensive survey on transfer learning. **CoRR**, abs/1911.02685, 2019. Disponível em: http://arxiv.org/abs/1911.02685.