TRABALHO DE GRADUAÇÃO

Desenvolvimento de Rede Neural Artificial Recorrente em FPGA para Previsão *online* de Oportunidades em Transmissões Oportunísticas em Redes de Comunicação *Wireless*

Mariana Rodrigues Makiuchi

Brasília, Dezembro de 2018

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Desenvolvimento de Rede Neural Artificial Recorrente em FPGA para Previsão *online* de Oportunidades em Transmissões Oportunísticas em Redes de Comunicação *Wireless*

Mariana Rodrigues Makiuchi

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Dr. Marcus Vinicius Lamar, CIC/UnB _____
Orientador

Prof. Dr. Jacir Luiz Bordim, CIC/UnB _____
Examinador interno

Prof. Dr. Marcos Fagundes Caetano, CIC/UnB _____
Examinador interno

Prof. Dr. Marcelo Grandi Mandelli, CIC/UnB _____
Examinador interno

Brasília, Dezembro de 2018

FICHA CATALOGRÁFICA

MARIANA, RODRIGUES MAKIUCHI
Desenvolvimento de rede neural artificial recorrente em FPGA para previsão *online* de oportunidades em transmissões oportunísticas em redes de comunicação *wireless*

[Distrito Federal] 2018.

xiv, 67p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2018). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Redes Neurais Artificiais Recorrentes 2. Rádios definidos port software 3. Comunicação oportunística
4. FPGA 5. Redes sem fio

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

MAKIUCHI, MARIANA RODRIGUES (2018). Desenvolvimento de rede neural artificial recorrente em FPGA para previsão *online* de oportunidades em transmissões oportunísticas em redes de comunicação *wireless*. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$11, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 81p.

CESSÃO DE DIREITOS

AUTOR: Mariana Rodrigues Makiuchi

TÍTULO DO TRABALHO DE GRADUAÇÃO: Desenvolvimento de rede neural artificial recorrente em FPGA para previsão *online* de oportunidades em transmissões oportunísticas em redes de comunicação *wireless*.

GRAU: Engenheiro ANO: 2018

Mariana Rodrigues Makiuchi

DF 425 KM 0.5 Cond. Vivendas Serranas, Sobradinho, Brasília.

73092-900 Brasília – DF – Brasil.

Dedicatória

A todas as pessoas que passaram pela minha vida.

Mariana Rodrigues Makiuchi

Agradecimentos

Primeiramente, gostaria de agradecer aos meus pais por terem me apoiado sempre em minhas batalhas e por terem me aconselhado com um carinho especial durante a graduação. Agradeço também a minha irmã Camila por ter escutado todas as minhas histórias, reclamações e piadas não só ao longo da minha graduação, mas desde sempre.
Deixo aqui agradecimentos especiais ao Prof. Marcus Vinicius Lamar por ter me orientado desde o meu segundo semestre de UnB, quando eu ainda era uma aluna do Programa Jovens Talentos para a Ciência. Obrigada por ter aberto as portas da pesquisa científica para mim, pela paciência e pelos ensinamentos.
Aos professores que concordaram em participar da banca de avaliação deste trabalho, pela disponibilidade e pelos comentários e sugestões que fizeram essa pesquisa atingir seu potencial máximo.
Agradeço também aos meus amigos do apoio técnico do departamento de Ciência da Computação da UnB: Aline, Camila, Chris, Jonatas e Valdy; por terem proporcionado um ambiente de acolhimento e de estudo dentro do departamento e por toda a ajuda (relacionada à computação ou não) que me deram. Aos estagiários da secretaria: Luhan e Marlo, pela companhia nos momentos em que estávamos no departamento e aos servidores Luis e Maria por me darem perspectivas honestas sobre quais caminhos seguir.
Agradeço aos meus melhores amigos Ju e Rod por me escutarem, saírem comigo e serem os melhores do mundo sempre.
Agradeço também às minhas colegas de curso, Débora e Juliana, que compartilharam comigo sua amizade, relatos e desabafos. Graças a vocês, nunca me senti isolada.
Agradeço ao meu namorado, Tatsuya, pelas viagens que nos levaram longe do estresse.
Agradeço aos meus gatos por estarem ao meu lado durante a execução de todo este trabalho.
Por fim, gostaria de agradecer àqueles que, mesmo tendo deixado este mundo, marcaram a minha vida e me tornaram uma pessoa melhor e mais forte para enfrentar meus desafios.

Mariana Rodrigues Makiuchi

RESUMO

A popularização do uso de dispositivos móveis com acesso a redes sem fio e a atual conjuntura de alocação estática do espectro eletromagnético são provavelmente as principais causas da escassez desse recurso. Para evitar o congestionamento em algumas faixas de frequência e promover o uso uniforme do espectro, vários pesquisadores e instituições estão considerando tornar o acesso dinâmico ao espectro uma realidade. Entretanto, um dos maiores desafios da mudança de política de acesso é o desenvolvimento de uma tecnologia capaz de identificar corretamente, em tempo real, oportunidades de transmissão. Neste contexto, este trabalho propõe um sistema em FPGA baseado em redes neurais recorrentes do tipo Elman para realizar o processamento de sinais eletromagnéticos capturados por um rádio definido por *software* e identificar oportunidades de transmissão a partir deles. Além disso, este trabalho apresenta metodologias para tornar a rede neural adaptável. Diversos experimentos foram realizados para avaliar o sistema descrito em *hardware* em termos de demanda por recursos computacionais, tempo de processamento e precisão de resultados. A partir desses experimentos, foi possível observar que, em relação ao mesmo sistema definido em *software*, o tempo de processamento do sistema descrito em *hardware* é cerca de 7,4 vezes mais rápido, mas seus resultados apresentaram um erro máximo de 24,4% quando comparado ao sistema em *software*, que, apesar de ser um valor considerável, não é suficiente para interferir na identificação de oportunidades realizada pelo sistema.

Palavras Chave: redes neurais artificiais recorrentes, rádios definidos por *software*, comunicação oportunística, *field programmable gate array*, redes sem fio.

ABSTRACT

The popularization of the usage of mobile devices with access to wireless networks and the current conjuncture of static spectrum allocation are probably the main causes of spectrum scarcity. In order to avoid the congestion in some of the frequency bands and to grant the uniformed usage of the spectrum, many researchers and institutions consider making the dynamic access to the spectrum a reality. However, one of the greatest challenges of changing the access policy is the development of a technology capable of correctly identifying, in real time, transmission opportunities. In this context, this work proposes a system in FPGA based on Elman recurrent neural networks to execute the processing of electromagnetic signals acquired by a software defined radio and to identify transmission opportunities from these signals. Moreover, this work presents methodologies to make the network adaptable. Several experiments were conducted to evaluate the system described in hardware in terms of computational resources, processing time and results' precision. From these experiments, it was possible to prove that, compared to the same system defined in software, the processing time of the system described in hardware is about 7.4 times faster, but its results have a maximum error of 24.4% when compared to the system defined in software, which, despite being a substantial quantity, it is not enough to interfere in the opportunities identification provided by the system.

Keywords: artificial recurrent neural networks, software defined radios, opportunistic communication, field programmable gate array, wireless networks.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

b	Bias
d	Resposta desejada
e	Erro de um neurônio
E	Energia
f	Frequência
$\mathcal{F}$	Transformada de Fourier
I	Sinal de fase do espectro eletro-magnético
j	Unidade imaginária
q	Sinal quantizado
Q	Sinal de quadratura do espectro eletro-magnético
v	Campo local induzido
w	Peso sináptico
W	Janela de Hamming
y	Saída de um neurônio
z^{-1}	Atraso unitário

Símbolos Gregos

α	Constante de momento
δ	Gradiente local
Δ	Variação de pesos sinápticos
ε	Energia do erro
η	Coeficiente de aprendizado
ξ	Limiar de quantização
φ	Função de ativação
ω	Frequência

Grupos Adimensionais

Subscritos

av	médio

Sobrescritos

$'$	Derivada

Siglas

ANATEL	Agência Nacional de Telecomunicação
DFT	Transformada de Fourier Discreta - *Discrete Fourier Transform*
DIT	Decimação no Tempo -*Decimation in Time*
DSA	Acesso Dinâmico ao Espectro - *Dynamic Spectrum Access*
FCC	*Federal Communications Commission*
FFT	Transformada Rápida de Fourier - *Fast Fourier Transform*
FPGA	*Field Programmable Gate Array*
IP	Propriedade Intelectual - *Intelectual Property*
LAN	*Local Area Network*
MEF	Máquina de Estados Finitos
MIMO	*Multiple Input Multiple Output*
NARX	*Nonlinear Autoregressive Network with Exogenous Input*
NTIA	*National Telecommunications and Information Administration*
RAM	*Random Access Memory*
RDS	Rádio Definido por *Software*
RNA	Rede Neural Artificial
RNR	Rede Neural Recorrente
USRP	*Universal Software Radio Peripheral*
WLAN	*Wireless Local Area Network*

Capítulo 1

Introdução

Com o crescimento do mercado da computação vestível [6], da pesquisa na área da internet das coisas [7] e do uso constante da comunicação *wireless* por meio de *smartphones* e *tablets* [8], a escassez do espectro eletromagnético se tornou uma das preocupações de acadêmicos e de membros de órgãos reguladores do espectro, como, por exemplo a *Federal Communications Commission* (FCC) nos Estados Unidos [9]. Além disso, a atual conjuntura de alocação estática do espectro, quando somada ao fator de crescimento da utilização do espectro supracitado, amplifica a preocupação geral em relação à provável futura escassez do espectro.

Atualmente, a alocação das faixas de frequência do espectro eletromagnético é realizada de forma estática, ou seja, cada faixa de frequência é reservada para um determinado serviço de forma regulada por instituições governamentais. Nos Estados Unidos, a instituição reguladora é a NTIA [10] e, no Brasil, a responsável por essa divisão do espectro é a Anatel [11]. Nessa circunstância, a utilização do espectro não é uniforme, ou seja, existem faixas de frequência congestionadas enquanto outras estão vazias [12]. Portanto, vários pesquisadores [13] [14] e instituições [15] estão considerando e desenvolvendo novas tecnologias para tornar o acesso dinâmico ao espectro uma realidade.

Em uma configuração de acesso dinâmico, usuários secundários, ou seja, usuários à princípio não licenciados para transmitir a uma determinada frequência, poderão explorar faixas de frequência de acordo com a ocupação do espectro a um dado instante de tempo. Entretanto, uma das preocupações a respeito do acesso dinâmico é a colisão entre os usuários primário e secundário. Uma vez que o usuário primário tem prioridade sobre a transmissão a uma determinada frequência, é importante evitar que usuários secundários interfiram na transmissão dos usuários primários.

Uma das soluções para permitir o acesso dinâmico do espectro é o uso de rádios cognitivos [16] [17], definidos como rádios que possuem conhecimento sobre o ambiente no qual estão inseridos e que conseguem adaptar seus parâmetros de transmissão de acordo com o ambiente para permitir a melhor comunicação possível. Esses rádios conseguem realizar um sensoriamento mais inteligente do espectro e detectar os canais de transmissão disponíveis, sendo, assim, uma opção viável no desenvolvimento de políticas de acesso dinâmico ao espectro.

Outra solução para realizar o acesso dinâmico seria o uso de inteligência artificial. Ao longo das últimas décadas, cientistas da computação e o mercado tecnológico observaram um grande impulso na pesquisa e no desenvolvimento de redes neurais artificiais. Atualmente, essas estruturas de inteligência artificial são capazes de realizar atividades de diversas naturezas e são reconhecidas por sua capacidade em solucionar problemas de classificação e de identificação de padrões.

Neste trabalho, considera-se o cenário em que dois usuários primários, ou seja, usuários licenciados, se comunicam em um dado canal enquanto que um usuário secundário, ao qual o sistema desenvolvido neste trabalho está incorporado, monitora as transmissões de maneira a viabilizar a previsão de oportunidades neste canal. Um diagrama desse cenário é indicado pela Figura 1.1.

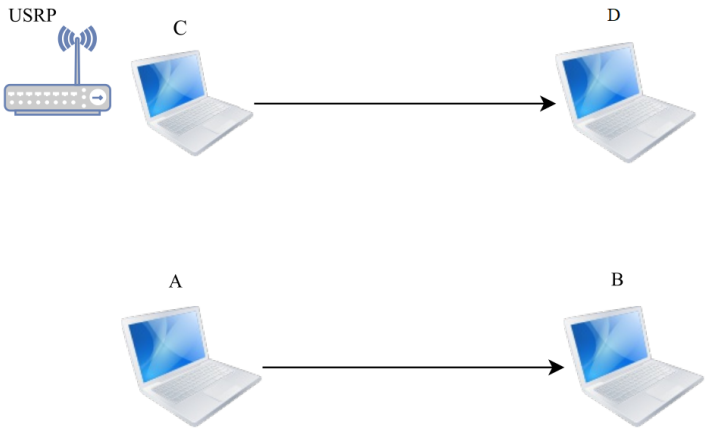

Figura 1.1: Um dos cenários de transmissão considerados neste trabalho, em que os nós A e B indicam os usuários primários e os nós C e D, os usuários secundários, sendo C o usuário que monitora o canal [1]

Além do cenário descrito graficamente pela Figura 1.1, cenários nos quais há transmissões entre dois pares de usuários primários também foram considerados, bem como ambientes com e sem interferência.

O trabalho apresentado por este relatório é uma continuação do trabalho descrito em [1], no qual foi desenvolvido um sistema em *software* capaz de identificar oportunidades de transmissão em redes de comunicação sem fio com taxa de acerto média ponderada acima de 80% nos cenários descritos pela Figura 1.1. O trabalho apresentado por este relatório objetiva a implementação do sistema definido em [1] em *hardware* de forma completa, ou seja, considerando a previsão *online* de oportunidades de transmissão e a adaptação do sistema de acordo com variações ambientais.

1.1 Definição do problema

Considerando o atual contexto da utilização do espectro eletromagnético, a criação de um sistema capaz de se ajustar automaticamente de acordo com a ocupação do espectro electromagnético para prever oportunidades de transmissão em redes de comunicação sem fio é um desafio. Além disso, o desenvolvimento desse sistema em *hardware* para viabilizar um tempo de processamento

menor em relação ao de uma implementação em *software* atribui um grau de dificuldade adicional a esse desafio.

1.2 Objetivos do projeto

Tendo em vista a versatilidade e o poder computacional das redes neurais, este trabalho propõe o desenvolvimento, em *hardware*, de uma rede neural recorrente capaz de identificar oportunidades de transmissão em redes de comunicação sem fio e se adaptar de acordo com variações ambientais, possibilitando, assim, o acesso dinâmico ao espectro eletromagnético.

1.2.1 Objetivos específicos

Os objetivos específicos deste trabalho consistem na execução dos seguintes passos:

1. **Estudos em redes neurais.** Uma vez que este trabalho se baseia no desenvolvimento de sistemas que se apoiam no uso de redes neurais, foi necessário realizar um estudo detalhado sobre essas estruturas de inteligência artificial. Em particular, pode-se citar os estudos realizados sobre os algoritmos de aprendizado, a estrutura recorrente das redes neurais e a utilização do *Neural Network Toolbox* [18] do Matlab.
2. **Desenvolvimento de sistema de previsão de oportunidades em *hardware*.** Esta etapa consistiu no desenvolvimento de um sistema completo em *hardware*, que realiza um processamento dos sinais do espectro eletromagnético e, a partir da ocupação desse espectro, indica se há oportunidade de transmissão em um dado instante de tempo.
3. **Estudo sobre adaptabilidade da rede neural.** Para verificar a possibilidade de adaptar os parâmetros da rede neural de forma *online* e de acordo com alterações na ocupação do espectro eletromagnético, foram realizados estudos sobre algoritmos e métodos de adaptação da rede neural.
4. **Validação dos resultados.** Os resultados obtidos durante os testes são analisados de forma detalhada para mensurar o desempenho do sistema desenvolvido e para identificar aspectos que podem ser melhorados.

1.3 Apresentação do manuscrito

Este relatório é dividido em cinco capítulos, incluindo este, introdutório. A estrutura dos demais capítulos é descrita a seguir.

- O Capítulo 2 detalha a fundamentação teórica necessária para a compreensão do trabalho desenvolvido com ênfase nas estruturas e modelos de redes neurais artificiais.

- O Capítulo 3 apresenta a metodologia proposta para atingir os objetivos deste trabalho e elenca todos os procedimentos adotados ao longo de seu desenvolvimento.
- O Capítulo 4 expõe os resultados parciais e finais adquiridos nos experimentos aplicados sobre sistema proposto.
- O Capítulo 5 conclui este relatório apresentando uma síntese dos pontos principais do trabalho desenvolvido e os trabalhos futuros que podem ser realizados.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta alguns conceitos envolvendo redes neurais artificiais, a transformada de Fourier, representações numéricas em computadores e dois dispositivos utilizados neste trabalho: o rádio definido por *software* e o *field programmable gate array* (FPGA). A Seção 2.1 trata sobre o modelo de redes neurais artificiais e discorre sobre os métodos de aprendizagem, o algoritmo de treino *backpropagation* e redes neurais recorrentes. As Seções 2.2 e 2.3 tratam da transformada de Fourier contínua e discreta respectivamente. Em particular, a Seção 2.3 ainda discorre sobre o algoritmo de transformada rápida de Fourier. A Seção 2.4 apresenta algumas formas de representação numérica em um computador, como o ponto fixo para representar números fracionários e o complemento de dois para representar números negativos. Por fim, a Seção 2.5 define os dispositivos utilizados neste trabalho: o rádio definido por *software* USRP N210 e o FPGA Altera Cyclone IV incorporado ao kit de desenvolvimento DE2-115.

2.1 Redes Neurais Artificiais

Redes neurais artificiais (RNAs), ou simplesmente redes neurais, podem ser definidas como estruturas computacionais que executam corretamente uma tarefa sem que sejam programadas previamente para efetuá-la. Neste sentido, afirma-se que as redes neurais "aprendem" a maneira correta de execução de certa atividade [19]. Redes neurais podem ser aplicadas para solucionar uma ampla variedade de problemas, tais como a classificação ou o agrupamento de padrões ou a solução de problemas de otimização [20].

Essas estruturas compõem uma das áreas de estudo da inteligência artificial e são inspiradas no sistema nervoso de seres vivos [2]. Portanto, as RNAs são compostas por estruturas similares a neurônios, interconectados por "sinapses". Geralmente, esses neurônios artificiais são agrupados em camadas, como mostra a Figura 2.1.

A Figura 2.1 representa uma RNA com três entradas, duas camadas escondidas, cada uma com quatro neurônios, e uma camada de saída com apenas um neurônio. A quantidade de entradas e de neurônios na camada de saída dependem das dimensões de entrada e saída do problema a ser solucionado. Entretanto, a quantidade de neurônios em uma camada escondida está diretamente

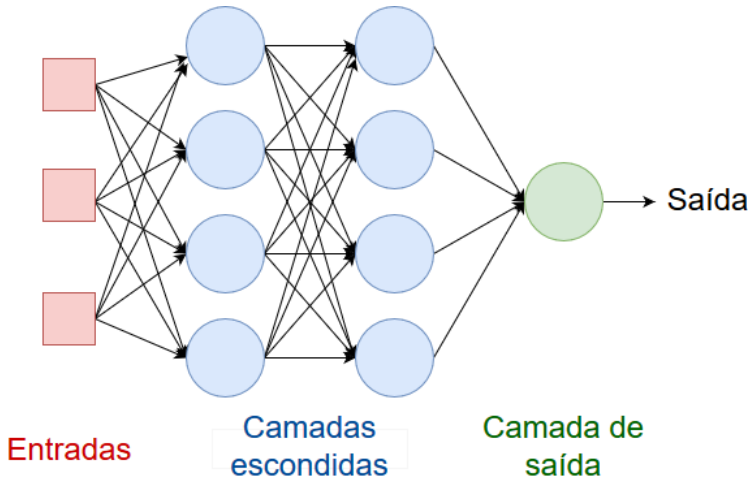

Figura 2.1: Diagrama básico de uma rede neural artificial

relacionada à complexidade do problema.

Além disso, o número de camadas escondidas também é um fator que depende do problema. RNAs que não possuem camadas escondidas são capazes de solucionar apenas problemas linearmente separáveis [21]. Já RNAs com uma ou múltiplas camadas escondidas conseguem, respectivamente, descrever funções contínuas [22] ou qualquer função [23].

Cada um dos círculos presentes na Figura 2.1 representa um neurônio artificial e, cada um dos quadrados, uma entrada da rede. As setas que conectam entradas e neurônios ou neurônios entre si, representam as sinapses entre esses elementos, às quais é atribuído um valor, também denominado por peso sináptico. O conjunto de pesos da rede define o seu conhecimento e a alteração de seus valores determina o processo de aprendizado, como será explicado matematicamente no decorrer desta seção. As definições matemáticas e da estrutura do neurônio artificial apresentadas no decorrer desta seção são embasadas pelo trabalho de Haykin [2].

2.1.1 Modelo de neurônio artificial

Cada neurônio artificial organizado em forma de camadas, como mostrado pela Figura 2.1, é modelado por um conjunto de operações matemáticas, representadas graficamente pela Figura 2.2, que mostra o modelo de um neurônio artificial.

Na Figura 2.2, a variável x_i com $i = 1, .., m$ representa a i-ésima entrada do neurônio k. Essas m entradas podem ser as próprias entradas da rede ou então as saídas dos neurônios da camada anterior, a depender do posicionamento do neurônio k na estrutura da rede, como pode ser observado na Figura 2.1. Já a entrada x_0 é uma entrada fixa de valor 1 relacionada ao *bias* do neurônio.

Os pesos sinápticos w_{ki} são valores numéricos, positivos, negativou ou nulos, que indicam a força da conexão entre a entrada x_i e o neurônio k. O peso sináptico w_{k0} é também denominado por bias do neurônio k ou b_k por ter como entrada sempre o valor fixo 1.

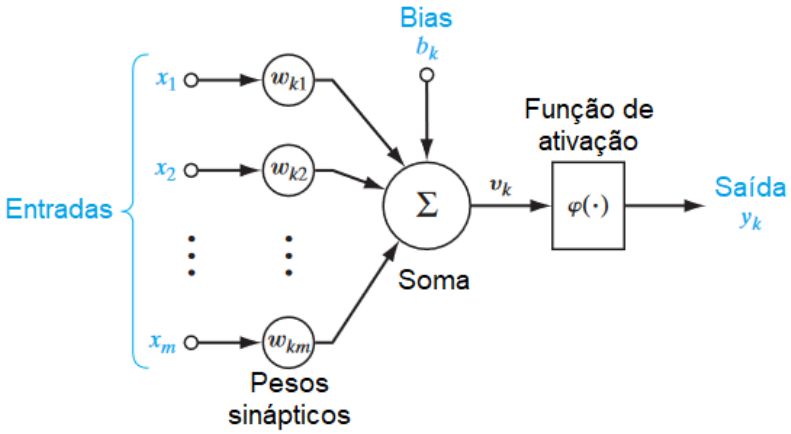

Figura 2.2: Modelo de neurônio artificial [2]

O papel do bias é similar ao papel do coeficiente linear de uma função linear [2]: ajustar a saída da função. No caso representado pela Figura 2.2, o bias b_k auxilia no ajuste da saída y_k da função de ativação $\varphi(\cdot)$.

O processamento matemático interno a um dado neurônio k representado pela Figura 2.2 consiste, então, em multiplicar entradas x_i $(i = 0, 1, 2, ..., m)$ pelos respectivos pesos w_{ki} e somar esses produtos por meio do operador indicado por "soma", gerando o potencial interno v_k, também chamado de campo local induzido, que é aplicado na função de ativação $\varphi(\cdot)$ do neurônio k, gerando, por fim, y_k como a saída desse neurônio.

Assim, escrevendo o procedimento da Figura 2.2 em termos de equações, v_k é definido como sendo

$$v_k = \sum_{i=0}^{m} w_{ki} \cdot x_i, \tag{2.1}$$

em que o índice i igual a zero é relativo ao bias, também representado na Figura 2.2 e discutido anteriormente.

Deste modo, a saída do neurônio k, ou seja, y_k, como representado pela Figura 2.2, é dada por

$$y_k = \varphi_k(v_k), \tag{2.2}$$

em que o símbolo k em $\varphi_k(\cdot)$ é utilizado para explicitar a função de ativação específica do neurônio k. A Equação (2.2) é a simples aplicação da função de ativação $\varphi_k(\cdot)$ sobre a soma ponderada v_k, previamente calculada pela Equação (2.1). As funções de ativação podem ser de vários tipos, sendo os mais comuns listados na Figura 2.3.

As funções de ativação $\varphi(\cdot)$ listadas na Figura 2.3 são definidas matematicamente, na ordem em que aparecem na figura e, em termos do campo local induzido v [2], como sendo

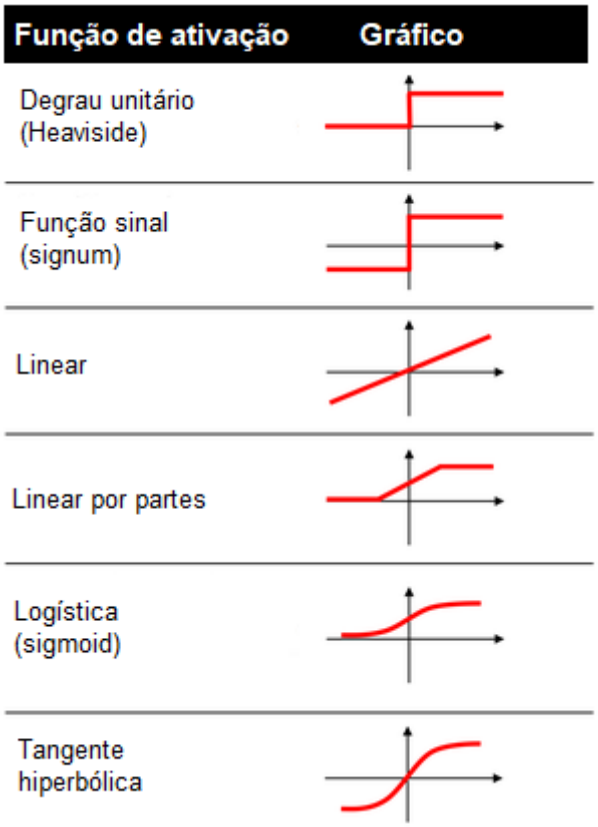

Figura 2.3: Funções de ativação mais comuns [3]

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v < 0 \end{cases} \tag{2.3}$$

$$\varphi(v) = \begin{cases} 1 & \text{se } v > 0 \\ 0 & \text{se } v = 0 \\ -1 & \text{se } v < 0 \end{cases} \tag{2.4}$$

$$\varphi(v) = v \tag{2.5}$$

$$\varphi(v) = \begin{cases} 1, & v \geq \frac{1}{2} \\ v, & \frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases} \tag{2.6}$$

$$\varphi(v) = \frac{1}{1 + e^{-av}}, \text{ a = constante} \tag{2.7}$$

$$\varphi(v) = tanh(v) \tag{2.8}$$

O processo indicado pela Figura 2.2 deve ocorrer camada a camada, assim, todos os neurônios que pertencem à mesma camada do neurônio k sofrem o mesmo processo interno, gerando um conjunto de saídas y_g ($g = 1, 2, ..., k, ..., n$), denominado como a saída da camada g. No caso dessa camada ser do tipo escondida, y_g será a entrada x da camada seguinte e, no caso de ser a camada de saída, será a própria saída da rede.

O processo definido pelas equações (2.1) e (2.2) é iniciado a partir das entradas da rede até a camada de saída, gerando a saída final da rede e é denominado *feedforward* ou propagação direta. Por meio desse procedimento, é possível avaliar a saída gerada por uma rede de pesos pré-determinados. Entretanto, como discutido anteriormente, é necessário um mecanismo de alteração dos valores w_{ki} para que a rede "aprenda" como solucionar um problema de forma correta. O processo de aprendizado, bem como os mecanismos necessários para executá-lo, serão apresentados a seguir.

2.1.2 Aprendizado

O processo de aprendizado de uma rede neural pode ser realizado por meio de três paradigmas distintos: supervisionado, por reforço e não supervisionado [19]. Esses métodos podem ser definidos resumidamente por:

- Supervisionado: a rede é alimentada com pares formados por uma entrada e a respectiva resposta desejada para aquela entrada. Essa resposta desejada, também denominada por *target*, representa a ação ótima a ser realizada pela rede dada aquela entrada específica. A partir da entrada, a rede sofre o processo de *feedforward* explicado anteriormente, gerando a saída real da rede, que é comparada com a saída desejada. Essa comparação gera um erro (simples subtração entre as duas saídas) que irá ser alimentado de novo na rede de forma a ajustar seus pesos com objetivo de transferir o conhecimento relativo ao par entrada-*target* para a rede. Essa realimentação do erro ocorre de acordo com um algoritmo de aprendizado bem definido. A alternância entre os processos de *feedforward* e de realimentação do erro é realizada iterativamente até que a rede seja considerada capaz de fornecer as respostas corretas para as entradas que recebe.

- Por reforço: o aprendizado do mapeamento de entradas e saídas é realizado por meio da interação com o ambiente. O objetivo desse tipo de aprendizado é minimizar o custo esperado de ações tomadas ao longo do tempo, em vez de minimizar o custo imediato relativo à tomada de uma ação em um dado instante de tempo. Em várias situações, uma ação realizada anteriormente em uma dada sequência de iterações é determinante para o comportamento geral do sistema ao longo do tempo. Dessa forma, o objetivo de um sistema baseado no aprendizado por reforço é determinar essas ações e realimentá-las para o ambiente.

- Não supervisionado (ou auto-organizado): uma medida de qualidade da representação que o sistema deve atingir é previamente definida de forma independente à tarefa a ser executada. O sistema que usa esse tipo de aprendizado, então, procura otimizar essa medida de

qualidade, alterando suas variáveis (pesos, no caso das redes neurais). O sistema possui informações sobre o ambiente, mas não é alimentado com a resposta desejada. Muitas vezes, é selecionado o aprendizado não supervisionado justamente por não haver respostas desejadas pré-determinadas ou uma medida de recompensa que varia com as ações tomadas.

No trabalho apresentado, o treino das redes neurais consideradas adota o paradigma supervisionado e o algoritmo de aprendizado é definido como sendo o *backpropagation*, que será explicado na próxima subseção.

2.1.3 *Backpropagation*

A partir do processo de *feedforward* descrito anteriormente como sendo a aplicação camada à camada das equações (2.1) e (2.2), é gerado um sinal de erro para cada neurônio presente na camada de saída. O sinal de erro do neurônio de saída j na n-ésima iteração é matematicamente definido como sendo

$$e_j(n) = d_j(n) - y_j(n), \tag{2.9}$$

em que $d_j(n)$ representa a resposta desejada do neurônio de saída j na n-ésima iteração e $y_j(n)$ indica a saída real desse mesmo neurônio na mesma iteração. A partir da Equação (2.9), pode-se definir também a energia instantânea do erro do neurônio j por

$$\mathcal{E}_j(n) = \frac{1}{2} e_j^2(n). \tag{2.10}$$

Pode-se, ainda, definir a energia do erro total instantânea de toda a rede como sendo

$$\mathcal{E}(n) = \sum_{j \in C} \mathcal{E}_j(n), \tag{2.11}$$

em que o conjunto C representa todos os neurônios da camada de saída. Por fim, considerando que uma amostra de treino consista em M exemplos, a energia média do erro sobre toda a amostra de treino é definida por

$$\mathcal{E}_{av}(n) = \frac{1}{M} \sum_{n=1}^{M} \mathcal{E}_j(n). \tag{2.12}$$

A partir do erro calculado para cada neurônio de saída, é possível determinar a correção dos pesos sinápticos da rede $\Delta w_{ji}(n)$ que conectam quaisquer neurônios i e j. Essa correção é computada por operações matemáticas executadas desde a camada de saída até a primeira camada da rede em um processo definido por *backpropagation*. Existem diversas variações do algoritmo de *backpropagation*, como Levenberg–Marquardt [24] e BFGS Quasi-Newton [25]. Neste trabalho, redes treinadas por meio da variação Levenberg–Marquardt do *backpropagation* foram utilizadas. Esse algoritmo de treino realiza uma adaptação dos pesos de acordo com

$$\Delta w_{ji}(n) = -(J_n^T J_n + \mu I)^{-1} J_n \epsilon_n, \tag{2.13}$$

em que J_n é a matriz Jacobiana avaliada na n-ésima iteração, μ é o coeficiente combinacional, I é a matriz identidade e ϵ_n, o vetor de erro.

Além do Levenberg–Marquardt, foram estudados e implementados os algoritmos de *backpropagation* com gradiente descendente [2] e de *backpropagation* com momento [26], que serão explicados com detalhes a seguir.

O algoritmo de *backpropagation* com gradiente descendente determina que a correção de cada peso da rede $\Delta w_{ji}(n)$ seja calculada por

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}, \tag{2.14}$$

em que η representa o coeficiente de aprendizado da rede e a derivada parcial $\partial \mathcal{E}(n)/\partial w_{ji}(n)$, também chamada de gradiente, pode ser expressa de acordo com a regra da cadeia como sendo

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}. \tag{2.15}$$

Diferenciando ambos os lados de (2.10) com respeito a $e_j(n)$, tem-se que

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n). \tag{2.16}$$

A diferenciação de ambos os lados de (2.9) com respeito a $y_j(n)$ resulta em

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1. \tag{2.17}$$

Diferenciando ambos os lados de (2.2) aplicado ao neurônio j da camada de saída com respeito a $v_j(n)$, tem-se

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)). \tag{2.18}$$

Por fim, ao se diferenciar a Equação (2.1) com respeito a $w_{ji}(n)$, tem-se que

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n). \tag{2.19}$$

Assim, considerando os cálculos das derivadas parciais dados pelas equações (2.16) a (2.19) e as definições dadas pelas equações (2.14) e (2.15), tem-se que a correção de pesos é calculada por

$$\Delta w_{ji}(n) = \eta \; e_j(n) \varphi_j'(v_j(n)) y_i(n). \tag{2.20}$$

Entretanto, de forma mais geral, define-se a chamada regra delta para o cálculo da correção dos pesos da rede como sendo

$$\Delta w_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n), \tag{2.21}$$

em que o cálculo do gradiente local $\delta_j(n)$ varia de acordo com a camada em que o neurônio j está. Ou seja, caso o neurônio j seja um neurônio da camada de saída, $\delta_j(n)$ é definido como visto anteriormente pela Equação (2.20), ou seja

$$\delta_j(n) = \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n)). \tag{2.22}$$

Porém, caso j seja um neurônio da camada escondida, $\delta_j(n)$ é calculado por

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}\varphi'_j(v_j(n)) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n), \tag{2.23}$$

em que k indica todos os neurônios da camada posterior à camada escondida em que o neurônio j está posicionado. Assim, o peso indicado por $w_{kj}(n)$ é o peso da conexão que parte do neurônio j até o neurônio k na n-ésima iteração.

Ao se comparar as Equações (2.13) e (2.21), percebe-se que a atualização de pesos executada pela variação de gradiente descendente do *backpropagation* é computacionalmente mais simples que a realizada pela variação Levenberg–Marquardt, uma vez que, para aplicar o cálculo definido em (2.13), é necessário computar valores de matrizes Jacobianas e inverter resultados de multiplicações matriciais enquanto que o cálculo expresso por (2.21) exige o cálculo de derivadas que podem ser reduzidas a simples somas e multiplicações.

À Equação (2.21), pode ser adicionado um termo de momento, resultando na chamada regra delta generalizada

$$\Delta w_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n) + \alpha \Delta w_{ji}(n-1). \tag{2.24}$$

A Equação (2.24) representa a variante com momento do algoritmo de *backpropagation* com gradiente descendente e seus termos α e $\Delta w_{ji}(n-1)$ são, respectivamente, a constante de momento e a correção do peso que conecta o neurônio i ao neurônio j calculada na iteração anterior.

A função do termo de momento é possibilitar o aumento do coeficiente de aprendizado η sem que a rede se torne instável, ou seja, oscilatória. Ambos η e α variam entre 0 e 1 e indicam, respectivamente, o grau de mudança nos pesos da rede entre uma iteração e outra e o quanto deve ser considerado dos pesos da iteração anterior no cálculo da correção dos pesos.

Por fim, a aplicação da Equação (2.21), no caso do *backpropagation* com gradiente descendente, ou da Equação (2.24), no caso do *backpropagation* com momento, pode ser executada de acordo com dois métodos de atualização de pesos utilizados durante o treino de uma rede neural: o *batch* e o *online* [26].

No método *batch*, a modificação dos pesos da rede é realizada após a apresentação de todos os M exemplos que compõem uma época de treino. Ou seja, a função de custo para um aprendizado *batch* é definida pela média do erro, $\mathcal{E}_{av}$, e os ajustes são realizados época a época. As vantagens da utilização desse método incluem a paralelização do processo de aprendizado e a estimação mais exata do vetor de gradiente. Já a desvantagem é a necessidade de armazenamento, referente aos M conjuntos de cálculos realizados sobre os exemplos de treino de uma época.

Já o método *online* consiste no ajuste de pesos sinápticos exemplo a exemplo, ou seja, a função de custo considerada por esse método é o erro total instantâneo $\mathcal{E}(n)$. Dessa forma, a cada par de entrada $x_k(n)$ e resposta desejada $d_k(n)$ apresentado à rede, a adaptação descrita pela Equação (2.21) é executada considerando o erro calculado naquele instante para o par considerado. O método *online* é o mais adotado para solucionar problemas de classificação de padrões, uma vez que é simples de ser implementado e proporciona soluções eficientes para problemas complexos e de larga-escala nessa área.

2.1.4 Redes Neurais Artificiais Recorrentes

Existem diversos tipos de redes neurais artificiais. O modelo até então apresentado e, como visto pela Figura 2.1, é o modelo de rede *feedforward* com múltiplas camadas, que é considerado como sendo a estrutura mais comum de RNAs.

Entretanto, existem também redes neurais recorrentes (RNRs), que se distinguem das redes *feedforward* pelo fato de apresentarem pelo menos um laço de *feedback* em suas estruturas. Devido à presença desses laços, as redes neurais recorrentes são ferramentas poderosas para representar e avaliar informações sequenciais. Na verdade, essas estruturas são bastante utilizadas em pesquisas de reconhecimento de caracteres manuscritos [27], reconhecimento de fala [28] e, inclusive, geração de texto [29].

A classificação de uma rede neural recorrente depende diretamente da natureza dos laços de *feedback* que essa rede apresenta. Um dos tipos mais simples de RNRs são as redes do tipo Elman, cujo diagrama é mostrado pela Figura 2.4.

A rede recorrente Elman foi proposta pelo pesquisador de mesmo nome em 1990 [30] para representar sequências bem definidas de caracteres cuja ordem é um fator fundamental. Com essa representação, era esperado que a solução problemas de relevantes ao processamento da linguagem natural se tornasse mais viável. Para tanto, Elman propôs uma rede com memória interna, representada pela chamada camada de contexto, que armazena a saída da camada escondida na iteração anterior. Os pesos que partem da camada escondida para a camada de contexto são todos unitários, enquanto que os pesos que partem da camada de contexto para a camada escondida, bem como os pesos que partem da camada escondida para a de saída, são calculados de acordo com a regra do *backpropagation*. Além disso, por definição, a função de ativação dos neurônios da camada de contexto é linear. No diagrama da rede Elman da Figura 2.4, o símbolo z^{-1} envolvido por um quadrado representa o atraso unitário, isto é, referente a iteração anterior.

A rede do tipo Elman foi inspirada no trabalho de Jordan realizado quatro anos antes [31], que

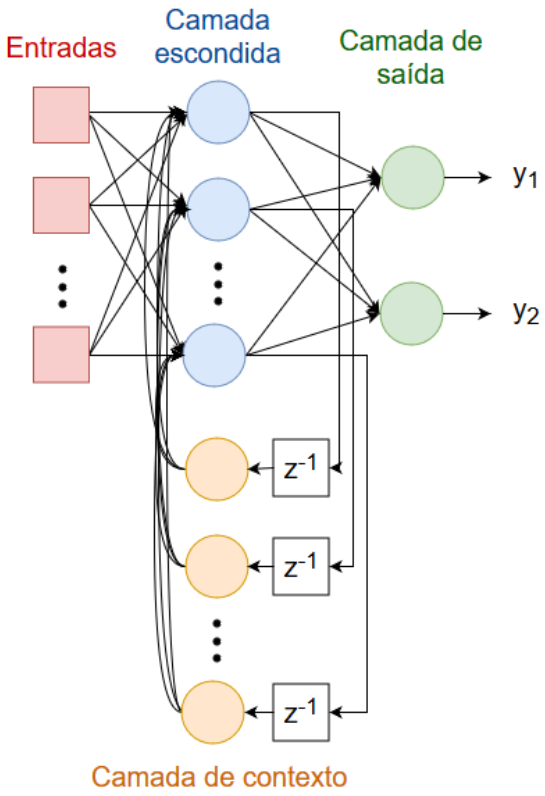

Figura 2.4: Diagrama de rede recorrente do tipo Elman

definiu uma rede com conexões recorrentes formadas por unidades de contexto que são alimentadas pelos neurônios de saída e que alimentam a camada escondida e a própria camada de contexto. Essa topologia de redes recorrentes é exemplificada pelo diagrama da Figura 2.5.

Além dessas RNRs supracitadas, existem, também, as redes do tipo NARX (*nonlinear autoregressive network with exogenous inputs*) [32], cuja estrutura pode ser exemplificada pela Figura 2.6.

Na Figura 2.6, os símbolos $z^{-(1..n)}$ e $z^{-(1..m)}$ envolvidos por quadrados representam atrasos de 1 a n e de 1 a m respectivamente. Ou seja, a saída de redes do tipo NARX depende dos n valores anteriores na saída da rede e dos m valores anteriores em sua entrada, como indicado, de forma genérica, pela Equação (2.25).

$$y(t) = f(y(t-1), y(t-2), ..., y(t-n), x(t-1), x(t-2), ..., x(t-m)). \tag{2.25}$$

Além das inúmeras aplicações das redes NARX para prever valores futuros em uma sequência temporal, para realizar filtragens não lineares e para modelar sistemas dinâmicos não-lineares, essas redes podem ser utilizadas em duas configurações distintas: paralela e em série-paralelo. No caso desta última configuração, a arquitetura da rede é considerada puramente *feedforward*, o que permite a utilização da versão estática do algoritmo de *backpropagation*.

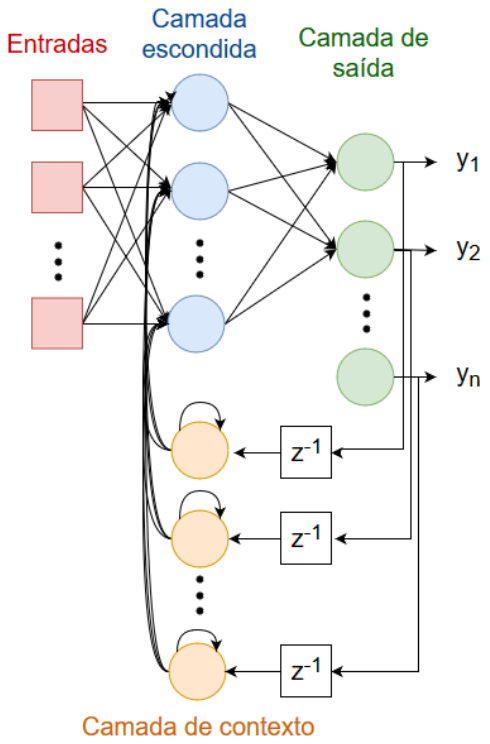

Figura 2.5: Diagrama de rede recorrente do tipo Jordan

2.2 Transformada de Fourier

A transformada de Fourier é uma ferramenta matemática que permite representar sinais não-periódicos e de tempo contínuo como componentes de frequência. Em muitas situações, um sinal aparentemente complexo no domínio do tempo é bastante simples no domínio da frequência. Portanto, a transformada de Fourier pode facilitar a análise de sinais, sendo frequentemente aplicada na área de processamento de sinais de natureza diversa [33] [34].

A definição da transformada $f(\omega_t)$ de uma função $f(t)$ é dada por

$$F(\omega) = \mathcal{F}\{f(t)\} = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt, \tag{2.26}$$

em que j é a unidade imaginária, t representa valores no domínio do tempo e ω, no domínio da frequência.

Aplicando a fórmula de Euler [35] sobre (2.26), pode-se notar que

$$F(\omega) = \mathcal{F}\{f(t)\} = \int_{-\infty}^{\infty} f(t)cos(\omega t)dt - j\int_{-\infty}^{\infty} f(t)sen(\omega t)dt. \tag{2.27}$$

No caso de problemas que envolvam sinais não-contínuos em sistemas digitais, a transformada de Fourier deve ser utilizada em sua forma discreta, que será detalhada a seguir.

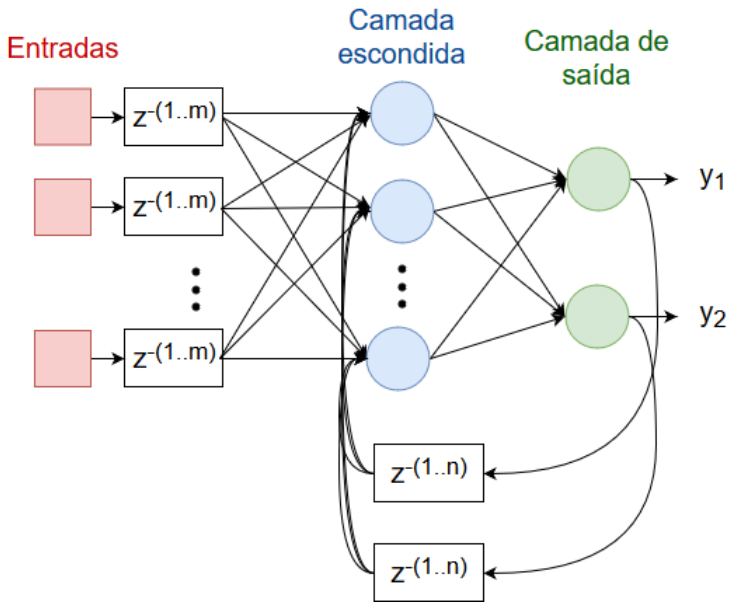

Figura 2.6: Diagrama de rede recorrente do tipo NARX

2.3 Transformada de Fourier Discreta (DFT)

Dada uma sequência $x[n]$ discreta no tempo, em que $n = 0, 1, ..., N-1$, a transformada de Fourier discreta de $x[n]$, $X[k]$, é definida por

$$X[k] = \sum_{n=0}^{N-1} x[n]\ e^{-j\frac{2\pi k}{N}n}, \tag{2.28}$$

em que j é a unidade imaginária.

De forma similar ao caso contínuo, a definição de DFT pode ser reescrita como sendo uma subtração entre senos e cossenos da seguinte forma

$$X[k] = \sum_{n=0}^{N-1} x[n]\left[cos\left(\frac{2\pi kn}{N}\right) - jsen\left(\frac{2\pi kn}{N}\right)\right]. \tag{2.29}$$

2.3.1 Transformada rápida de Fourier (FFT)

A transformada rápida de Fourier desenvolvida em 1965 por Cooley e Tukey [36] é um algoritmo numericamente eficiente para calcular a transformada discreta de Fourier definida anteriormente pela Equação (2.28). Em seu trabalho, Cooley e Tukey demonstram que a complexidade do cálculo de séries de Fourier pode ser reduzida de $O(N^2)$ para $O(NlogN)$, em que N é a quantidade de elementos na série de Fourier.

Uma vez que o princípio do algoritmo desenvolvido por Cooley e Tukey é computar uma DFT por meio de DFTs menores, é possível combinar o algoritmo com outras técnicas de cálculo de FFT, tais como o algoritmo de Bluestein [37] ou o de Rader [38].

A forma mais comum de cálculo da FFT de Cooley e Tukey é a decimação no tempo (DIT) *radix*-2, cujo nome indica a decomposição de uma DFT de tamanho N em duas DFTs intercaladas de tamanho $N/2$. Esse tipo de FFT tem como princípio o cálculo da DFT das entradas $x[n]$ com índice par, isto é, $x[2m] = \{x[0], x[2], \ldots, x[N-2]\}$, e da DFT das entradas com índice ímpar, $x[2m+1] = \{x[1], x[3], \ldots, x[N-1]\}$, separadamente e da posterior combinação dos dois resultados para obter a DFT $X[k]$ da sequência completa. Essa FFT assume que N é uma potência de 2 por ser mais eficiente do ponto de vista computacional [36].

Dessa forma, o cálculo da DFT indicado pela Equação (2.28) é executado da seguinte forma

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\ e^{-j\frac{2\pi k}{N}(2m)} + \sum_{m=0}^{N/2-1} x[2m+1]\ e^{-j\frac{2\pi k}{N}(2m+1)}. \tag{2.30}$$

Pode-se, ainda, definir

$$W_N = e^{-j\frac{2\pi}{N}} \tag{2.31}$$

e reescrever a Equação 2.30 como

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\ W_N^{k(2m)} + \sum_{m=0}^{N/2-1} x[2m+1]\ W_N^{k(2m+1)}, \tag{2.32}$$

que pode ser reorganizada na seguinte forma

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\ (W_N^2)^{km} + W_N^k \sum_{m=0}^{N/2-1} x[2m+1]\ (W_N^2)^{km}. \tag{2.33}$$

Entretanto, ao se analisar a Equação (2.31), percebe-se que $W_N^2 = W_{N/2}$. Assim, é possível reescrever (2.33) como

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\ (W_{N/2})^{\,km} + W_N^k \sum_{m=0}^{N/2-1} x[2m+1]\ (W_{N/2})^{\,km}. \tag{2.34}$$

Ao se comparar os termos da Equação (2.34) com a definição expressa por (2.28), nota-se que o primeiro somatório de (2.34) é uma DFT de $N/2$ pontos das amostras de índice par da sequência $x[n]$ e o segundo somatório, uma DFT de mesmo tamanho calculada sobre as amostras de índice ímpar da sequência $x[n]$. Denominando a primeira DFT por $X_p[d]$ e a segunda por $X_i[d]$, pode-se descrever $X[k]$ por

$$X[k] = X_p[d] + W_N^k \cdot X_i[d]. \tag{2.35}$$

A Equação (2.35) representa a *butterfly* para a FFT com DIT *radix*-2 e pode ser descrita graficamente pela Figura 2.7.

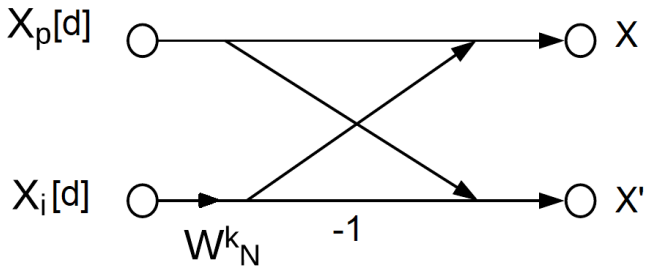

Figura 2.7: Diagrama de uma *butterfly* da FFT com DIT *radix*-2

Para uma DFT de tamanho $N = r \cdot m$, em que r é o valor do *radix*, o algoritmo de FFT separa a DFT em transformadas menores, de tamanho $m = N/r$ e combina seus resultados por meio de *butterflies* de tamanho r. Portanto, uma FFT com DIT *radix*-2 de 8 pontos pode ser computada pelo cálculo de duas DFTs de 4 pontos associadas a uma composição de *butterflies*, como descrito pelo diagrama da Figura 2.8.

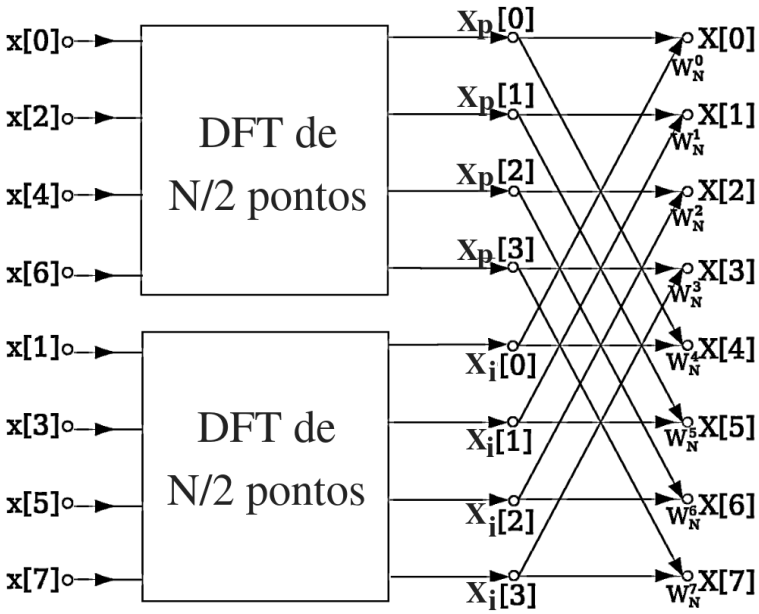

Figura 2.8: Diagrama do cálculo de uma FFT com DIT *radix*-2 [4]

Percebe-se que a FFT é um típico exemplo de algoritmo de divisão e conquista, uma vez que o problema da DFT é dividido em problemas menores (em DFTs menores), que são solucionados e cujas soluções são combinadas até que se obtenha a solução do problema principal inicial.

A partir do algoritmo de Cooley e Tukey [36], algumas variantes mais eficientes da FFT foram desenvolvidas, tais como a FFT com DIT *radix*-4 e a FFT com *split-radix*, que promete utilizar apenas metade das multiplicações requisitadas pela FFT *radix*-2 [39].

2.4 Representações numéricas

A forma em que um computador representa dados é de fundamental importância e pode inclusive determinar alguns aspectos da organização de computadores [40].

A representação de dados em um computador é feita na base binária, cuja unidade fundamental de informação é denominada por *bit*, um acrônimo da expressão inglesa *binary digit*. Neste trabalho, números escritos na base binária serão representados por uma sequência de dígitos 0 ou 1 seguida pelo subscrito 2. Já os números na base decimal serão sequências de dígitos que variam de 0 a 9 que podem estar indicadas pelo subscrito 10 ou não.

Da mesma forma que a base decimal, a base binária é um sistema de numeração posicional. Assim, é atribuído um peso a cada algarismo que compõe um número, sendo que esse peso corresponde a uma potência da base do sistema numérico de acordo com a posição do algarismo. Assim, o número 257_{10} na base decimal (ou base 10) pode ser escrito da seguinte forma

$$257_{10} = 2 \cdot 10^2 + 5 \cdot 10^1 + 7 \cdot 10^0 \tag{2.36}$$

e o número 10101001_2 na base binária (ou base 2) pode ser escrito como sendo

$$10101001_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 169_{10}. \tag{2.37}$$

Vale ressaltar que o raciocínio da Equação (2.37) pode ser extrapolado para situações em que outros números inteiros e positivos são representados na base binária. Para representar, na base binária, números fracionários, deve-se utilizar representações em ponto fixo ou em ponto flutuante e, para representar números negativos, é comum o uso da representação em complemento de dois.

2.4.1 Ponto fixo

A representação em base binária de números fracionários em ponto fixo também é análoga à representação em base decimal, ou seja, é definida uma vírgula que delimita os dígitos que são multiplicados por potências negativas da base e aqueles que são multiplicados por potências positivas [40]. Assim, da mesma forma que o número $1{,}367_{10}$ pode ser escrito como sendo

$$1{,}367_{10} = 1 \cdot 10^0 + 3 \cdot 10^{-1} + 6 \cdot 10^{-2} + 7 \cdot 10^{-3}, \tag{2.38}$$

o número $1011{,}1101_2$ pode ser expresso por

$$1011{,}1101_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = 11{,}8125_{10}. \tag{2.39}$$

Percebe-se que a posição da vírgula, tanto binária quanto decimal, define como o número deve ser compreendido [41]. No caso dos números binários, associa-se a notação Q à representação

em ponto fixo para indicar a quantidade de *bits* após a vírgula [42]. Ou seja, uma notação Q14 informa que existem 14 *bits* após a vírgula.

2.4.2 Complemento de dois

As representações por complemento são particularmente úteis para representar números negativos na base binária. A vantagem desse tipo de representação em relação a uma do tipo sinal-e-magnitude (em que o primeiro bit indica a presença ou a ausência do sinal negativo) consiste no fato de não ser necessário o processamento do bit de sinal separadamente e, ainda, de ser possível identificar o sinal do número por meio de uma análise rápida do bit mais significativo.

A primeira representação em complemento foi denominada por complemento de dez e é aplicada para números descritos na base decimal. O princípio dessa forma de complemento baseia-se no fato de que uma subtração $a - b$ entre números decimais a e b pode ocorrer por meio da adição de três termos:

1. O minuendo a;
2. O resultado da diferença entre um número formado apenas por dígitos 9 e o subtraendo b;
3. O número 1.

Assim, a diferença $167 - 52$ pode ser definida pela soma entre 167, o resultado da subtração $999 - 52$ e o número 1. Ou seja, se

$$167 + (999 - 52) = 167 + 947 = (1)114, \tag{2.40}$$

em que o algarismo 1 entre parênteses indica o *carry* da soma e é desprezado.

$$167 - 52 = 114 + 1 = 115. \tag{2.41}$$

O complemento de dois de um número N que contenha d dígitos na base r é definido como sendo $r^d - N$ para N não nulo e 0 para N nulo. Assim, o complemento de 2 de 0011_2 é dado por $2^4 - 0011_2 = 10000_2 - 0011_2 = 1101_2$. Outra forma de calcular o complemento de dois de um número binário é somar 1_2 ao complemento de 1 do número. Ou seja, é inverter os *bits* do número e somar 1_2.

Na representação em complemento de dois, os pesos de cada dígito são atribuídos da mesma forma explicada anteriormente, porém o *bit* mais significativo tem valor negativo e, os demais, positivo. Portanto, assumindo-se uma representação em complemento de dois, o número 10101001_2, convertido para a base decimal é

$$10101001_2 = -1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = -87_{10} \tag{2.42}$$

e não 169_{10} como expresso na Equação (2.37), que assume que o número é positivo.

2.5 *Hardware* utilizado

Nesta seção, são apresentados os dois equipamentos utilizados neste projeto: o rádio definido por *software* USRP N210, cuja função neste trabalho foi a aquisição dos sinais do espectro eletromagnético, e o FPGA Cyclone IV, no qual foi definido todo o sistema desenvolvido em linguagem Verilog.

2.5.1 Rádios definidos por *software*

O padrão IEEE 1900.1 [43] estipula que rádios definidos por *software* (RDS) são aqueles em que todas ou algumas das funções de sua camada física são implementadas em *software*. Portanto, esses dispositivos são bastante versáteis, permitem a execução mais rápida de testes e reduzem os custos de desenvolvimento de produtos. Além disso, a definição de algumas funções em *software* permite o reuso de parte de código, o que pode ser bastante interessante em um projeto que envolva os RDSs. Essa flexibilidade tornou os rádios definidos por *software* uma opção bastante popular nos meios acadêmicos, militar e até nas pesquisas e explorações espaciais realizadas pela NASA [44].

O rádio utilizado neste trabalho é o USRP (*Universal Software Radio Peripheral*) N210 [45] desenvolvido pela Ettus Research e representado na Figura 2.9.

Figura 2.9: Rádio definido por *software* USRP N210

Esse RDS inclui um conversor analógico-digital de 100 MS/s, um conversor digital-analógico de 400 MS/s e um módulo de conexão Gigabit Ethernet. Este modelo de rádio possui um conversor analógico-digital capaz de capturar sinais com uma frequência de amostragem de até 400MHz, com uma resolução de 16 *bits* por amostra. Ele ainda possui uma entrada que permite que múltiplos rádios da série N210 se sincronizem e utilizem uma configuração MIMO (*Multiple Input Multiple Output*). Um módulo opcional GPSDO pode ainda ser usado para sincronizar o *clock* de referência do rádio para que seja 0.01 ppm do padrão de GPS utilizado no mundo. O USRP N210 consegue transmitir até 50 MS/s de e para aplicações.

O USRP N210 também inclui um FPGA do tipo Xilinx® Spartan® 3A-DSP 3400, cujo *firmware* pode ser recarregado por meio de uma interface Gigabit Ethernet. Esse FPGA possui o potencial

para processar até 100 MS/s em ambas as direções de transmissão e recepção do rádio. Usuários podem implementar funções customizadas no FPGA ou no microprocessador RISC de 32 *bits* presente no USRP N210.

O USRP N210 é bastante utilizado em aplicações que envolvem o sensoriamento do espectro eletromagnético, o acesso dinâmico ao espectro, o desenvolvimento de rádios cognitivos ou a prototipagem da camada física.

As funcionalidades do USRP N210 podem ser definidas a partir dos *softwares*: GNU Radio, LabVIEW e MATLAB de forma rápida e prática por meio de diagramas de blocos.

Apesar do USRP N210 incorporar o FPGA Xilinx® Spartan® 3A-DSP 3400, o sistema desenvolvido neste trabalho não foi definido nesse FPGA devido ao fato dele conter apenas 53.712 elementos lógicos [46]. Assim, neste trabalho, foi utilizado o FPGA apresentado na próxima seção.

2.5.2 *Field Programmable Gate Array* (FPGA)

Field Programmable Gate Arrays são circuitos integrados que podem ser configurados após a sua manufatura por seus usuários. Esses dispositivos semicondutores são formados por blocos de lógica reconfigurável cujas conexões são definidas a nível de programação. Para que a configuração dos FPGAs ocorra, necessita-se que essas conexões sejam definidas por projetos escritos em linguagem de descrição de *hardware* como, por exemplo, VHDL ou Verilog ou por meio de circuitos esquemáticos.

FPGAs são majoritariamente manufaturados por empresas como Altera [47], adquirida pela Intel em 2015 [48], e Xilinx [49], que, juntas, controlavam 89% do mercado de FPGAs em 2016 [50]. Esse mercado sofreu um crescimento de 6% em 2017 e os FPGAs são dispositivos cada vez mais comuns nos mundos acadêmico, industrial e comercial. Provavelmente, a popularização dos FPGAs se justifica por sua versatilidade e flexibilidade, que permitem a redução de custos nas fases de execução de testes e prototipagem. Além disso, as aplicações dos FPGAs são inúmeras [51] e variam desde o diagnóstico de doenças na área médica à solução de problemas no campo aeroespacial.

Neste trabalho, o FPGA utilizado possui 115 mil elementos lógicos e encontra-se incorporado ao kit de desenvolvimento DE2-115 [52] fabricado pela Terasic. Além de conter o FPGA Altera® Cyclone® IV EP4CE115F29C7N, esse kit integra vários dispositivos de entrada e de saída que são particularmente úteis nas etapas de desenvolvimento de sistemas, análise de resultados e solução de erros. A Figura 2.10 mostra uma foto do kit de desenvolvimento com seus elementos enumerados.

Os componentes destacados na Figura 2.10 são:

1. Conector de energia de 12V;
2. Botão Liga/Desliga;
3. *Chipset* controlador Altera USB Blaster;

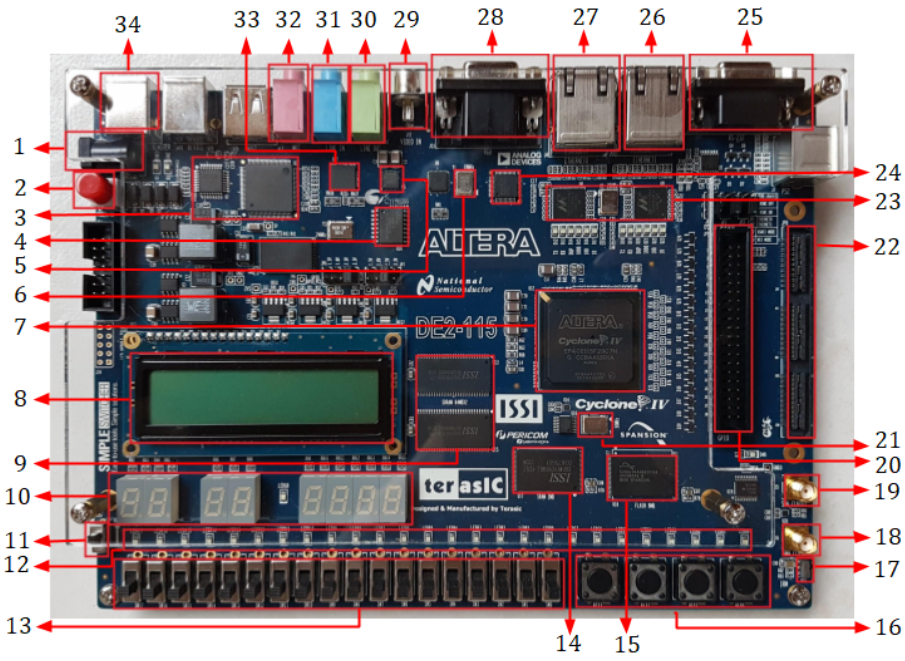

Figura 2.10: Kit de desenvolvimento Altera® DE2-115 com seus componentes enumerados

4. Dispositivo de configuração da Altera;
5. Áudio CODED;
6. Oscilador de 28MHz;
7. Altera Cyclone IV EP4CE115F29C7N com 115 mil elementos lógicos;
8. Módulo de LCD 16x2;
9. Duas memórias SDRAM de 64MB;
10. *Display* de 7 segmentos;
11. Chave de modo programação;
12. 18 LEDs vermelhos e 8 LEDs verdes;
13. 18 chaves de seleção;
14. Memória SRAM de 2MB;
15. Memória Flash de 8MB;
16. 4 botões de pressão;
17. Receptor de infra-vermelho (IR);
18. Extensão SMA *Clock In*;
19. Extensão SMA *Clock Out*;

20. Conector de expansão;
21. Oscilador de 50MHz;
22. Conector HSMC;
23. Ethernet Gigabit PHY;
24. VGA DAC de 8 *bits*;
25. Porta RS-232;
26. Porta Ethernet 1;
27. Porta Ethernet 0;
28. Saída VGA;
29. Entrada de vídeo;
30. *Line Out*;
31. *Line In*;
32. Entrada de som;
33. Controlador do USB Host/Slave;
34. Porta USB Blaster.

Neste trabalho, além do próprio *chip* do FPGA, enumerado por 7, outros componentes incorporados ao kit de desenvolvimento foram utilizados. As estruturas indicadas por 1,2, 3 e 34 são essenciais para fornecer energia elétrica ao kit e para carregar descrições de *hardware* no FPGA. Já 10, 12, 13 e 16 são utilizados para observar estados do sistema e valores calculados durante processos intermediários. O oscilador de 50MHz indicado por 21 define o *clock* que sincroniza o funcionamento do sistema.

Este capítulo apresentou conceitos e ferramentas essenciais à execução do trabalho proposto. A combinação de todos os conceitos descritos para elaborar de um sistema de identificação de oportunidades de transmissão em redes sem fio é exposta no capítulo seguinte.

Capítulo 3

Metodologia

Este capítulo descreve a metodologia proposta para o desenvolvimento de um sistema, baseado em redes neurais, capaz de prever oportunidades de transmissão em redes sem fio. As etapas metodológicas adotadas para o desenvolvimento do sistema são expostas na Figura 3.1.

Figura 3.1: Etapas da metodologia para desenvolvimento do sistema

Na primeira etapa, os sinais de fase e quadratura são coletados a partir do RDS. Na segunda etapa, esses sinais são processados para gerar um sinal de energia do espectro eletromagnético. Na terceira etapa, o sinal de energia gerado é quantizado. Na quarta etapa, o sinal quantizado é analisado pela rede neural e, na quinta etapa, a saída da rede neural é observada. As etapas da Figura 3.1 foram implementadas tanto em *hardware* (FPGA) quanto em *software* (Matlab e C++).

Além da aplicação dessas etapas ao longo do desenvolvimento do sistema, para que sua execução esteja sincronizada em *hardware*, é necessário descrever um controlador em máquina de estados finitos.

A Seção 3.1 deste capítulo apresenta uma descrição do sistema proposto. As secções 3.2, 3.3, 3.4 e 3.5 referem-se às etapas de aquisição, processamento de sinal, quantização e rede neural da Figura 3.1 respectivamente. Já a Seção 3.6 discorre sobre o controlador elaborado no sistema em *hardware* e a Seção 3.7, sobre a metodologia adotada nos estudos da adaptabilidade da rede neural.

3.1 Descrição do sistema

O sistema proposto por Fernandes *et al.* em [1] foi implementado e expandido, resultando na metodologia geral resumida pela Figura 3.2.

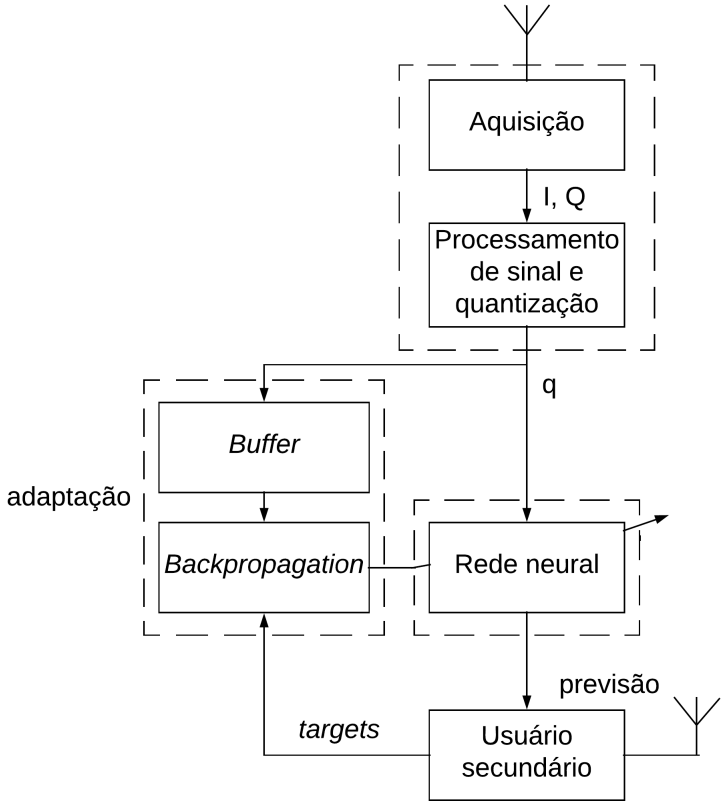

Figura 3.2: Diagrama da metodologia adotada

Na Figura 3.2, a seta que atravessa o bloco indicado por "Rede neural" representa a adaptabilidade dessa estrutura de inteligência artificial.

O sistema proposto utiliza um rádio definido por *software* para realizar a aquisição de sinais de fase e de quadratura I e Q, que são posteriormente processados e quantizados para gerar o sinal q. Este sinal é composto por apenas 0s e 1s, indicando, respectivamente, a ausência e a presença de energia em uma janela temporal.

O sinal q é utilizado para preencher um *buffer* de tamanho definido e serve também de entrada para a rede neural, que realiza a previsão de oportunidade. O usuário secundário utiliza a previsão da rede neural para determinar se ele irá transmitir ou não em dado momento. A todo instante, o usuário secundário monitora o espectro eletromagnético e gera respostas desejadas (*targets*) para um período de monitoramento correspondente ao tamanho do *buffer*. Essas respostas desejadas devem considerar erros cometidos pela rede que podem ter provocado colisões no espectro eletromagnético, bem como atitudes excessivamente conservadoras que podem deixar o canal ocioso por um tempo desnecessário. A partir dos pares formados pelo *buffer* e pelos *targets*, a RNA pode passar por um processo de adaptação definido pelo algoritmo do *backpropagation*.

Para viabilizar o desenvolvimento do sistema definido pelo diagrama da Figura 3.2, três procedimentos principais foram executados neste trabalho:

1. Desenvolvimento das funções referentes ao processamento de sinal, à quantização e à rede neural mostradas na Figura 3.2 em *hardware*;
2. Desenvolvimento das funções referentes ao processamento de sinal, à quantização e à rede neural mostradas na Figura 3.2 em *software* para comparar seus resultados com os obtidos em 1;
3. Produção de projetos em *software* para avaliar e testar algoritmos que permitam a adaptabilidade da rede neural.

Os processos que definem os blocos de aquisição, de processamento de sinal, de quantização e de rede neural do diagrama da Figura 3.2 são detalhados de acordo com a Figura 3.3. Esses processos foram definidos em *hardware* para executar o procedimento 1 e, assim, possibilitar a previsão de oportunidades efetuada por uma rede neural descrita em FPGA à medida em que os sinais $I(t)$ e $Q(t)$ são apresentados na entrada do sistema.

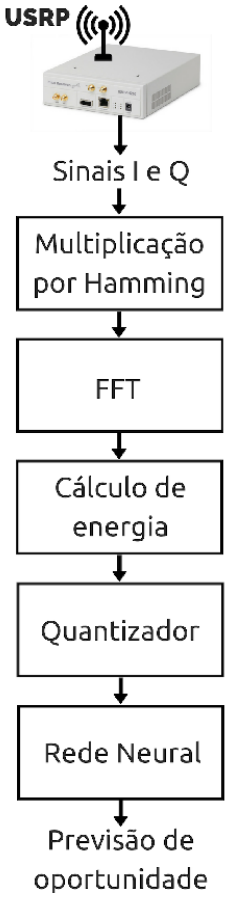

Figura 3.3: Detalhamento do sistema desenvolvido em *hardware*

As etapas que compõem a metodologia para o desenvolvimento do sistema em *hardware* descrito pela Figura 3.3 são expostas na Figura 3.1. Essas etapas também foram aplicadas no desenvol-

vimento do mesmo sistema em *software* para permitir a comparação com a implementação em *hardware*, como objetivado pelo procedimento 2.

3.2 Aquisição

A aquisição foi executada pelo rádio USRP N210 apresentado na Seção 2.5.1 em redes no padrão 802.11g em modo *Ad-hoc*.

O IEEE 802.11 atua na camada física e define uma série de padrões de transmissão e codificação para comunicações sem fio WLAN (*Wireless Local Area Network*). A classe de padrões IEEE 802.11 se tornou tão popular que passou a ser utilizada na maioria das transmissões em WLANs e ficou conhecida por WiFi [53].

Existem vários padrões para a tecnologia LAN sem fio, tais como 802.11a, 802.11b e 802.11g, que, apesar de terem suas peculiaridades, compartilham algumas características em comum. Por exemplo, esses padrões utilizam o protocolo de acesso CSMA/CA e podem operar tanto no modo "infraestrutura" quanto "ad hoc". Outra característica comum desses padrões é a utilização de faixas de frequência do espectro eletromagnético divididas em 14 canais sobrepostos, como mostra a Figura 3.4.

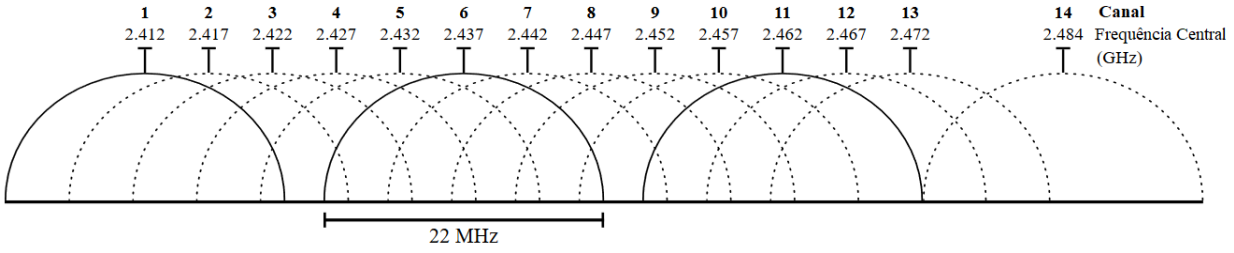

Figura 3.4: Canais do padrão IEEE 802.11

Pela Figura 3.4, percebe-se que dois canais não interferem entre si quando estão distantes por 4 ou mais canais. Entretanto, todos os 14 canais sofrem interferência dos canais laterais mais próximos.

Por meio do *software* GNU Radio conectado ao rádio USRP N210 utilizado, foi possível desenvolver um diagrama de blocos cuja função é coletar os sinais do espectro eletromagnético, como mostra a Figura 3.5.

A partir do projeto definido na Figura 3.5, é possível observar o estado do espectro eletromagnético, bem como capturar os sinais de fase $I(t)$ e quadratura $Q(t)$ que compõem esse espectro. Esses sinais são amostrados a uma frequência f_s, definida como $f_s = 25$ MHz, pois, como descrito por [1], frequências de amostragem maiores causam erros de escrita dos sinais $I(t)$ e $Q(t)$ em arquivo. Dessa forma, o período T_s entre a captura de uma amostra e outra é de 0,04μs.

Os valores de fase e de quadratura armazenados em arquivos são representados por números em ponto flutuante precisão simples (32 *bits*) segundo o padrão IEEE 754.

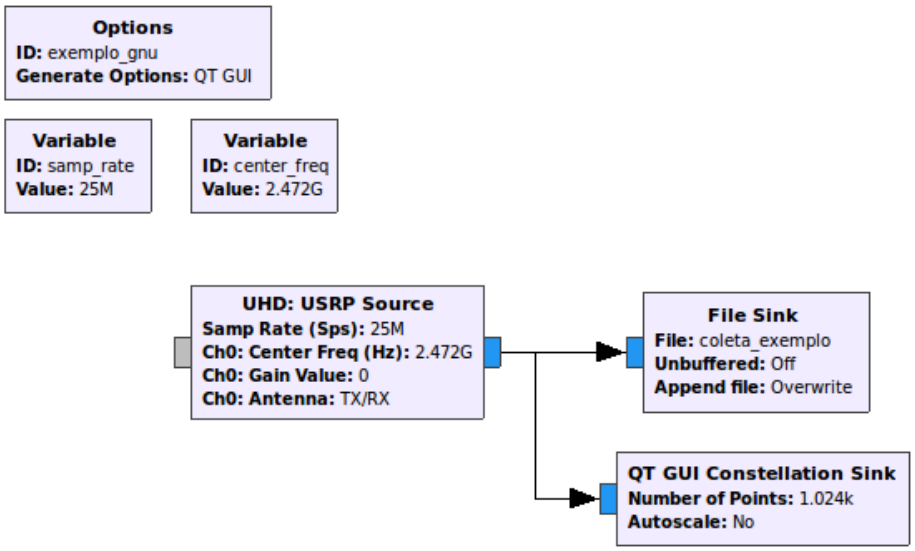

Figura 3.5: Diagrama de blocos utilizado para observar o espectro eletromagnético

O gráfico da Figura 3.6 mostra os sinais de fase e de quadratura e o espectrograma de um exemplo de uma coleta.

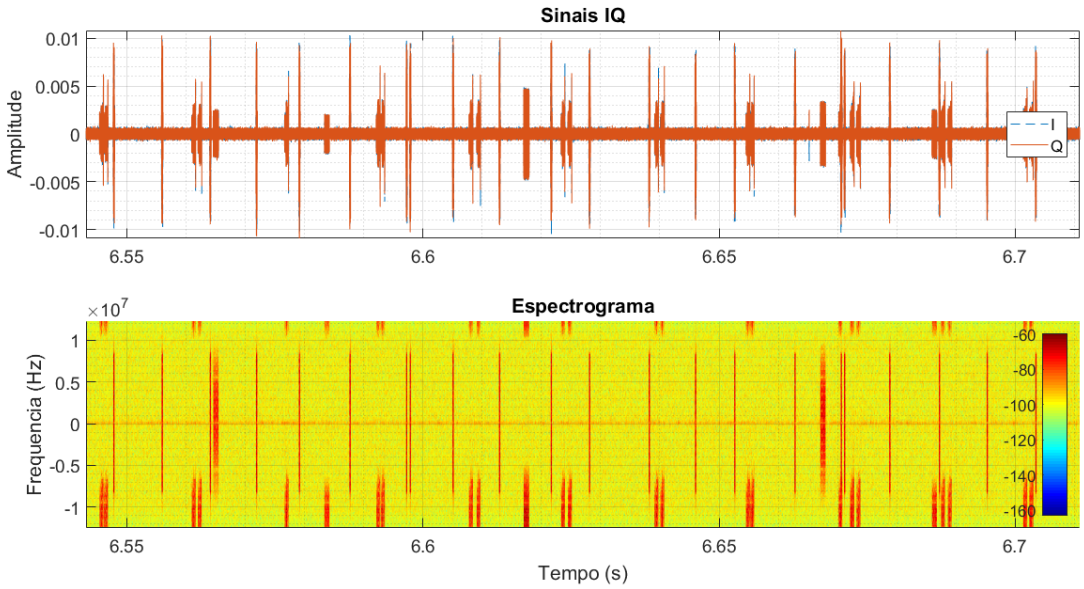

Figura 3.6: Exemplo de coleta do espectro eletromagnético e sinais $I(t)$ e $Q(t)$ realizada com o USRP N210

Percebe-se, pelo espectrograma da Figura 3.6, que, na transmissão considerada, há interferência dos canais laterais, como descrita na Figura 3.4. Esse fato pode ser concluído a partir dos sinais observados nas frequências não-centrais apresentados na figura.

3.3 Processamento de sinal

Após a contínua captura dos sinais $I(t)$ e $Q(t)$, como descrito na Seção 3.2, a energia presente no canal em um dado intervalo de tempo é estimada. Esse sinal de energia $E(n)$ é estimado no domínio da frequência a partir de N amostras temporais de $I(t)$ e $Q(t)$.

A estimação de $E(n)$ é efetuada no domínio da frequência para que seja possível filtrar a interferência dos canais laterais, mostrada pela Figura 3.4, de forma mais simples. Ou seja, no domínio da frequência é possível identificar diretamente valores de energia que se referem à energia presente na faixa de frequências em que há tanto o canal de interesse quanto canais laterais.

Assim, o sinal de energia $E(n)$ é calculado de acordo com

$$E(n) = \frac{1}{N} \sum_{k=F_0}^{F_1} |F(k)|^2, \tag{3.1}$$

em que F_0 é o limite inferior do filtro aplicado a uma janela, F_1 é o limite superior desse filtro e o espectro $F(k)$ do sinal recebido é dado por

$$F(k) = FFT_N\{W \cdot [I(t) + j \cdot Q(t)]\}. \tag{3.2}$$

Na Equação (3.2), o operador FFT_N representa o cálculo de uma FFT de N pontos, j, a unidade imaginária e W, a janela de Hamming [54] de N pontos, sendo que cada ponto $W(n)$ é definido por

$$W(n) = 0{,}54 - 0{,}46cos\left(2\pi \frac{n}{N}\right),\ 0 \leq\ n\ \leq\ N. \tag{3.3}$$

O tamanho N da janela deve ser definido cuidadosamente de forma a considerar não apenas a frequência de amostragem, mas também os detalhes do protocolo de transmissão utilizado pelo usuário primário. Neste trabalho, N é escolhido como sendo 128. Já os limites F_0 e F_1 utilizados para calcular a energia de uma janela de acordo com (3.1) são selecionados de forma a filtrar a saída da FFT e evitar interferências dos canais laterais, ou seja, de forma a eliminar amostras de energia observadas em baixas e em altas frequências em relação à frequência central do canal. Neste trabalho, F_0 e F_1 foram definidos como 32 e 120 respectivamente

O janelamento por Hamming dos sinais $I(t)$ e $Q(t)$ é necessário para evitar o chamado vazamento espectral, que é causado quando o sinal de entrada da FFT possui descontinuidades em suas extremidades. O janelamento de um sinal, isto é, a multiplicação de um sinal por uma janela, como a de Hamming, reduz os efeitos desse fenômeno. A janela de Hamming W utilizada tem $N = 128$ pontos e é descrita graficamente pela Figura 3.7.

Por fim, um exemplo de $E(n)$ calculado de acordo com a Equação (3.1), para os mesmos sinais $I(t)$ e $Q(t)$ mostrados na Figura 3.6 da Seção 3.2, é representado pela Figura 3.8. Se a frequência de amostragem de cada par de valores I e Q é f_s, o sinal de energia $E(n)$ é um sinal amostrado a uma frequência f_s/N.

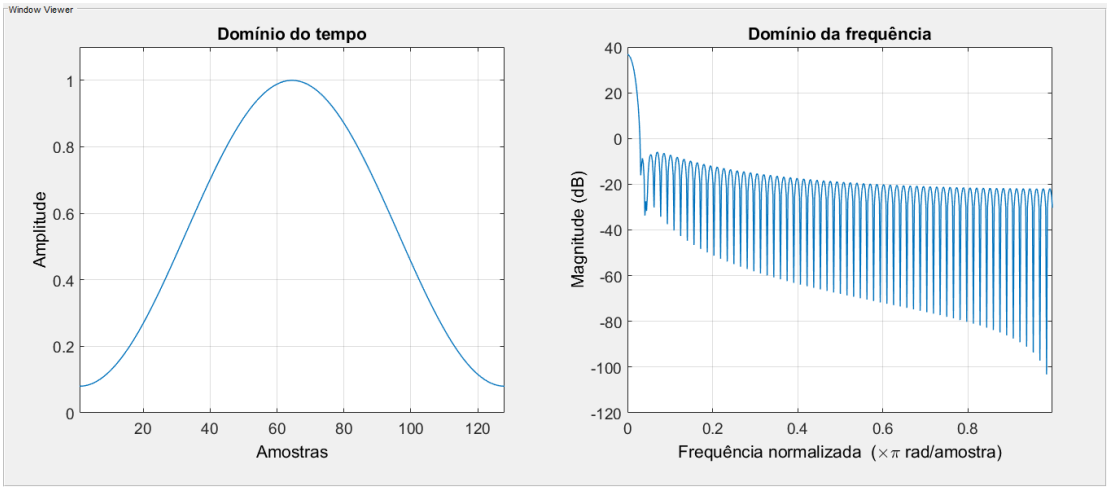

Figura 3.7: Janela de Hamming de 128 pontos nos domínios do tempo e da frequência

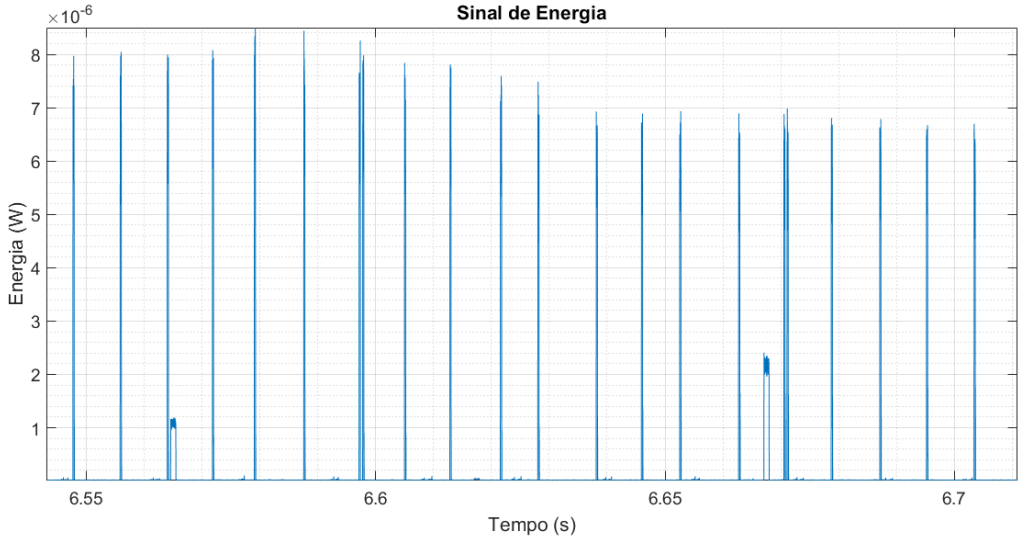

Figura 3.8: Exemplo de sinal de energia

3.3.1 Armazenamento e representação dos valores de I(t) e Q(t) e da janela de Hamming em *hardware*

Para armazenar os valores de $I(t)$ e $Q(t)$, foram definidos dois bancos de registradores de tamanho 2048. Cada um dos valores de $I(t)$ e $Q(t)$ é armazenado em um registrador de tamanho 16 *bits* e eles podem ser lidos na subida positiva do *clock* desde que seja fornecido o valor de seu endereço.

Para armazenar os valores da janela de Hamming, um banco de registradores também foi definido. Entretanto, o tamanho desse banco é de 128, uma vez que são necessários apenas 128 valores fixos para representar a janela utilizada. Cada uma das amostras da janela é também armazenada em um registrador de 16 *bits* e pode ser lida da mesma forma que os valores de $I(t)$ e $Q(t)$.

Para utilizar os valores amostrados dos sinais $I(t)$ e $Q(t)$ em *hardware*, é necessário representar seus valores na base binária. Nos blocos de propriedade intelectual da Intel definidos para o FPGA Altera DE2-115 Cyclone IV utilizado, cada operação de soma e multiplicação em ponto flutuante precisão simples requer uma quantidade específica de recursos físicos [55], como indicado pela Tabela 3.1.

Tabela 3.1: Recursos físicos utilizados em operações entre dois números de 32 *bits* representados em ponto flutuante

Operação	Recursos
Soma	170 elementos lógicos 377 registradores 0 multiplicadores de 9 bits
Multplicação	111 unidades lógicas 209 registradores 7 multiplicadores de 9 bits

As operações de soma e multiplicação entre dois números representados em ponto fixo por 32 *bits* também requerem quantidades específicas de recursos físicos, tal como detalhado pela Tabela 3.2.

Tabela 3.2: Recursos físicos utilizados em operações entre dois números de 32 *bits* representados em ponto fixo

Operação	Recursos
Soma	32 elementos lógicos 0 registradores 0 multiplicadores de 9 bits
Multplicação	28 elementos lógicos 0 registradores 6 multiplicadores de 9 bits

Assim, tendo em vista que multiplicações e somas em ponto flutuante são computacionalmente mais custosas do que operações em ponto fixo, como indicado pelas Tabelas 3.1 e 3.2, foi escolhido representar os valores dos sinais $I(t)$ e $Q(t)$ em ponto fixo.

A partir do gráfico dos sinais $I(t)$ e $Q(t)$ mostrado pela Figura 3.6 e de sinais de diversas coletas realizadas, percebeu-se que tanto $I(t)$ quanto $Q(t)$ possuem como valores máximo e mínimo, 0,011 e $-0{,}011$ respectivamente. Assim, para possibilitar a representação de $I(t)$ e $Q(t)$ com menos *bits*, seus valores foram escalados por 100, o que resulta em números que variam de $-1{,}1$ a 1,1. Esses

números, descritos em 16 *bits* Q13 (ou seja, 16 *bits*, sendo que os 13 *bits* menos significativos se encontram após a vírgula e os 3 mais significativos, antes da vírgula), são multiplicados pela janela de Hamming, cujos valores também são representados em ponto fixo 16 *bits* Q13. Entretanto, no caso dos valores da janela de Hamming W, o escalamento não é necessário uma vez que o valor máximo de W é 1, conforme pode ser observado na Figura 3.7.

As multiplicações entre os valores de $I(t)$ e da janela de Hamming W e entre os valores de $Q(t)$ e de W resultam em números de 32 *bits* Q26. Assim, para obter as partes real e imaginária da entrada da FFT em representação 16 *bits* Q13, é necessário fazer um deslocamento de 13 *bits* à direita e selecionar os 16 *bits* menos significativos do resultado dos deslocamentos. Dessa forma, são obtidas as partes real e imaginária do sinal a ser entregue ao módulo de FFT.

3.3.2 Transformada Rápida de Fourier em *Hardware*

A Transformada rápida de Fourier é calculada por meio de um bloco de propriedade intelectual (IP) desenvolvido pela Unicore Systems e disponibilizado pela OpenCores [5]. Este bloco, descrito na linguagem de *hardware* Verilog, é capaz de executar o algoritmo da FFT radix-8 de 128 valores complexos e da função inversa da FFT. Os valores de entrada e os coeficientes da FFT devem ter tamanhos que variem entre 8 e 16 *bits*. Tanto os valores de entrada quanto os de saída são representados por inteiros complexos em complemento de dois. Além disso, se os dados de entrada possuem nb *bits*, os valores de saída são representados em $nb + 4$ *bits*.

É possível configurar a saída da FFT para retornar os valores na ordem direta, o que é computacionalmente mais custoso, ou na ordem bit reversa. No primeiro caso, o atraso latente entre a apresentação da entrada e o fornecimento da saída é de 440 ciclos de *clock*, enquanto que, no segundo, é de 310 ciclos de *clock*. Um diagrama representando a interface de entrada e saída desse bloco IP é mostrado na Figura 3.9.

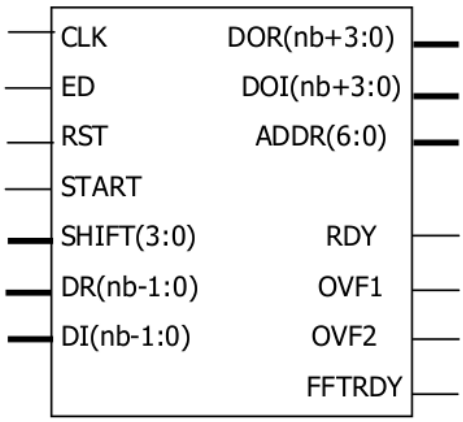

Figura 3.9: Interface de entrada e saída do bloco de FFT do OpenCores [5]

Os sinais representados no lado esquerdo do bloco da Figura 3.9 são os sinais de entrada do bloco IP e os sinais no lado direito, são os sinais de saída. Todos os sinais apresentados na Figura 3.9 são detalhados na Tabela 3.3.

Tabela 3.3: Descrição dos sinais de entrada e saída do bloco de FFT do OpenCores [5]

Sinal	Descrição
CLK	Sinal global de *clock*
ED	Sinal que habilita a operação da FFT
RST	Sinal global de *reset*
START	Sinal para inicializar a operação da FFT
SHIFT(3:0)	Quantidade de bits deslocados à esquerda após cálculos intermediários do bloco
DR(nb-1:0)	Parte real de uma amostra de entrada da FFT
DI(nb-1:0)	Parte imaginária de uma amostra de entrada da FFT
DOR(3:0)	Parte real de uma amostra de saída da FFT
DOI(3:0)	Parte imaginária de uma amostra de saída da FFT
ADDR(6:0)	Endereço da amostra na saída do bloco
RDY	Indica que o resultado da FFT está pronto
OVF1	Indica a presença de *overflow* durante os cálculos intermediários do bloco
OVF2	Indica a presença de *overflow* no último cálculo do bloco
FFTRDY	Indica que o bloco está pronto para receber dados de entrada

Os sinais das quatro primeiras linhas da Tabela 3.3 devem ser cuidadosamente controlados para que a saída da FFT seja gerada. O comportamento desses sinais, quando controlados, é descrito pela Figura 3.10, em que o símbolo de reticências refere-se à quantidade necessária de ciclos de *clock* para que o bloco da FFT gere saídas válidas (evento indicado pelo sinal *rdy*).

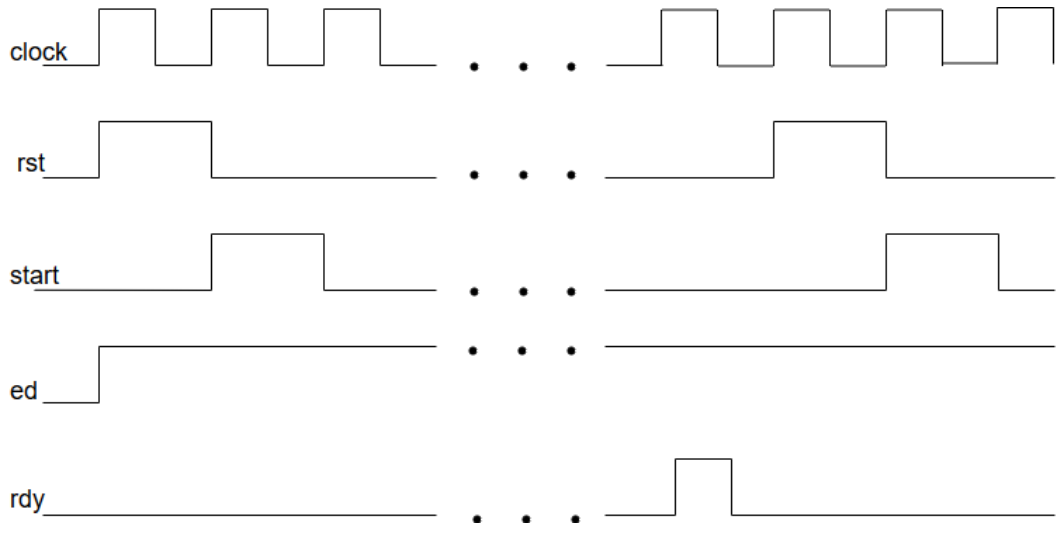

Figura 3.10: Controle dos sinais do bloco da FFT

Neste trabalho, foram utilizadas entradas cujas partes real *dr* e imaginária *di* são descritas por $nb = 16$ *bits* ponto fixo Q13. Portanto, tem-se que a saída da FFT é composta de partes real *dor* e imaginária *doi* ambas com 20 *bits* Q13. Dessa forma, a energia $E(n)$, calculada por (3.1), tem

seus valores representados por 40 *bits* Q26.

Para validar o cálculo da DFT feito pelo bloco de FFT, alguns sinais de entrada foram selecionados e representados em 16 *bits* ponto fixo Q13. Esses sinais de entrada, representados graficamente na Figura 3.11 são: a constante 1, um pulso 1 na primeira amostra, dois períodos de seno e dois períodos de cosseno.

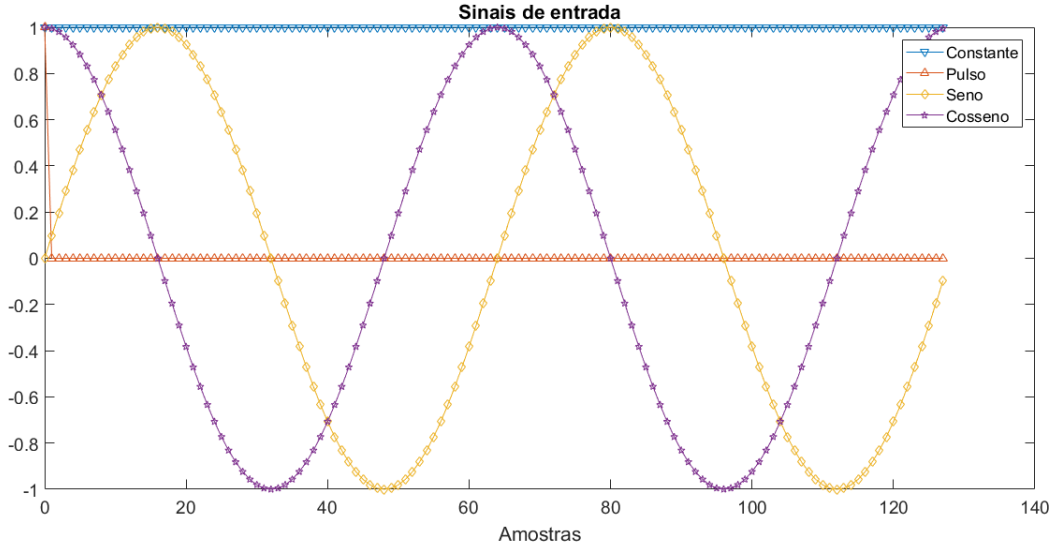

Figura 3.11: Sinais fornecidos na entrada do bloco da FFT utilizados para testar seu funcionamento

As partes real e imaginária das saídas geradas pelo bloco de FFT a partir dessas entradas são apresentadas respectivamente nas Figuras 3.12 e 3.13, em gráficos com dois eixos y, cada um em uma escala distinta.

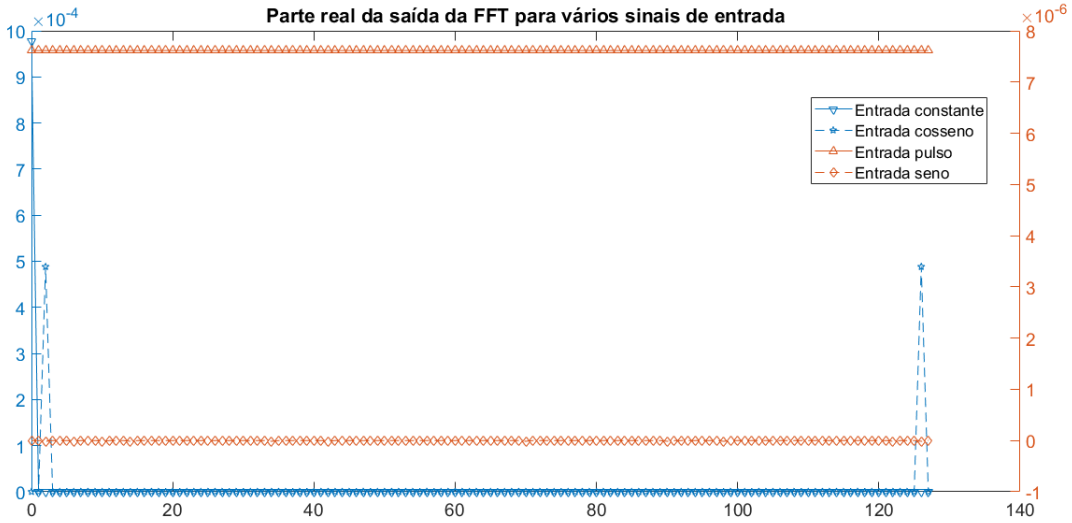

Figura 3.12: Parte real da saída da FFT para os sinais de entrada representados na Figura 3.11

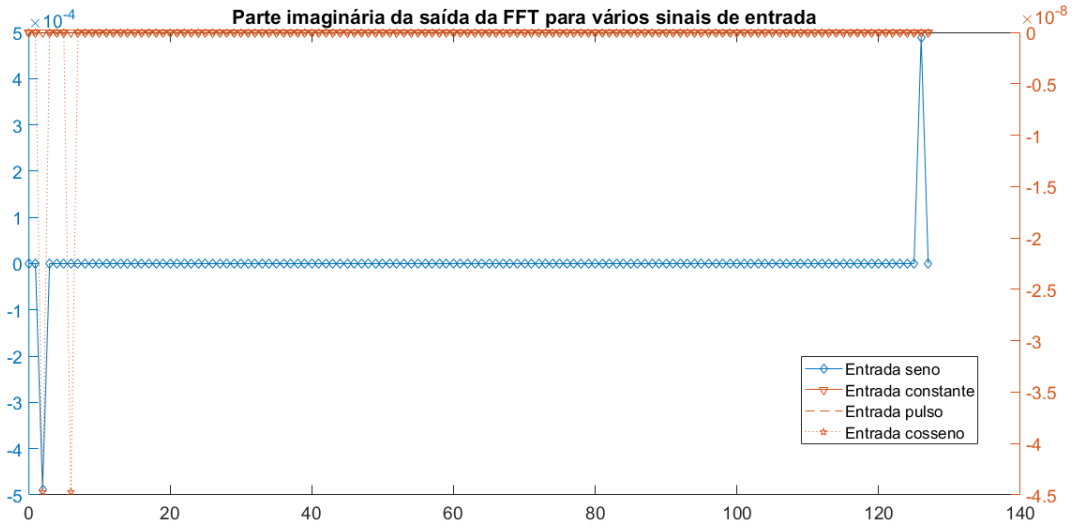

Figura 3.13: Parte imaginária da saída da FFT para os sinais de entrada representados na Figura 3.11

Ao se comparar os resultados corretos da DFT dos sinais utilizados nesse teste e os resultados dados pelo bloco da FFT (descritos pelas Figuras 3.12 e 3.13), percebe-se que os valores na saída do bloco definido em *hardware* são escalados por um fator aproximadamente igual a $7{,}614 \cdot 10^{-6}$. Assim, a parte real da saída da FFT para uma entrada pulso, que deveria ser constante igual a 1, é, na verdade, constante igual a $7{,}614 \cdot 10^{-6}$. De forma análoga, na Figura 3.12, a saída para a entrada constante possui valor máximo em $9{,}765 \cdot 10^{-4}$ e, para a entrada cosseno, em $4{,}883 \cdot 10^{-4}$, o que, de forma escalada, equivale aos valores esperados 128 e 64 respectivamente.

Neste teste de validação, em particular, os sinais de entrada não foram multiplicados pela janela de Hamming, uma vez que o objetivo do teste era apenas verificar a acurácia do bloco da FFT disponibilizado pela OpenCores. Para demonstrar o efeito do janelamento das entradas, pode-se definir como entrada do bloco de FFT o sinal de cosseno descrito na Figura 3.11 multiplicado pela janela de Hamming representada graficamente na Figura 3.7. Assim, pode-se obter a saída da FFT dada pelo gráfico da Figura 3.14.

3.4 Quantização

O sinal de energia $E(n)$ obtido após o processamento dos sinais $I(t)$ e $Q(t)$ é aplicado no quantizador, que mapeia $E(n)$ em uma sequência de símbolos q de acordo com a sua intensidade. No caso mais simples, apenas os símbolos 0 e 1 são utilizados, indicando, respectivamente, a ausência e a presença de energia no canal. Neste trabalho, as sequências de sinais utilizados nos experimentos conduzidos são limitadas a esse caso mais simples, uma vez que o foco deste trabalho é direcionado às redes neurais. Portanto, o quantizador é reduzido a uma simples comparação do sinal de energia a um limiar ξ de acordo com

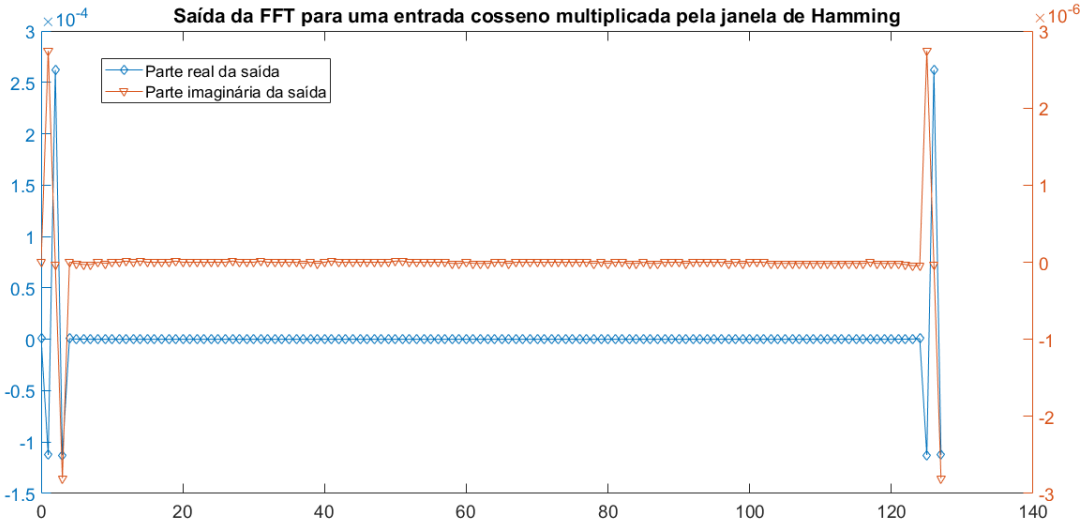

Figura 3.14: Saída da FFT para um sinal de dois períodos de cosseno multiplicado pela janela de Hamming

$$q(n) = \begin{cases} 1, & \text{se } E(n) \geq \xi \\ 0, & \text{caso contrário.} \end{cases} \tag{3.4}$$

Apesar da adoção dessa técnica simples de quantização, técnicas adaptativas podem ser utilizadas para melhorar a performance do sistema em aplicações mais complexas. O valor do quantizador é definido como $1{,}5 \cdot 10^{-7}$ para situações em que a energia é calculada por

$$E = \frac{1}{N^2} \sum_{0}^{N-1} (O_r^2 + O_i^2), \tag{3.5}$$

em que O_r e O_i representam as partes real e imaginária da saída da FFT e N é o tamanho da janela, escolhida como sendo 128.

Neste trabalho, como explicado na Seção 3.3, os valores de $I(t)$ e $Q(t)$ são multiplicados por 100 antes do cálculo da FFT ser realizado na aplicação em *hardware*. Essas multiplicações geram valores de energia escalados por uma constante em relação à energia calculada sem a multiplicação por 100 dos sinais de fase e quadratura. A partir de experimentos que comparam os valores de energia calculados sem a escala por 100 e com essa escala, foi descoberto que o valor de ξ deve ser dividido por 0,0256.

Além disso, tem-se que, para evitar a representação de valores com muitos algarismos após a vírgula, a divisão por N^2 da Equação (3.5) não é realizada. Dessa forma, tem-se que, em *hardware*, o valor de ξ é definido por

$$\xi = \frac{1{,}5 \cdot 10^{-7} \cdot 128^2}{0{,}0256} = 0{,}096. \tag{3.6}$$

A representação do valor de ξ definido por (3.6) em ponto fixo 40 *bits* Q26 é $0000624DD3_{16}$.

Assim, para gerar a sequência q de $0's$ e $1's$ que exerce o papel de entrada da rede neural, como indicado no diagrama da Figura 3.2 da Seção 3.1, os valores de energia $E(n)$ representados por valores de 40 *bits* ponto fixo Q26 são comparados com $\xi = 0000624DD3_{16}$.

Um exemplo de quantização de um sinal de energia é mostrado na Figura 3.15.

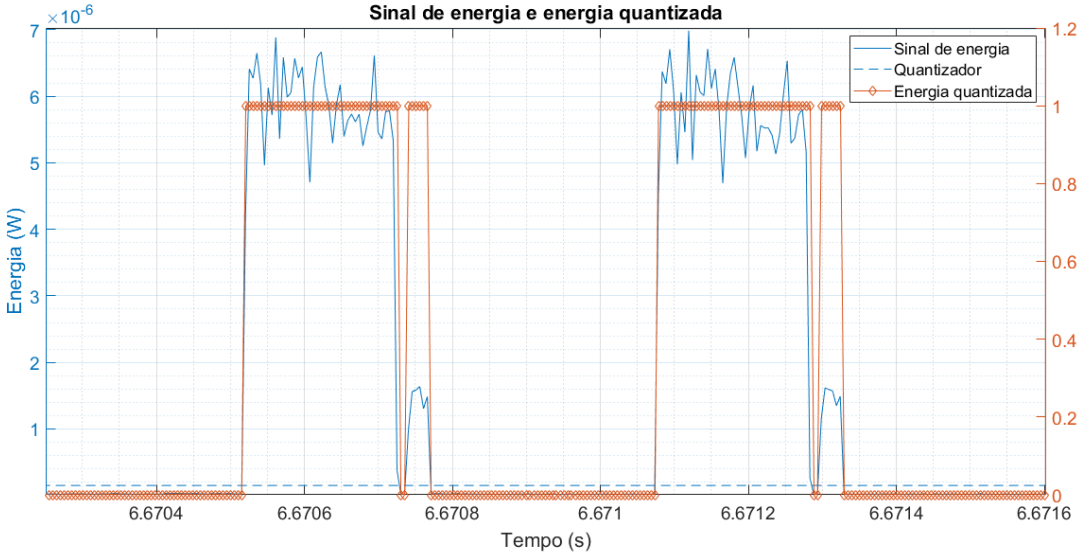

Figura 3.15: Exemplo de sinal de energia quantizado

A Figura 3.15 mostra que, após a quantização, a sequência de $0's$ e $1's$ é gerada de acordo com a comparação com o quantizador. Essa sequência é, então, utilizada como entrada da rede neural.Na próxima seção, será apresentada a metodologia utilizada para aplicar sequências quantizadas na entrada da rede neural.

3.5 Rede neural

Para descrever uma rede neural treinada em Verilog, um *script* feito em Matlab e um programa em linguagem C são utilizados. Esses programas definem uma rede neural em Verilog equivalente a uma rede treinada e declarada em Matlab. Na definição em Verilog, todos os pesos da rede são representados em ponto fixo, sendo que a quantidade de *bits* e o valor de Q dos pesos podem ser escolhidos de acordo com o problema.

A rede utilizada neste trabalho foi treinada com a função *train* presente no *Neural Network Toolbox* [18] do Matlab com a variação Levenberg–Marquardt do algoritmo de *backpropagation*. Estruturalmente, essa rede é composta por duas camadas recorrentes do tipo Elman com 12 neurônios na camada escondida e 2 neurônios na camada de saída. A escolha dessa estrutura de rede partiu de um amplo estudo [1] em que redes com quantidades diversas de neurônios na camada escondida foram desenvolvidas e testadas.

A definição e o treino da rede utilizada são os mesmos descritos em [1], assim os dois neurônios

da camada de saída correspondem respectivamente às classes de não-oportunidade e de oportunidade. Além disso, a função de ativação de todos os neurônios das duas camadas é a tangente hiperbólica e, como definido na Seção 2.1.4, a função de ativação dos neurônios da camada de contexto é sempre linear. A representação gráfica fornecida pelo Matlab da rede utilizada é mostrada na Figura 3.16.

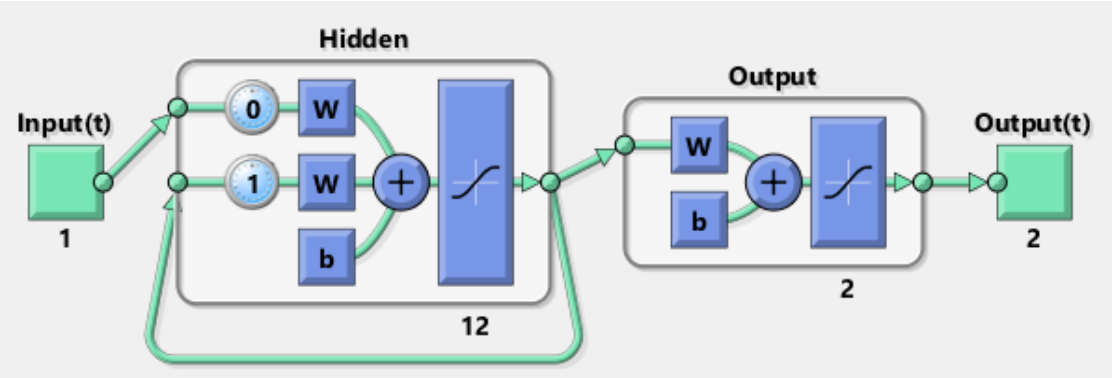

Figura 3.16: Rede treinada no Matlab e utilizada neste trabalho

A saída dos neurônios pode assumir valores no intervalo $[-1, 1]$, uma vez que a função de ativação dos neurônios de saída é a tangente hiperbólica, como pode ser visto pela Figura 3.16. Entretanto, a rede utilizada foi treinada para entradas e respostas desejadas (*targets*) que assumem apenas dois valores: 0 ou 1. Dessa forma, a rede é treinada para mostrar, na saída de seus dois neurônios, valores pertencentes ao intervalo $[0, 1]$. Assim, pode-se definir as classes c de saída dos neurônios $i = 1, 2$ como sendo verdeiras ou falsas a partir do valor da saída y de cada um dos neurônios para uma dada amostra n de acordo com:

$$c_i(n) = \begin{cases} verdadeiro, & \text{se } y(n) \geq 0{,}5 \\ falso, & \text{caso contrário,} \end{cases} \tag{3.7}$$

sendo que a classe do neurônio $i = 1$ corresponde à não precedência de oportunidade e a classe do neurônio $i = 2$, à precedência de oportunidade, como definido em [1].

3.5.1 Representação dos pesos da rede em *hardware*

Para manter a precisão do cálculo realizado pela rede neural, foi escolhido o tamanho máximo de 32 *bits* para representar seus pesos. O valor de Q foi definido como sendo 20, o que permite a representação de valores na faixa $[-1024, 1024]$. Como, a rede utilizada, representada na Figura 3.16, não tem pesos de magnitude maior que 4, bastariam 7 *bits* para representar a parte inteira dos valores dos pesos e dos resultados dos cálculos intermediários executados pela rede sem que ocorra *overflow*. É possível provar isso por meio de uma análise do pior caso.

Para a rede selecionada, com 12 neurônios na camada escondida e 2 neurônios na camada de saída, os neurônios da camada escondida são os que recebem uma maior quantidade de valores em suas entradas. Ao total, as entradas de um neurônio da camada escondida são as saídas dos

12 neurônios da camada de contexto, o valor da entrada da rede e o valor do *bias*. Assim, cada neurônio da camada escondida calcula a soma ponderada de 14 entradas. No pior caso, todos os pesos que se conectam a um neurônio dessa camada são iguais a 4 ou a -4 e as 14 entradas são todas iguais aos valores 1 ou -1, o que resulta em um valor máximo igual a 56 ou -56 como entrada da função de ativação do neurônio da camada escondida. Sendo assim, para representar a faixa dinâmica $[-56, 56]$ em complemento de dois, é necessário apenas 7 *bits* antes da vírgula.

3.5.2 Funções de ativação em *hardware*

As funções de ativação são definidas de forma aproximada dentro de um módulo em Verilog. No total, há 7 aproximações de algumas funções de ativação conhecidas. Essas aproximações definidas em Verilog são apresentadas graficamente na Figura 3.17.

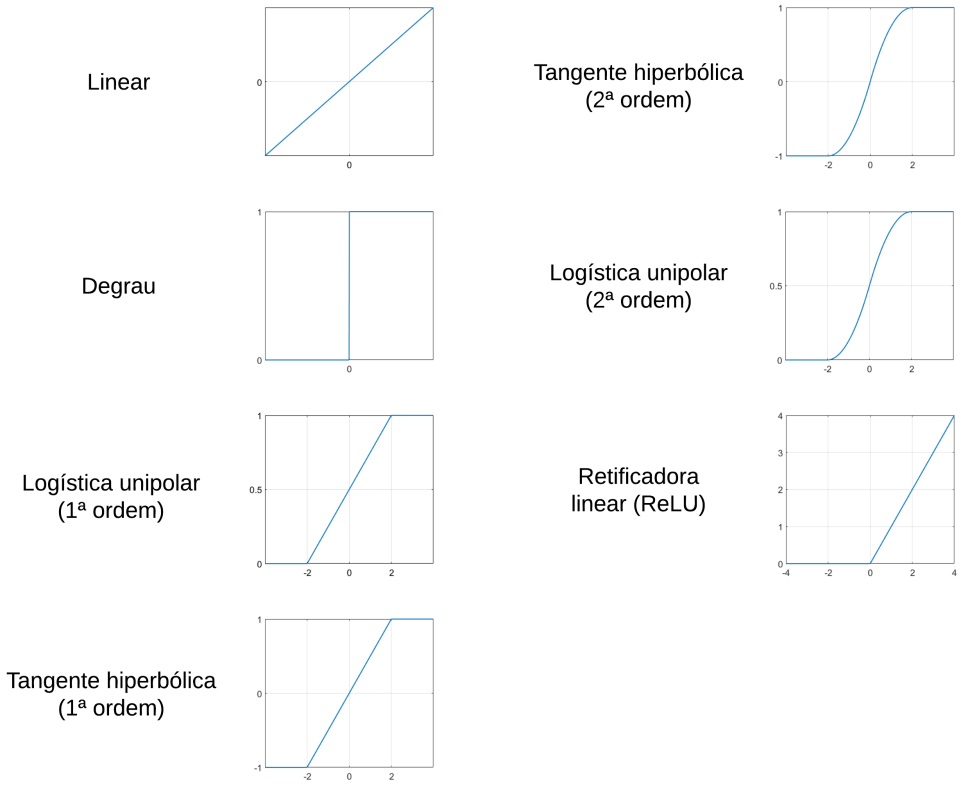

Figura 3.17: Aproximações de funções de ativação utilizadas

As funções linear, degrau e retificadora linear mostradas na Figura 3.17 foram desenvolvidas de forma exata em Verilog. Já as funções sigmoide unipolar e sigmoide bipolar, implementadas, respectivamente, pelas funções logística e tangente hiperbólica mostradas na Figura 2.3 da Seção 2.1, foram desenvolvidas de forma aproximada em *hardware*. Para cada uma das duas funções, foram definidas aproximações de 1ª e de 2ª ordem, ou seja, aproximações por segmentos de reta e por parábolas, respectivamente. As aproximações da função logística, apresentada pela Equação (2.7)

da Seção 2.1, por polinômios de primeira e de segunda ordem são definidas respectivamente por

$$\varphi(v) = \begin{cases} 1, & v > 2 \\ \frac{v+2}{4}, & -2 \leq v \leq 2 \\ 0, & v < -2 \end{cases} \tag{3.8}$$

e

$$\varphi(v) = \begin{cases} 1, & v > 2 \\ \frac{-(v-2)^2}{8} + 1, & 0 \leq v \leq 2 \\ \frac{(v+2)^2}{8}, & -2 \leq v < 0 \\ 0, & v < -2 \end{cases}. \tag{3.9}$$

Já as aproximações de primeira e de segunda ordem para a função tangente hiperbólica, indicada pela Equação (2.8) são respectivamente definidas por

$$\varphi(v) = \begin{cases} 1, & v > 2 \\ v/2, & -2 \leq v \leq 2 \\ -1, & v < -2 \end{cases} \tag{3.10}$$

e

$$\varphi(v) = \begin{cases} 1, & v > 2 \\ \frac{-(v-2)^2}{4} + 1, & 0 \leq v \leq 2 \\ \frac{(v+2)^2}{4} - 1, & -2 \leq v < 0 \\ -1, & v < -2 \end{cases}. \tag{3.11}$$

3.5.3 Modelo dos neurônios em *hardware*

Cada neurônio da rede é definido por um módulo em Verilog, cuja estrutura e funcionamento depende da natureza do neurônio, isto é, *feedforward* ou recorrente. O módulo que define o primeiro tipo de neurônio tem como entradas os valores de entrada do neurônio e os pesos que conectam essas entradas ao neurônio. Como geralmente neurônios artificiais possuem mais de uma entrada, é declarado um conjunto de fios (*bus*) para representar tanto as entradas quanto seus respectivos pesos. Além disso, cada módulo de neurônio *feedforward* tem também como entrada o tipo de função de ativação que deve ser aplicada à soma ponderada. Por fim, esses módulos fornecem apenas uma saída, que é a própria saída do neurônio. Um diagrama de blocos de um desses módulos para um neurônio *feedforward* é indicado na Figura 3.18.

O módulo de um neurônio recorrente é estruturado de forma bastante similar ao de um neurônio *feedforward*. No caso mais geral, um neurônio recorrente deve receber entradas, pesos, o tipo de função de ativação, um sinal de *clock* e um sinal de *reset*. Esses dois últimos sinais devem ser

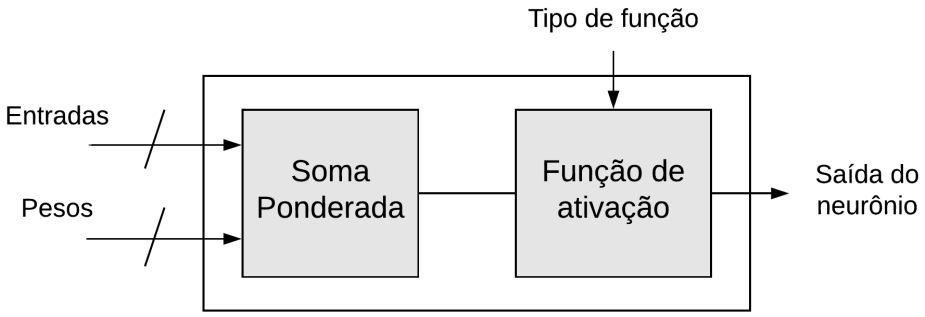

Figura 3.18: Modelo de neurônio *feedforward* em *hardware*

incluídos no modelo de neurônio recorrente para, respectivamente, regular a disponibilização de sua saída e para reiniciar a camada de contexto. Para que a saída de um neurônio recorrente seja regulada por esses sinais de *clock* e de *reset*, ela deve ser armazenada em um registrador. Assim, em uma dada iteração, a saída de um neurônio de contexto é a saída do neurônio da camada escondida correspondente na iteração anterior ou o valor zero, caso esse neurônio tenha sido reiniciado. O diagrama de blocos do módulo de neurônio recorrente é mostrado pela Figura 3.19.

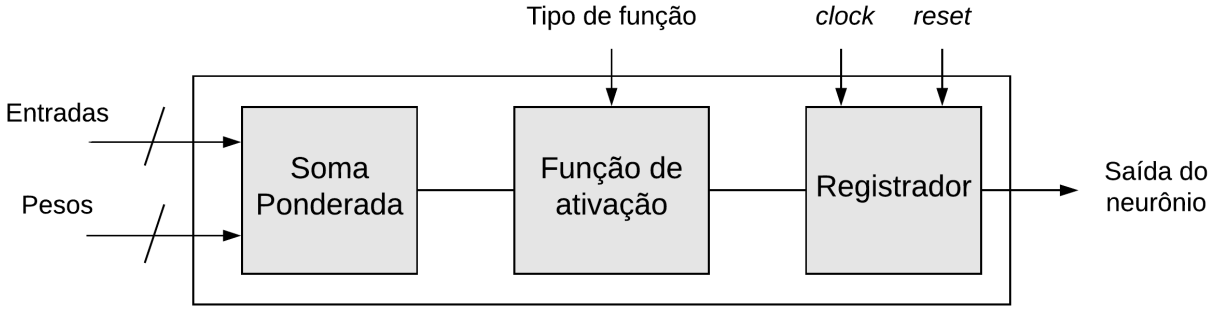

Figura 3.19: Modelo de neurônio recorrente em *hardware*

Apesar dos módulos de neurônios recorrentes permitirem a definição de neurônios versáteis, percebe-se que, pelas definições apresentadas na Seção 2.1 relativas à rede do tipo Elman, todos os módulos de neurônios recorrentes têm apenas uma entrada e um peso (e não conjuntos de entradas e pesos) e seu tipo de função de ativação é sempre o linear.

Na definição de uma rede neural em Verilog, existem tantos módulos de neurônios quanto a quantidade de neurônios na rede. Dessa forma, para a rede utilizada com 12 neurônios na camada escondida e 2 neurônios de saída, são descritos 14 módulos de neurônios *feedforward* e 12 módulos de neurônios recorrentes.

3.6 Controlador

Para sincronizar o funcionamento das etapas definidas na Figura 3.1 da Seção 3.1 e explicadas ao longo das Seções 3.2 a 3.5, é necessário definir um controlador por máquina de estados finitos

(MEF). O diagrama do controlador desenvolvido neste projeto é apresentado na Figura 3.20.

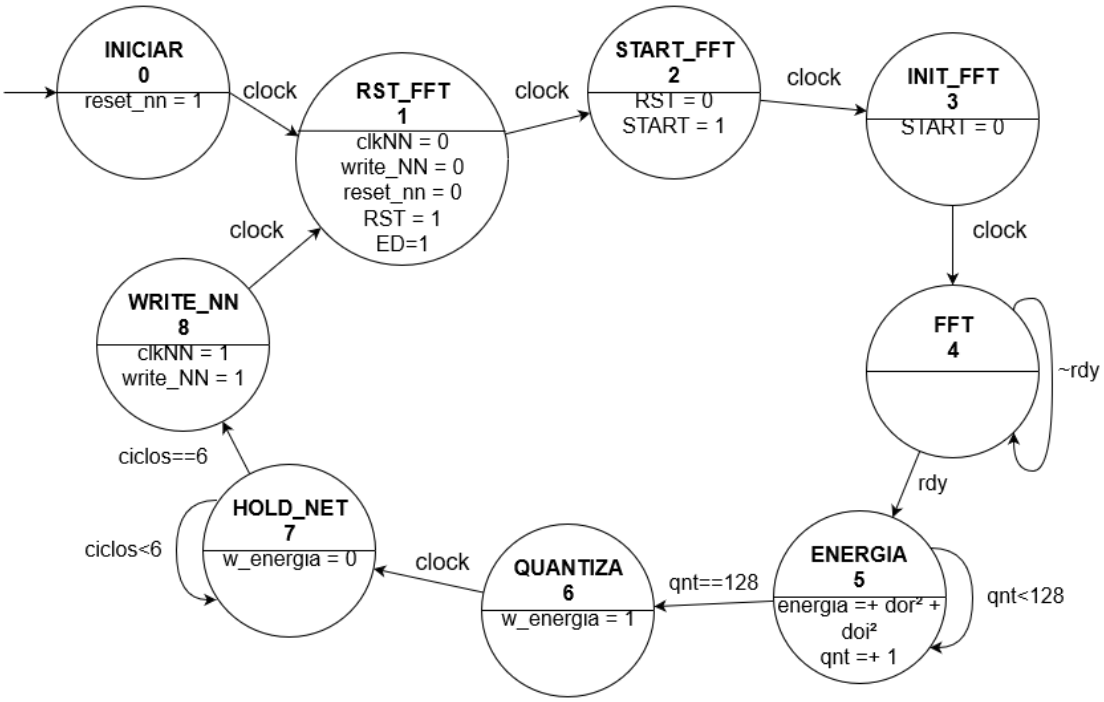

Figura 3.20: Máquina de estados finitos que representa o controlador

O estado inicial do sistema é identificado como sendo o estado 0 “INICIAR". Este estado tem como objetivo apenas a inicialização das saídas dos neurônios de contexto da rede neural com o valor 0 e só é possível transicionar para esse estado na inicialização do sistema e quando há um *reset* geral aplicado de forma externa e manual.

As transições da máquina de estados finitos com o termo “*clock*” são executadas sempre que há uma borda positiva de *clock*. Por definição, transições em MEFs ocorrem de acordo com o *clock*, mas, para explicitar a transição não condicional, foi adicionado o termo.

Os estados 1 a 3 são referentes ao controle de sinais do bloco da FFT, como indicado pela forma de onda da Figura 3.10. Esses estados são fundamentais para que o bloco de FFT comece a executar os cálculos de forma correta e sincronizada

Durante o estado 4, as 128 entradas da FFT são apresentadas ao bloco de FFT a cada ciclo de *clock* e espera-se até o momento em que a saída da FFT estiver disponível (fato que é indicado pelo sinal positivo de *rdy*). A transição para o estado 5 apenas ocorre quando *rdy* se torna um sinal lógico positivo.

No estado 5, o valor de energia é calculado, sendo necessário um ciclo de *clock* para cada sinal de saída da FFT (composto por uma parte real *dor* e uma parte imaginária *doi*). Assim, um contador *qnt* é definido para indicar quantos elementos foram utilizados no cálculo de energia. Quando *qnt* é igual a 128, pode-se afirmar que o cálculo de energia foi finalizado. Assim, transiciona-se para o estado 6, em que ocorre o processo de quantização e é fornecido à rede neural um sinal 0 ou um sinal 1. Durante esse estado, ocorre também a escrita (*w_energia*=1), em memória, do valor de

energia calculado anteriormente.

O estado 7 é necessário para evitar que a saída da rede neural seja considerada válida antes de ter se estabilizado. O objetivo desse estado é apenas introduzir um atraso de 6 ciclos de *clock* ao sistema, entre a apresentação da entrada à rede e a leitura de sua saída. A quantidade de 6 ciclos foi definida de acordo com experimentos conduzidos em que, por meio de formas de onda, foi possível observar que são necessários pelo menos 60 ns para que a saída da rede se estabilize. Assim, considerando um *clock* de 50MHz, seriam necessários ao menos 3 ciclos de *clock*. Para garantir que não ocorram erros, o dobro de ciclos necessários foi utilizado.

No estado 8, a saída da rede neural é armazenada em memória e a sua camada de contexto é atualizada para a próxima amostra de entrada. Após a execução do estado 8, o sistema retorna ao estado 1 para reinicializar os passos necessários para a execução do processamento de sinais, cálculo de energia, quantização e previsão de oportunidades realizada pela rede neural.

Os valores dos sinais $I(t)$ e $Q(t)$ multiplicados pela janela de Hamming estão disponíveis em qualquer um dos estados que o sistema possa assumir.

3.7 Sistema em *software* e adaptação da rede neural

O sistema desenvolvido em linguagem C++ define redes do tipo Elman com apenas uma camada escondida e, a partir de uma rede, é possível escolher três funcionalidades distintas:

1. Simulação da rede, que é apenas a aplicação de entradas pelo caminho *feedforward* de uma rede previamente treinada;
2. Treino de acordo com o algoritmo de *backpropagation* com a opção de incluir a variável de momento. Ou seja, essa funcionalidade inicializa uma rede com pesos aleatórios e a treina a partir de um conjunto de entradas e respostas desejadas;
3. Retreino da rede, que consiste em aplicar o algoritmo de *backpropagation* por poucas iterações em uma rede previamente treinada.

Apesar de possuir várias funções e a possibilidade de alteração de algumas variáveis, o objetivo fundamental do sistema desenvolvido é executar o retreino de uma rede, denominada por rede base, de forma *online* e garantir a adaptabilidade da rede de acordo com o ambiente no qual ela está inserida, como apresentado no capítulo de metodologia pelo diagrama da Figura 3.2. Assim, a partir dos testes realizados em ambientes de *software* para garantir a adaptabilidade da rede, é possível descobrir uma metodologia bem-sucedida para ser desenvolvida posteriormente em *hardware*.

A rede base utilizada neste trabalho é a mesma apresentada na Figura 3.16 da Seção 3.5, ou seja, uma rede treinada por meio da variação Levenberg–Marquardt do *backpropagation*, que é capaz de treinar redes em uma velocidade maior quando comparado ao *backpropagation* com gradiente descendente. Entretanto, sua complexidade computacional é maior, uma vez que requer

o cálculo de matrizes Hessianas. Portanto, tendo em vista a limitação de recursos computacionais do FPGA e visando a posterior implementação do sistema de adaptação da rede nesta forma de *hardware*, o algoritmo de treino do sistema em *software* foi escolhido como sendo a variação do gradiente descendente do *backpropagation*.

Além de utilizar as funcionalidades de treino e de retreino desse sistema para testar métodos de adaptação de redes neurais, a funcionalidade de simulação também foi necessária para a validação tanto do sistema em *software* quanto em *hardware*.

Capítulo 4

Resultados

Este capítulo apresenta a caracterização dos modelos propostos em *hardware* e em *software*, bem como testes executados para demonstrar a validade dos sistemas propostos. A Seção 4.1 caracteriza os modelos de neurônio definidos em *hardware* em termos de recursos físicos e temporais e também demonstra a validade das aproximações das funções de ativação adotadas em *hardware*. A Seção 4.2 caracteriza o sistema completo, ou seja, discorre sobre os gastos de um sistema que segue as etapas de aquisição, processamento de sinal, quantização e rede neural detalhadas no Capítulo 3. A Seção 4.3 compara os resultados da saída do sistema descrito em *hardware* e os sistemas definidos em *software*. A Seção 4.4, por fim, mostra os resultados obtidos nos testes realizados para adaptar a rede de forma *online*.

Os resultados mostrados neste capítulo foram obtidos por meio da aplicação do sistema no *hardware* FPGA Cyclone IV apresentado na Seção 2.5.2 associado ao *software* de desenvolvimento Quartus Prime 18.0 Lite Edition. A linguagem de descrição de *hardware* escolhida para definir o projeto é o Verilog. O sistema descrito em *software* foi definido por meio das linguagens Matlab na versão R2017a e em C++11 associado ao compilador gcc 5.4.0.

4.1 Caracterização do modelo de neurônio desenvolvido em *hardware*

Nesta seção, são apresentados os resultados relativos aos experimentos de caracterização dos dois modelos de neurônio desenvolvidos em descrição de *hardware*, isto é, o neurônio do tipo *feedforward* e o de contexto. A caracterização desses modelos envolve tanto a definição de suas complexidades em termos de recursos físicos quanto a descrição dos atrasos referentes às operações internas que ocorrem nos neurônios.

4.1.1 Neurônio *feedforward*

Os recursos utilizados por um neurônio do tipo *feedforward*, descrito pela Figura 3.18 da Seção 3.5, são diretamente dependentes da quantidade de entradas que ele recebe. Se for considerado

que um neurônio deve receber pelo menos duas entradas (uma entrada e um *bias*), a quantidade de recursos físicos utilizada por cada neurônio do tipo *feedforward* é descrita pela Tabela 4.1, em que x indica a quantidade de entradas +1 (referente ao bias) que alimentam o neurônio.

Tabela 4.1: Recursos físicos utilizados por um neurônio *feedforward* em função da quantidade x de entradas +1

Recurso	Quantidade utilizada
Elementos lógicos	$113x - 139$
Registradores	0
Multiplicadores de 9 *bits*	$8x - 8$

Percebe-se, pela Tabela 4.1, que neurônios do tipo *feedforward* não utilizam registradores, o que condiz com a teoria de redes neurais apresentada no Capítulo 2, uma vez que esses neurônios não armazenam dados entre a apresentação de duas entradas subsequentes.

Além dos recursos necessários ao funcionamento de um neurônio, seu tempo de resposta também depende da quantidade de entradas que ele recebe. O atraso de propagação *tpd* entre a apresentação das entradas e a finalização do cálculo da soma ponderada dessas entradas para um neurônio com $2, 3, \ldots, 14$ entradas de 32 *bits* é mostrado no gráfico da Figura 4.1, bem como uma aproximação polinomial de primeira ordem desses atrasos, cuja equação é dada por

$$tpd = 1{,}2701x + 21{,}9861, \tag{4.1}$$

em que x é a quantidade de entradas, incluindo o bias, que são apresentadas ao neurônio.

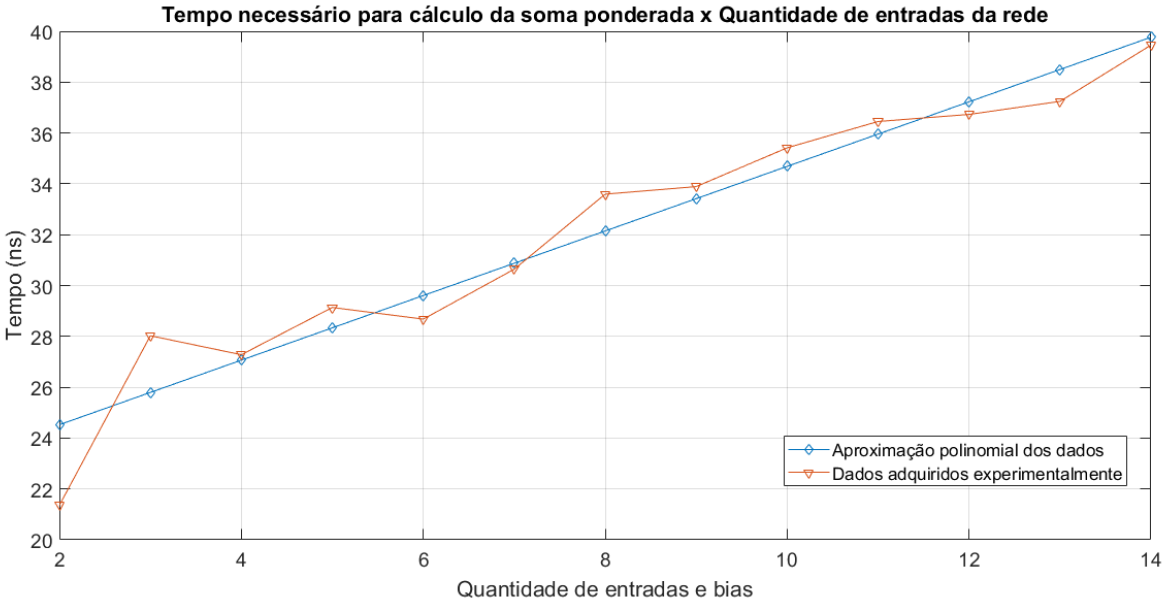

Figura 4.1: Atraso de propagação de um neurônio pela quantidade de entradas

O cálculo realizado pela função de ativação do neurônio, diferentemente do cálculo da soma das entradas ponderadas por seus pesos, independe da quantidade de entradas, pois toda função de ativação recebe apenas um parâmetro, que é o resultado da soma ponderada das entradas e do bias do neurônio, e fornece apenas um resultado. Desta forma, é possível avaliar o tempo de execução de cada função de ativação apresentada pela Figura 3.17 da Seção 3.5. Além disso, dependendo da função de ativação, o uso de mais elementos lógicos ou de multiplicadores de 9-*bits* é necessário.

A Tabela 4.2 apresenta os tempos de execução e recursos adicionais necessários para os cálculos de cada uma das funções de ativação desenvolvidas. Os valores indicados nessa Tabela foram aferidos pelo *software* Quartus Prime.

Tabela 4.2: Tempo de execução e recursos necessários a cada uma das funções de ativação

Função	Tempo (ns)	Elementos lógicos	Multiplicadores
Degrau	7,974	0	0
Linear	8,604	0	0
Retificadora linear	13,358	31	0
Logística unipolar (1^a ordem)	20,131	54	0
Tangente hiperbólica (1^a ordem)	21,040	43	0
Logística unipolar (2^a ordem)	27,068	251	16
Tangente hiperbólica (2^a ordem)	29,712	273	16

A partir da Tabela 4.2, é possível perceber que, quanto maior a complexidade da função de ativação, maior é o tempo necessário a sua execução.

4.1.2 Neurônio recorrente

O modelo de neurônio recorrente desenvolvido e descrito pela Figura 3.19 da Seção 3.5, no FPGA utilizado, necessita dos recursos explicitados na Tabela 4.3. Para esse tipo de neurônio, a quantidade de entradas não é variada, uma vez que, na estrutura de uma rede Elman, cada neurônio recorrente recebe apenas uma entrada. Além disso, o tipo de função de ativação desses neurônios é sempre linear, como apresentado no Capítulo 2 - Fundamentação Teórica.

Tabela 4.3: Recursos físicos utilizados por um neurônio recorrente

Recurso	Quantidade utilizada
Elementos lógicos	32
Registradores	32
Pinos de entrada/saída	66
Multiplicadores de 9 *bits*	0

A quantidade de registradores (*flip-flops*) utilizados é 32, como mostra a Tabela 4.3, pois, uma vez que a entrada e a saída do neurônio são compostas por 32 *bits*, o neurônio deve armazenar valores de 32 *bits* até a transição da borda positiva de *clock*.

Além disso, o modelo de neurônio recorrente pode ser caracterizado temporalmente por meio dos valores expressos pela Tabela 4.4.

Tabela 4.4: Caracterização temporal de um neurônio recorrente

Variável temporal	Valor (ns)
setup time (tsu)	2,384
hold time (th)	−1,833
clock to output time (tco)	7,570

A Tabela 4.4 mostra que a entrada do neurônio deve ser apresentada e mantida constante 2,384 ns antes da transição positiva do *clock*. Além disso, a variável temporal *hold time* negativa indica que é necessário que a entrada do neurônio se mantenha constante por pelo menos 1,833 ns depois da transição de borda positiva do *clock*. Já o *clock to output time* indica que o tempo de propagação do sinal até a saída após a transição de *clock* é de 7,570 ns.

4.1.3 Aproximação das funções de ativação

As aproximações utilizadas para as funções logística e tangente hiperbólica foram avaliadas. Para cada uma dessas funções de ativação, há duas aproximações, como definido na Figura 3.17: uma aproximação por segmentos de reta, aproximação de 1^a ordem, e outra composta por segmentos de parábolas, aproximação de 2^a ordem. As aproximações das funções logística e tangente hiperbólica são apresentadas graficamente nas Figuras 4.2 e 4.3 respectivamente. Essas figuras também apresentam os gráficos da função a ser aproximada e o erro absoluto entre ambas as aproximações e a função correspondente.

Os erros mostrados nas Figuras 4.2 e 4.3 não consideram o uso de ponto fixo, apenas as funções de aproximação. Assim, todos os números utilizados foram representados em ponto flutuante no mesmo ambiente.

Percebe-se, pelos gráficos das Figuras 4.2 e 4.3, que a aproximação de 1^a ordem, por segmentos de retas, é a mais adequada no caso da função logística, pois seu erro é menor. Entretanto, para o caso da função tangente hiperbólica, a melhor aproximação é a realizada por funções de 2^a ordem, que foi utilizada neste trabalho em todos os testes em *hardware*.

Pode-se avaliar ainda a influência do uso do ponto fixo no cálculo da função de ativação. Assim, é aplicada a aproximação de 2^a ordem da função tangente hiperbólica sobre 600 valores x no intervalo $[-4, 4]$ descritos em 32 *bits* Q20, que é a mesma precisão utilizada para representar os valores internos à rede neural em *hardware*. A partir desse experimento, foi possível comparar o cálculo da aproximação da função de ativação feito em ponto fixo e feito em ponto flutuante.

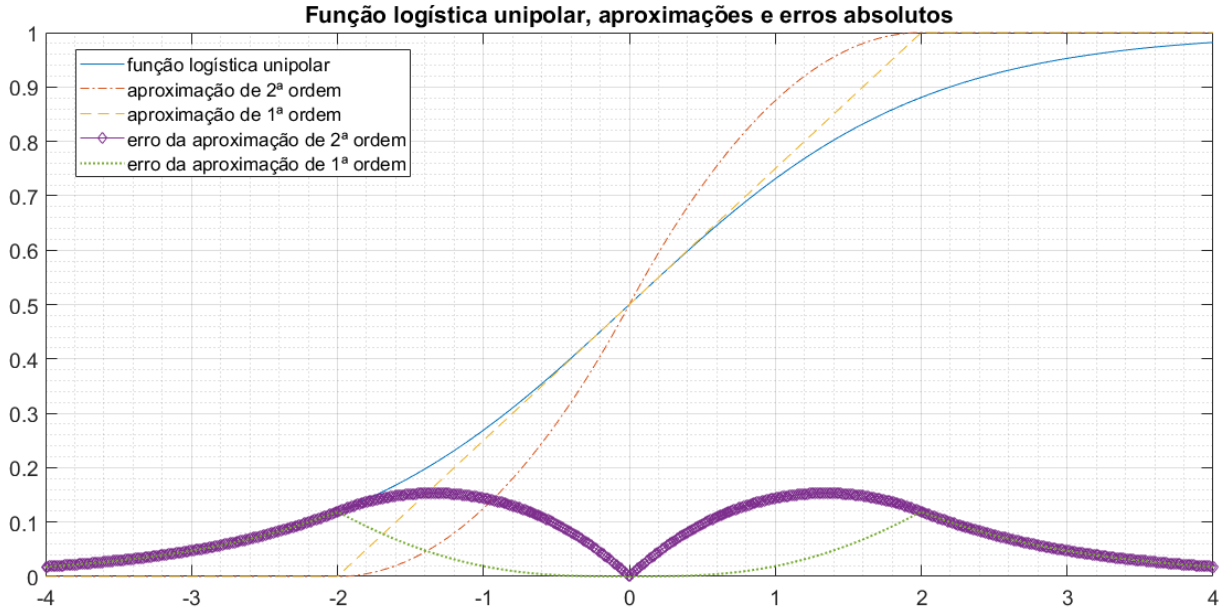

Figura 4.2: Aproximações para a função logística e erro absoluto das aproximações

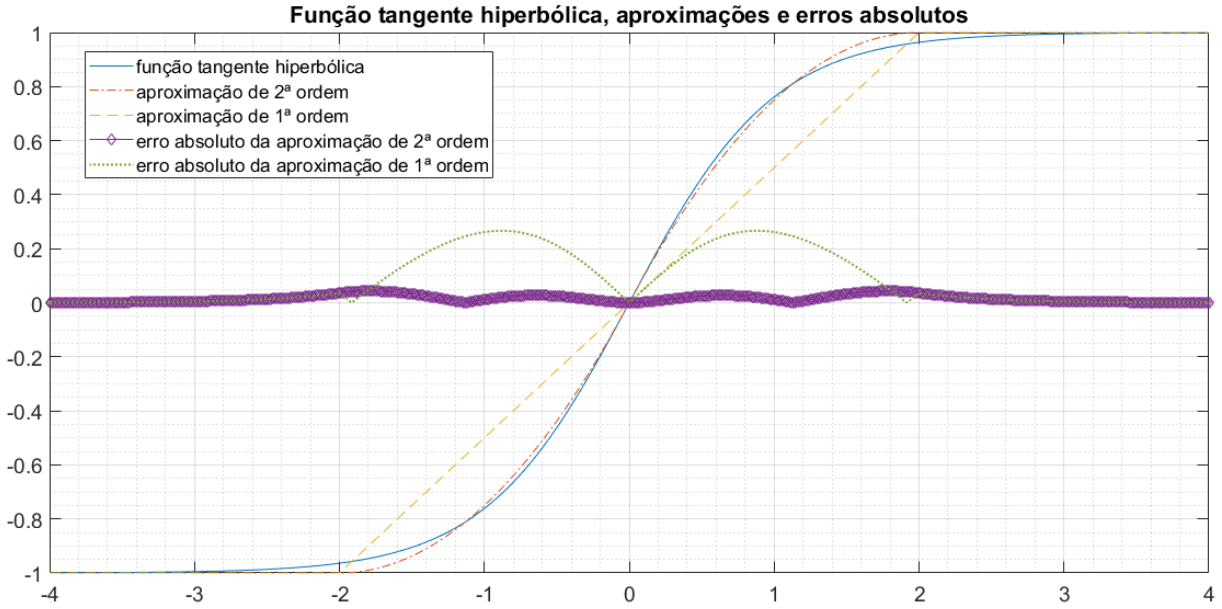

Figura 4.3: Aproximações para a função tangente hiperbólica e erro absoluto das aproximações

Essa comparação é indicada pelo gráfico de erro da Figura 4.4.

Assim, a partir das Figuras 4.3 e 4.4, percebe-se que o erro relacionado à aproximação da função de ativação do tipo tangente hiperbólica é maior que o erro relativo à representação em ponto fixo.

Figura 4.4: Erro relativo à representação por ponto fixo para a aproximação parabólica da função tangente hiperbólica

4.2 Caracterização dos sistemas de previsão

Nesta seção, são apresentadas as caracterizações para o sistema desenvolvido tanto em *hardware* quanto em *software*. Ou seja, são avaliados os gastos em termos de recursos físicos e os tempos de execução de cada um dos sistemas.

A rede neural presente no sistema é uma rede do tipo Elman com 12 neurônios na camada escondida, como descrito na Seção 3.5 do capítulo de Metodologia.

4.2.1 Sistema desenvolvido em *hardware*

O sistema desenvolvido no kit Altera DE2-115, representado pelo diagrama da Figura 3.3 e detalhado nas Seções 3.2 a 3.6 utiliza os recursos físicos expostos pela Tabela 4.5.

Tabela 4.5: Recursos utilizados pelo sistema desenvolvido em *hardware*

Recurso	Quantidade utilizada	Ocupação do FPGA (%)
Elementos lógicos	100.922	88
Registradores	6.460	-
Pinos de entrada/saída	108	20
Multiplicadores de 9 *bits*	532	100

Os recursos indicados na Tabela 4.5 foram adquiridos por meio da ferramenta de análise e

síntese do *software* Quartus Prime disponibilizado pela fabricante Intel. Essa ferramenta permite caracterizar cada um dos módulos utilizados no projeto em termos de ocupação dos recursos físicos do FPGA. Assim, os recursos utilizados pelo módulo da FFT são descritos pela Tabela 4.6.

Tabela 4.6: Recursos utilizados pelo módulo de FFT do sistema desenvolvido em *hardware*

Recurso	Quantidade utilizada	Ocupação do FPGA (%)
Elementos lógicos	4.822	4,2
Registradores	5.251	-
Multiplicadores de 9 *bits*	8	1,5

A partir da comparação entre as Tabelas 4.5 e 4.6, é possível concluir que o módulo de cálculo da FFT utiliza a maior parte dos registradores alocados para o sistema.

Os recursos físicos ocupados pelo módulo da rede neural utilizada (Elman com 12 neurônios na camada escondida) são detalhados pela Tabela 4.7.

Tabela 4.7: Recursos utilizados pelo módulo de rede neural do sistema desenvolvido em *hardware*

Recurso	Quantidade utilizada	Ocupação do FPGA (%)
Elementos lógicos	91.412	79,7
Registradores	384	-
Multiplicadores de 9 *bits*	520	97, 7

A partir da observação das Tabelas 4.5 e 4.7, é possível concluir que a rede neural é responsável pela utilização da maior parte dos elementos lógicos e multiplicadores de 9 *bits* alocados ao sistema desenvolvido.

Além disso, por meio do *software* Quartus Prime, é possível verificar, também, a alocação de recursos no FPGA. Essa alocação é descrita visualmente pela Figura 4.5

O gradiente de tons azuis na Figura 4.5 indica a utilização dos LABs, que são conjuntos de 16 células combinacionais e 16 células registradoras. Quanto mais escuro o tom de azul, maior é a ocupação desses componentes. Na Figura 4.5 há ainda 10 traços verticais que podem ser de dois tipos que se intercalam: verde claro com verde e lilás com branco. O primeiro tipo representa os blocos de memória do tipo M9K, sendo que o verde escuro indica a utilização do recurso e o verde claro a não utilização. Já o segundo tipo representa blocos do tipo DSP. As partes em cor preta da Figura 4.5 indicam o plano de fundo do *chip*, ou seja, partes em que não há componentes. Por fim, os elementos mais próximos às bordas do *chip* representam os pinos de entrada e saída do FPGA.

A partir da Figura 4.5, é possível perceber que os LABs são utilizados de maneira relativamente uniforme, uma vez que a maioria desses elementos estão marcados pela cor azul, sendo que os LABs com ocupação máxima de seus 32 elementos estão no lado esquerdo do *chip*. Além disso,

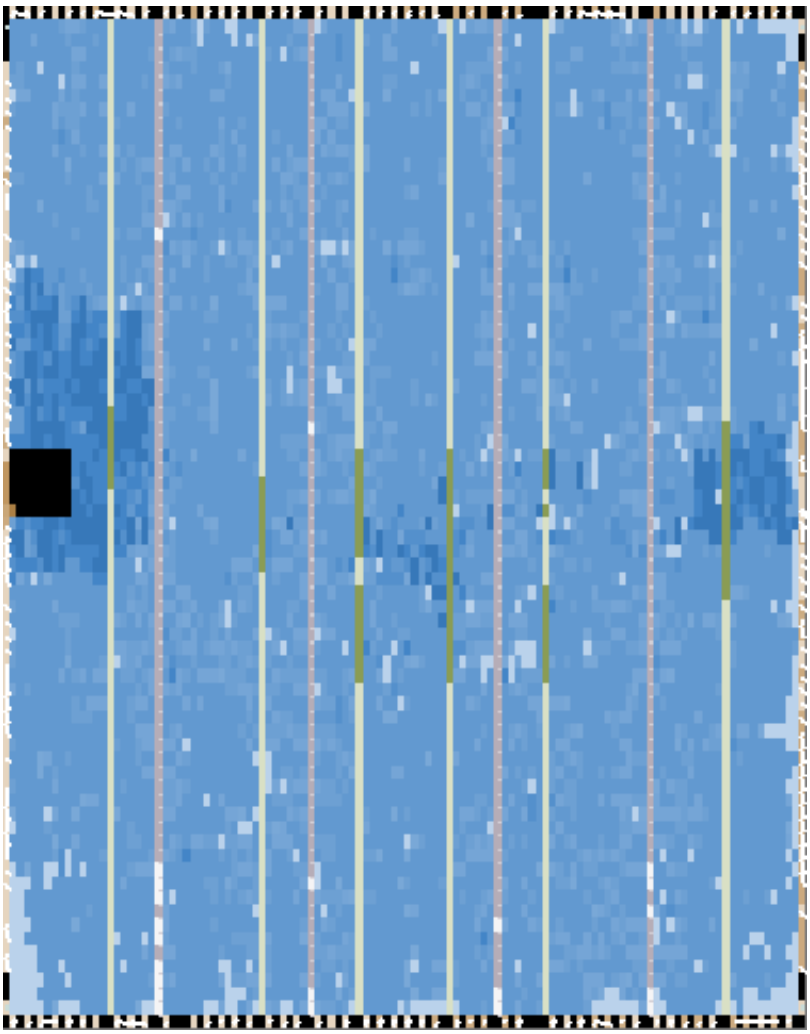

Figura 4.5: Mapa de alocação de recursos do FPGA

percebe-se que as memórias do tipo M9K não foram muito utilizadas, uma vez que os elementos indicados pela cor verde não são muitos, quando comparados às memórias M9K não ocupadas, e se concentram na porção central do *chip*.

O tempo de execução do sistema em *hardware* pode ser estimado por meio da análise da máquina de estados descrita na Seção 3.6. Para uma janela, composta por 128 amostras complexas do espectro eletromagnético, tem-se a geração de apenas um valor de energia, que, quando quantizado, torna-se o sinal de entrada para a rede neural. Assim, nesta situação, o tempo necessário desde o processamento das 128 amostras complexas até a geração da saída da rede neural equivale a $3 + 310 + 128 + 1 + 6 + 1 = 449$ ciclos de *clock*. Para o *clock* utilizado de 50MHz, tem-se que o tempo necessário a cada janela é de 8,98 μs.

Caso esse sistema fosse aplicado diretamente no FPGA presente no rádio USRP N210, considerando um *clock* também de 50MHz, o processamento realizado não seria executado em tempo real. Tendo em vista que o período de captura do rádio entre uma amostra complexa ($I(t)$ e $Q(t)$) e outra é de 0,04μs, são necessários $0{,}04\mu s \cdot 128 = 5{,}12\mu s$ para capturar os 128 valores complexos que compõem uma janela do sistema. Assim, para evitar a ociosidade do sistema, seria necessário

que a frequência de processamento do FPGA fosse aumentada para $449/5{,}12\mu s = 87{,}7$ MHz.

4.2.2 Sistema desenvolvido em *software*

Para os testes em *software*, o sistema da Figura 3.3 foi definido em duas linguagens de programação: Matlab e C++. O computador utilizado durante todos os testes possui processador Intel®Core™i7-6500U, frequência de *clock* de 2.50GHz e memória RAM de 8GB.

O teste realizado em ambas as definições do sistema em *software* consistiram na apresentação de 9 recortes de coletas, efetuadas por meio da metodologia apresentada na Seção 3.2, aos sistemas. Esses recortes podem ser observados na Figura 4.6.

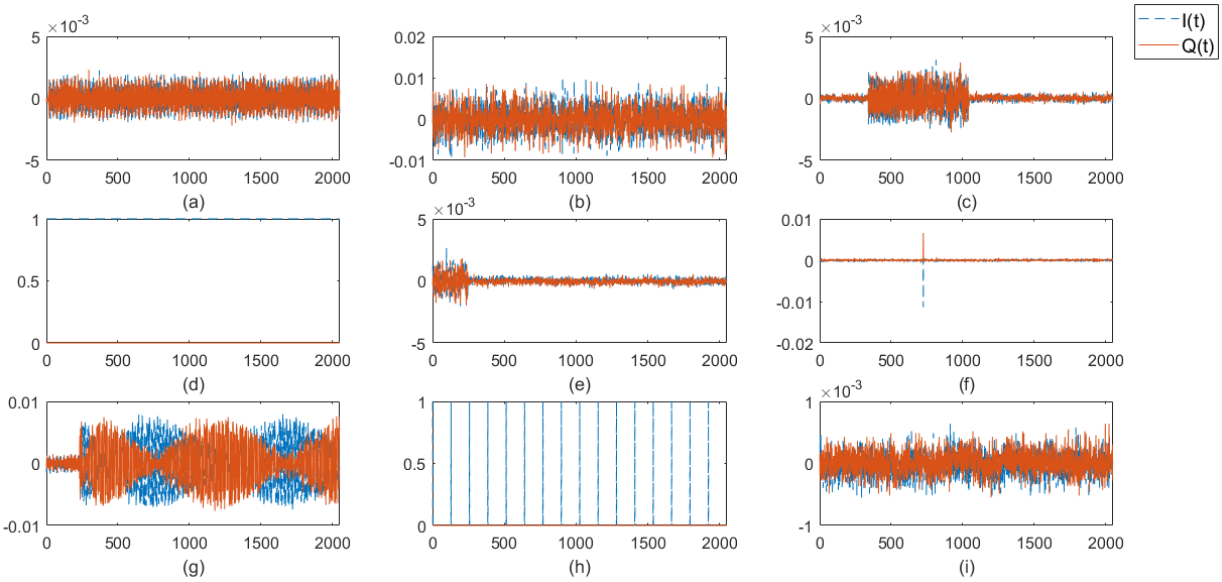

Figura 4.6: Recortes de coletas aplicados na entrada do sistema desenvolvido, cujo eixo y indica as magnitudes dos sinais $I(t)$ e $Q(t)$ e o eixo x o número referente à amostra complexa

Todos os 9 recortes utilizados contêm 2048 pares de sinais $I(t)$ e $Q(t)$, o que, para o tamanho de janela $N = 128$ utilizado, é equivalente a 16 janelas. Os recortes da Figura 4.6 foram tirados a partir de coletas capturadas em ambientes diversos, com exceção dos recortes indicados por (d) e (h), que representam uma entrada constante $I(t) = 1$ e uma entrada $I(t)$ pulso unitário na primeira amostra de cada uma das 16 janelas.

Cada um dos recortes foi apresentado aos sistemas 1000 vezes, resultando, assim, em 9000 execuções de cada um dos sistemas. Para cada uma dessas vezes, o tempo de execução de cada um dos sistemas foi registrado e uma média foi calculada sobre todos os 9000 valores de tempo. O resultado final desses cálculos é sumarizado na Tabela 4.8.

A partir da Tabela 4.8, é possível concluir que, em média, o sistema em C++ realiza o processo completo de processamento de sinal, quantização e aplicação da rede neural cerca de 7 ms mais rapidamente que o sistema desenvolvido em Matlab para um recorte com 2048 amostras.

Além disso, a partir do mesmo experimento com os 9 recortes definidos pela Figura 4.6, o

Tabela 4.8: Tempo médio para execuções do sistema em *software*

Linguagem	Tempo de execução (ms)
Matlab	9,41225
C++	2,11733

tempo de processamento médio da rede neural (desconsiderou-se o processamento dos sinais $I(t)$ e $Q(t)$) foi estimado. As estimativas encontradas para os sistemas em Matlab e em C++ são apresentadas pela Tabela 4.9.

Tabela 4.9: Tempo médio para execuções da rede neural definida em *software*

Linguagem	Tempo de execução (ms)
Matlab	7,2
C++	0,06256

Percebe-se, assim, pelos resultados das Tabelas 4.8 e 4.9, que a execução da rede neural em C++ ocupa, em média, apenas 2,95% do tempo total utilizado pelo sistema e que, a execução da rede em Matlab ocupa, em média, 76,50% do tempo total.

A diferença temporal observada na Tabela 4.8 pode-se apoiar no fato de que a linguagem utilizada em Matlab, por ser interpretada, requer um tempo maior para a interpretação e a execução de códigos, quando comparado a linguagens compiladas, tais como o C++ [56]. Além disso, o fato do Matlab definir estruturas de rede genéricas pode impactar na performance de algoritmos e sistemas que utilizem redes neurais nesse ambiente.

Entretanto, o resultado indicado pela Tabela 4.9 sugere que o processamento dos sinais de fase e de quadratura ocorre em menos tempo no Matlab, quando comparado com o ambiente em C++. Tendo em vista que, durante o processamento de sinal, a etapa mais custosa temporalmente é o cálculo da FFT, pode-se concluir que a função que realiza esse cálculo em Matlab é mais eficiente que a função em C++. Essa eficiência pode ser justificada pelo fato de que as variáveis em Matlab são definidas como matrizes e os algoritmos desenvolvidos nesse ambiente são otimizados para cálculos matriciais [57]. Assim, o cálculo da FFT definido na Seção 2.3, pode ser descrito por multiplicações e somas matriciais otimizadas no ambiente do Matlab.

4.2.3 Comparação entre sistemas

Pela análise dos resultados indicados pela Tabela 4.8, percebeu-se que o sistema descrito em C++ é mais rápido que o sistema em Matlab. Para o processamento completo de uma janela de tamanho $N = 128$, tem-se que, em C++, são necessários, em média, $2{,}11733/16 = 0{,}13233\ ms$ e, em Matlab, são requisitados $0{,}58826\ ms$. Os tempos de processamento de uma janela para os

sistemas desenvolvidos em *software* e em *hardware* são apresentados na Tabela 4.10.

Tabela 4.10: Comparação entre os tempos de execução dos sistemas para o processamento de uma janela de tamanho $N = 128$

Sistema	Tempo de execução (ms)
Matlab	0,58826
C++	0,13233
FPGA	0,00898

Por meio da comparação entre os resultados da Tabela 4.10, nota-se que a execução do sistema em *hardware*, para um *clock* de 50MHz, é 7,4 vezes mais rápido que a execução em C++, por exemplo. O tempo de execução reduzido em *hardware* justifica o uso do FPGA em vez de definições de sistemas em *software* incorporados ao rádio USRP. Na verdade, como observado na Seção 4.2.1, para desenvolver um sistema que opere na frequência máxima de amostragem f_s do rádio, seria necessário que o tempo de execução desse sistema fosse ainda menor que o valor encontrado de $8{,}98\mu s$.

4.3 *Testbench* dos sistemas de previsão

Nesta seção, a validação dos sistemas de previsão descritos em *hardware* e em *software* é apresentada. Para tanto, foram aplicados os mesmos 9 recortes de coletas apresentados na Figura 4.6 da Seção 4.2.

A partir desses 9 recortes, os sistemas desenvolvidos em Matlab e em C++ foram comparados. O erro entre os resultados dos dois sistemas foi calculado como uma simples subtração entre a saída do neurônio que indica oportunidade de transmissão do sistema definido em Matlab e a saída do neurônio correspondente do sistema definido em C++. O resultado dessa subtração é apresentado na Figura 4.7, em que a posição dos gráficos corresponde à posição dos 9 recortes de coletas definidos na Figura 4.6.

Pode-se afirmar, pela Figura 4.7, que a diferença entre o resultado do sistema em Matlab e em C++ é desprezível, uma vez que a saída desses sistemas varia dentro do intervalo $[0, 1]$ e o erro encontrado é da ordem de 10^{-16} para todos os recortes testados.

O erro da saída do neurônio que indica a não-oportunidade entre os dois sistemas definidos em *software* também é desprezível. O erro máximo entre os dois sistemas, encontrado ao longo de todos os 9 recortes, para as duas classes de neurônio é apresentado na Tabela 4.11.

Também a partir dos mesmos 9 recortes da Figura 4.6, o sistema desenvolvido em *hardware* foi comparado com o sistema definido em Matlab. A saída do neurônio que indica oportunidade para os dois sistemas avaliados sobre os 9 recortes é apresentada pela Figura 4.8, em que a posição dos gráficos corresponde à posição dos 9 recortes de coletas definidos na Figura 4.6.

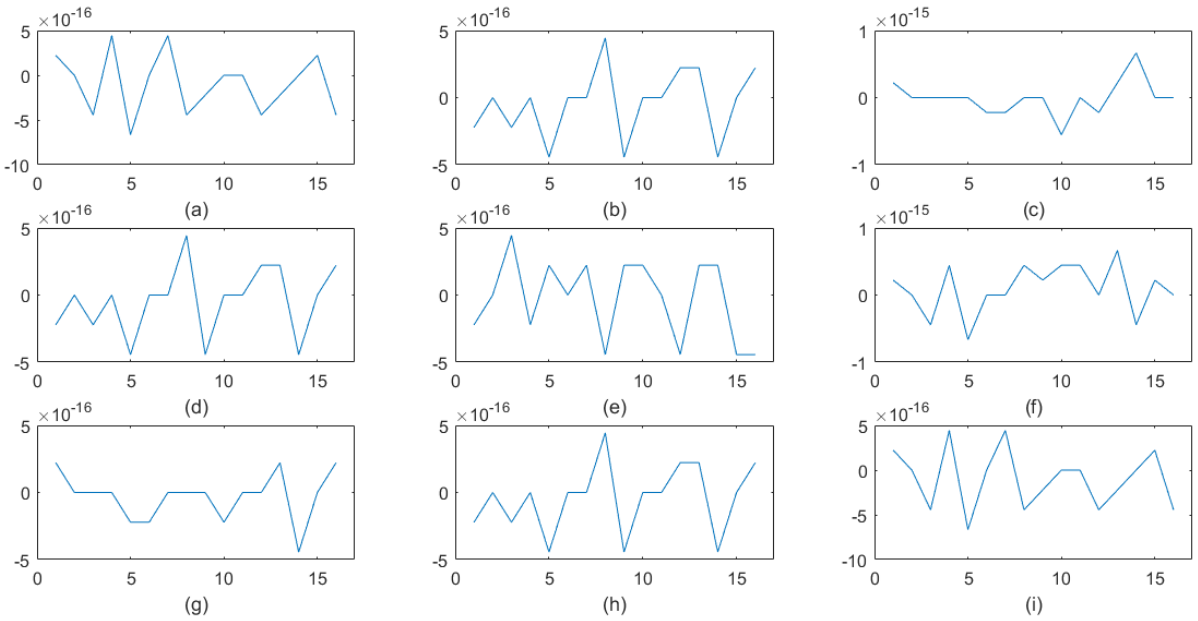

Figura 4.7: Subtração entre a saída do neurônio que indica oportunidade do sistema em Matlab e a do mesmo neurônio do sistema em C++ para os recortes definidos na Figura 4.6. O eixo y indica as magnitudes dos erros e o eixo x, o número da janela

Tabela 4.11: Erro máximo entre os dois sistemas definidos em *software* para as duas classes de neurônio

Classe do neurônio	Erro máximo
Não-oportunidade	$1{,}1102 \cdot 10^{-15}$
Oportunidade	$6{,}6613 \cdot 10^{-16}$

Percebe-se, pelos gráficos da Figura 4.8, que a saída do neurônio que indica oportunidade em *hardware* é bastante similar à saída do mesmo neurônio em Matlab. Mesmo no caso de maior discrepância entre os resultados (resultado indicado por (h)), a saída do neurônio em *hardware* nunca é tão diferente da saída em Matlab a ponto de alterar a previsão da rede. Os erros máximos, para cada classe de neurônio de saída, são descritos na Tabela 4.12.

Tabela 4.12: Erro máximo entre o sistema definido em *hardware* e o sistema definido em Matlab para as duas classes de neurônio

Classe do neurônio	Erro máximo
Não-oportunidade	0,1532
Oportunidade	0,2440

O maior erro encontrado é de 24,2% na classe de neurônio que indica a oportunidade e é

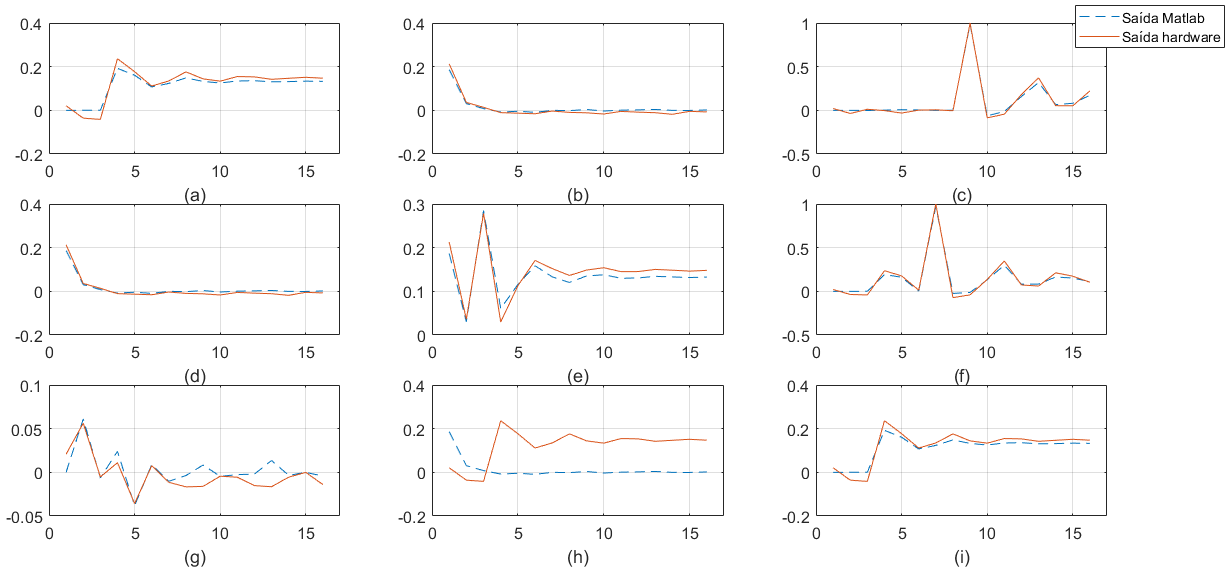

Figura 4.8: Saída do neurônio que indica oportunidade do sistema descrito em *hardware* e do mesmo neurônio do sistema descrito em Matlab para os recortes definidos na Figura 4.6. O eixo y indica as magnitudes das saídas dos neurônios e o eixo x é referente ao número da janela

relativo ao resultado indicado por (h) na Figura 4.8. Como visto na Seção 4.1, esse erro pode ser justificado como sendo um acúmulo de erros gerados, principalmente, pela aproximação da função de ativação utilizada em todos os neurônios das camadas escondida e de saída.

4.4 Adaptabilidade da rede neural

Redes do tipo Elman, com quantidades variadas de neurônios na camada escondida, treinadas com a variação Levenberg-Marquard do *backpropagation*, atingem resultados satisfatórios, com uma média de acerto de 81,7% [1] para o problema de identificação de oportunidades. Além disso, como desenvolvido em trabalhos anteriores [58], o sistema, quando implementado com adaptação dos pesos da rede por meio do mesmo algoritmo, alcançou uma precisão de 82,52% em ambientes ruidosos e 96,78% para ambientes sem interferências significativas.

Entretanto, como discutido na Seção 3.7, a variação Levenberg-Marquard do *backpropagation* é computacionalmente custosa. Além disso, por meio da Tabela 4.5 e da Figura 4.5 da Seção 4.2.1, nota-se que o sistema desenvolvido em *hardware*, mesmo sem o algoritmo de adaptação dos pesos, já utiliza todos os multiplicadores de 9 *bits* e 88% dos elementos lógicos do FPGA Cyclone IV.

Dessa forma, foi verificada a possibilidade de utilizar a variação do *backpropagation* gradiente descendente com momento para a adaptação dos pesos da rede definida em *hardware*. Para tanto, testes iniciais foram executados em *software* com o objetivo de definir um algoritmo adequado para permitir a adaptabilidade da rede neural de acordo com alterações das características estatísticas do canal ao longo do tempo.

Nesta seção serão apresentados os resultados obtidos após a aplicação da metodologia descrita

na Seção 3.7 para criar uma proposta de adaptação da rede em tempo real baseada no *backpropagation* gradiente descendente com momento. Para tanto, dois testes, que serão apresentados a seguir, foram conduzidos. O primeiro experimento partiu de um problema simples e teve como objetivo provar a validade do algoritmo de treino, ao qual a adaptabilidade da rede está fortemente conectada. Esse experimento procura garantir que redes treinadas com o sistema desenvolvido realmente são capazes de solucionar e generalizar o problema proposto a elas. O segundo experimento, por outro lado, tratou da aplicação do treino e do retreino sobre o conjunto de captura do espectro eletromagnético para treinar uma rede de previsão de oportunidades de transmissão.

4.4.1 Treino da rede para solucionar o problema do XOR

O experimento de treino de uma rede para solucionar o problema do ou-exclusivo (XOR) foi utilizado para validar o algoritmo de treino implementado. A partir de técnicas de *debug*, foi possível observar que o cálculo executado pelo sistema em C++ é o mesmo realizado pela função *adapt* do Matlab, que representa o treino de forma *online* do *backpropagation* clássico.

O problema do XOR consiste em apresentar a resposta do operador XOR aplicado sobre as entradas anterior e a atual, considerando que elas podem ser apenas 0 ou 1. Para solucioná-lo, uma rede recorrente do tipo Elman com uma entrada, 10 neurônios na camada escondida e um neurônio de saída foi treinada para 20 sequências de tamanhos variados aplicadas à entrada. O treino foi executado para 25 mil épocas, coeficiente de aprendizado igual a 0,1 e momento nulo. Depois de treinada, essa rede foi avaliada sobre 5 sequências de teste distintas daquelas apresentadas durante o treino. A saída da rede, para essas 5 sequências, foi então subtraída da resposta desejada, gerando o valor de erro observado pela Figura 4.9.

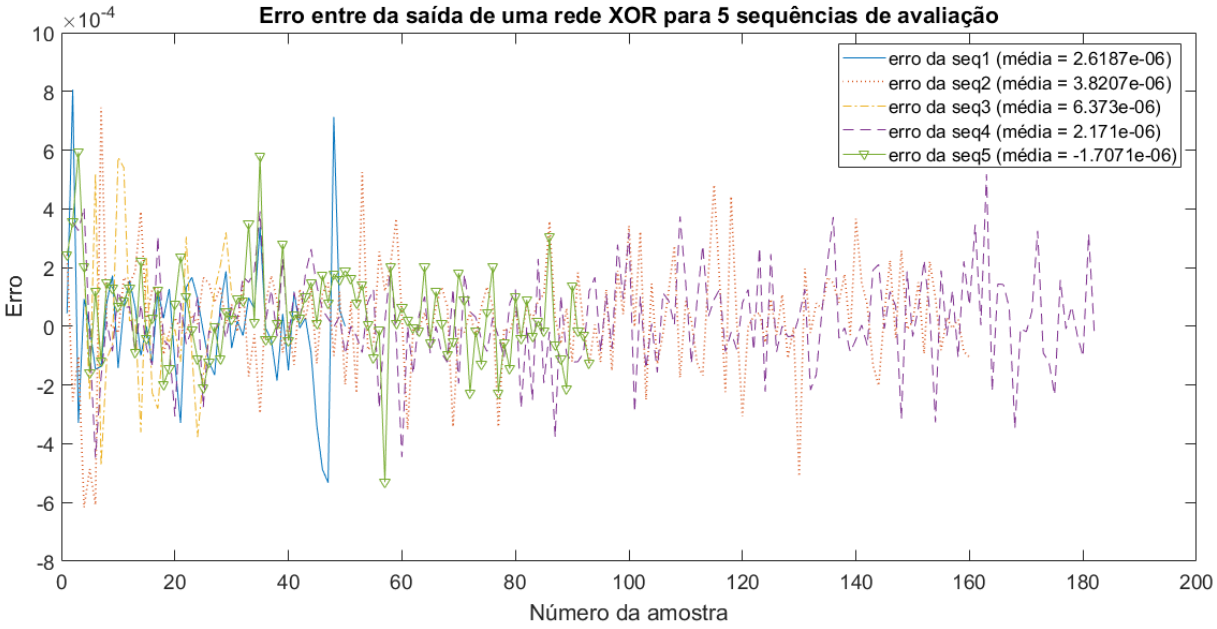

Figura 4.9: Exemplo de erro de rede treinada para o problema da XOR avaliada sob 5 sequências de teste

Como pode ser visto pela Figura 4.9, a diferença entre a resposta real da rede e a desejada é desprezível, uma vez que é da ordem de 10^{-4} e a saída da rede está contida no intervalo $[0, 1]$. Assim, pode-se dizer que a rede é capaz de executar a operação XOR satisfatoriamente para qualquer sequência formada por 0s e 1s.

4.4.2 Treino e adaptação da rede para o problema de previsão de oportunidades

Após o desenvolvimento dessa rede que soluciona o problema da XOR, uma rede para solucionar a identificação de oportunidades em um cenário bastante simples foi implementada. Neste problema de identificação, são apresentadas 17 sequências já quantizadas q_i de tamanhos variados retiradas de uma transmissão que ocorreu em um ambiente sem interferências. Assim, essas sequências representam as situações mais simples do problema de transmissão. A quantização das sequências ocorreu de forma externa ao sistema para simplificar ainda mais o problema proposto de acordo com o diagrama da Figura 3.2 apresentado e explicado na Seção 3.1. Essas sequências são, portanto, formadas por sinais 0 e 1, indicando a ausência e a presença de energia, respectivamente.

A rede proposta para solucionar esse problema é uma rede Elman com uma entrada, 20 neurônios na camada escondida e dois neurônios na camada de saída. Essa rede foi treinada com dois milhões de épocas, coeficiente de aprendizado 0,1, momento nulo e função de ativação tangente hiperbólica para ambas as camadas escondida e de saída. A rede resultante deste treino foi avaliada sobre as mesmas sequências utilizadas no treino, para as quais espera-se um acerto próximo de 100% quando comparado com a sequência desejada. Contudo, o erro de várias das 17 sequências, para os dois neurônios de saída, apresentou-se como mostra a Figura 4.10.

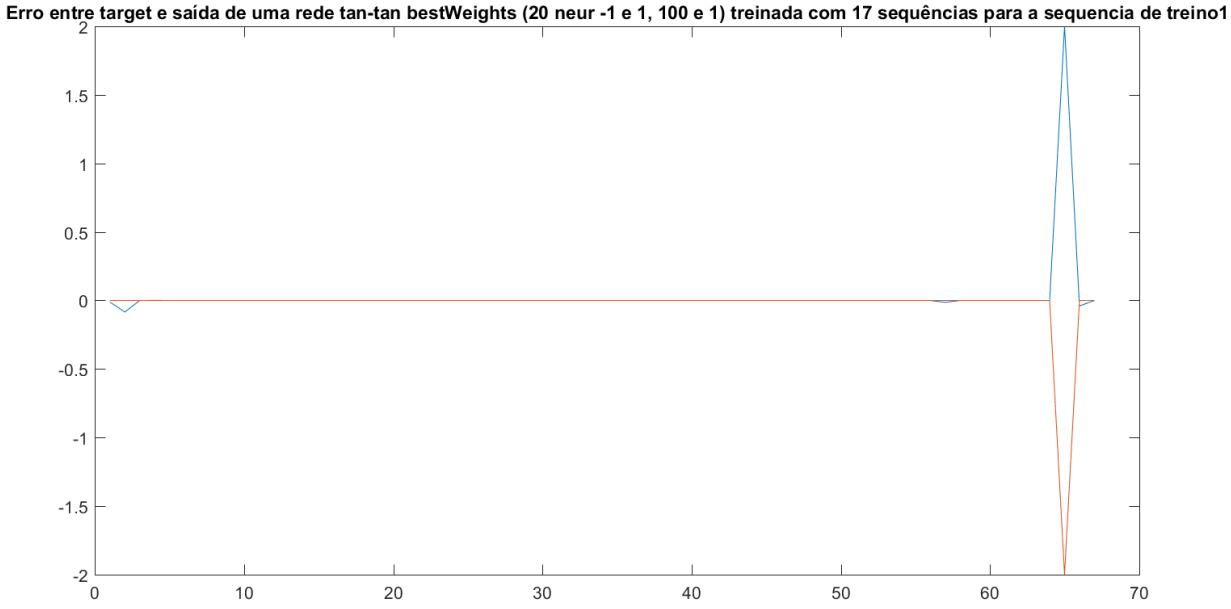

Figura 4.10: Exemplo de erro de rede treinada para sequências simples de transmissão

A partir da Figura 4.10, foi possível concluir que a rede erra a previsão de apenas uma amostra,

que, no caso das sequências simples, coincide com a única previsão da presença de oportunidade que ocorre ao longo da sequência. Percebe-se, assim, que a rede mantém a sua saída indicando sempre a não existência de oportunidade na maioria das sequências apresentadas. Portanto, pode-se concluir que não foi possível treinar a rede para esse problema simples com essa configuração de treino e de rede. O treino para esse problema foi repetido para redes com funções de ativação distintas, maior quantidade de neurônios na camada escondida, maior quantidade de iterações e diferentes valores de coeficiente de aprendizado e de momento. Entretanto, nenhuma das redes treinadas foi capaz de solucionar o problema proposto satisfatoriamente.

Como o algoritmo desenvolvido em C++ não foi capaz de treinar uma rede para o problema mais simples de identificação de oportunidade, conclui-se que provavelmente o algoritmo de *backpropagation online* encontrado na literatura não é capaz de solucionar o problema proposto. Portanto, deve-se considerar uma modificação do algoritmo para que funcione no modo *batch* ou que a estrutura da rede seja alterada para uma rede NARX em vez da Elman utilizada. Pode-se considerar, ainda, a modificação do algoritmo de treino e de adaptação para a variação Levenberg–Marquardt do *backpropagation* à custo de recursos computacionais. Essa última modificação foi realizada em um trabalho parcial [58], a partir do qual foi possível obter resultados preliminares satisfatórios em relação à adaptação da rede neural de forma *online* em *software*.

Entretanto, para o FPGA utilizado neste trabalho, a implementação da variação Levenberg–Marquardt do *backpropagation* não é possível, uma vez que esse algoritmo é relativamente complexo e 88% dos elementos lógicos do FPGA são utilizados apenas para o processamento dos sinais $I(t)$ e $Q(t)$, para o armazenamento de uma rede com 12 neurônios na camada escondida e para o cálculo *feedforward* dessa rede.

Capítulo 5

Conclusões

O uso crescente de dispositivos móveis com acesso à rede sem fio e a alocação estática do espectro eletromagnético tornaram a escassez do espectro uma preocupação entre cientistas e instituições reguladoras desse recurso.

Neste cenário, estudos relacionados à alocação dinâmica do espectro se tornaram cada vez mais comuns e têm o intuito de possibilitar um acesso melhor distribuído e mais justo do espectro. Entretanto, os desafios do desenvolvimento de sistemas capazes de promover o acesso dinâmico ao espectro são inúmeros, pois é necessário que eles funcionem em tempo real e tenham taxas de acerto próximas de 100% para evitar ao máximo colisões, interferências e perdas de pacotes durante a transmissão.

Dessa forma, este trabalho propôs o desenvolvimento de um sistema, em *hardware*, capaz de identificar oportunidades de transmissão em redes sem fio e de se adaptar de acordo com variações no uso do espectro. O sistema proposto foi baseado em redes neurais recorrentes do tipo Elman treinadas com a versão Levenberg-Marquardt do algoritmo *backpropagation*. Porém, para realizar a adaptação da rede, propôs-se o uso da versão gradiente descendente por ser mais simples e viável de ser implementada em *hardware* devido à limitação de recursos computacionais.

Neste trabalho, foi desenvolvido um sistema que realiza um processamento dos sinais de fase e de quadratura adquiridos pelo monitoramento do espectro eletromagnético na banda de frequência do canal 13 do padrão IEEE 802.11g, para gerar um sinal de energia, que, ao ser quantizado, é aplicado à rede neural, que prevê oportunidades de transmissão. Esse sistema foi implementado em *hardware* FPGA e em *software*. A partir de alguns experimentos, foi possível concluir que, em média, o sistema em *hardware* é 7,4 vezes mais rápido que o em *software*. Além disso, as previsões de oportunidade realizadas, tanto em *hardware* quanto em *software*, corresponderam às expectativas.

Para um conjunto com 9 recortes de capturas, o sistema em *hardware* teve um erro máximo de 24,4% em relação ao resultado esperado, que, apesar de parecer uma quantidade considerável, não interferiu na previsão final da rede.

Além disso, no ambiente em *software*, foram realizados alguns experimentos com o algoritmo

de *backpropagation* com gradiente descendente para possibilitar a adaptação da rede neural. Entretanto, os resultados desses testes mostraram que, apesar desse algoritmo treinar e adaptar redes para solucionar o problema simples da XOR, ele não é adequado para treinar e adaptar uma rede Elman para executar a previsão de oportunidades.

Pode-se concluir, a partir dos resultados obtidos neste trabalho, que os objetivos específicos propostos foram cumpridos, uma vez que o sistema desenvolvido em *hardware* indica corretamente oportunidades de transmissão em todos os cenários testados. Entretanto, o objetivo geral do projeto foi parcialmente atingido, pois o algoritmo proposto para ajustar os pesos da rede de forma *online* não se mostrou adequado ao problema de previsão de oportunidades. Em um FPGA com recursos computacionais suficientes para o desenvolvimento da variação Levenberg–Marquardt do *backpropagation*, a adaptação da rede neural pode ser incorporada ao sistema com sucesso.

Essa adaptação bem sucedida foi alcançada em um sistema em *software*, que faz parte dos resultados parciais deste trabalho, publicados na *International Joint Conference on Neural Networks* (IJCNN) de 2018 [58].

5.1 Trabalhos Futuros

A metodologia proposta não foi totalmente executada, o que abre espaço para melhorias e expansão do escopo deste trabalho. Algumas das sugestões para trabalhos futuros são:

- **Utilização de outras redes recorrentes, como a NARX e a LSTM.** A adoção de outras redes recorrentes, além de permitir comparar a performance entre vários modelos, pode possibilitar, também, uma melhor adaptabilidade da rede.
- **Testes com outros algoritmos de treino.** Espera-se que algoritmos com adaptação de pesos do tipo *batch* e/ou com uma melhor performance de treino possibilitem a adaptabilidade da rede de forma *online*.
- **Aplicação do sistema no FPGA do USRP.** Neste trabalho, foi utilizado um kit de desenvolvimento para testar o sistema descrito em *hardware*. A implementação no FPGA do USRP permite a execução *online* do sistema no seu ambiente real de utilização.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] FERREIRA, P. A. L.; FERNANDES, S. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação), *Previsão de oportunidades de acesso utilizando redes neurais artificiais recorrentes*. 2016.

[2] HAYKIN, S. *Neural Networks A Comprehensive Foundation*. 2. ed. [S.l.]: Prentice Hall International Inc., 1999.

[3] RASCHKA, S. *Activation Functions for Artificial Neural Networks*. 2018. `http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/`. Online; Acesso: 2018-12-03.

[4] WIKIPEDIA. *Fast Fourier Transform*. 2018. `https://en.wikipedia.org/wiki/Fast_Fourier_transform`. Online; Acesso: 2018-12-03.

[5] SERGIYENKO, A.; UZENKOV, O. *Pipelined FFT/IFFT 128 points processor :: Overview*. 2010. `https://opencores.org/project/pipelined_fft_128`. Online; Acesso: 2018-10-15.

[6] RANK, J. *The wearable computing market: a global analysis*. [S.l.], 2012.

[7] ASHTON, K. et al. That 'internet of things' thing. *RFID journal*, Jun, v. 22, n. 7, p. 97–114, 2009.

[8] ESTADAO. *Até o fim de 2017, Brasil terá um smartphone por habitante, diz FGV*. 2017. `https://link.estadao.com.br/noticias/gadget,ate-o-fim-de-2017-brasil-tera-um-smartphone-por-habitante\-diz-pesquisa-da-fgv,70001744407`. Online; Acesso: 2018-07-20.

[9] KOLODZY, P.; AVOIDANCE, I. Spectrum policy task force. *Federal Commun. Comm., Washington, DC, Rep. ET Docket*, v. 40, n. 4, p. 147–158, 2002.

[10] TELECOMMUNICATIONS, N.; ADMINISTRATION, I. *United States Frequency allocation Chart*. 2016. `https://www.ntia.doc.gov/page/2011/united-states-frequency-allocation-chart`. Online; Acesso: 2018-07-20.

[11] ANATEL. *Plano de atribuição, destinação e distribuição de frequências no Brasil*. 1. ed. [S.l.]: Anatel, 2017.

[12] MCHENRY, M. A. et al. Chicago spectrum occupancy measurements & analysis and a long-term studies proposal. In: *Proceedings of the First International Workshop on Technology and Policy for Accessing Spectrum*. New York, NY, USA: ACM, 2006. (TAPAS '06). ISBN 1-59593-510-X. Disponível em: <http://doi.acm.org/10.1145/1234388.1234389>.

[13] AKYILDIZ, I. F. et al. Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey. *Computer networks*, Elsevier, v. 50, n. 13, p. 2127–2159, 2006.

[14] ZHAO, Q.; SADLER, B. M. Dynamic spectrum access: Signal processing, networking, and regulatory policy. *arXiv preprint cs/0609149*, 2006.

[15] COMMISSION, F. C. *Promoting More Efficient Use of Spectrum Through Dynamic Spectrum Use Technologies*. 2010. Notice of Inquiry; FCC-10-198.

[16] PRETZ, K. *Overcoming Spectrum Scarcity*. 2012. `http://theinstitute.ieee.org/technology-topics/smart-technology/overcoming-spectrum-scarcity`. Online; Acesso: 2018-07-20.

[17] LIANG, Y. C. et al. Cognitive radio networking and communications: an overview. *IEEE Transactions on Vehicular Technology*, v. 60, n. 7, p. 3386–3407, Sept 2011. ISSN 0018-9545.

[18] MATHWORKS. *Deep Learning Toolbox*. 2018. `https://www.mathworks.com/products/deep-learning.html`. Online; Acesso: 2018-11-20.

[19] SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding Machine Learning: From Theory to Algorithms*. [S.l.]: Cambridge University Press, 2014.

[20] FAUSETT, L. *Fundamentals of neural networks: architectures, algorithms, and applications*. [S.l.: s.n.].

[21] MINSKY, M.; PAPERT, S. Perceptrons,"mit press, cambridge, ma. 1969.

[22] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, Springer, v. 2, n. 4, p. 303–314, 1989.

[23] RESEARCH, U. of Illinois at Urbana-Champaign. Center for S.; DEVELOPMENT; CYBENKO, G. *Continuous valued neural networks with two hidden layers are sufficient*. [S.l.: s.n.], 1988.

[24] HAGAN, M. T.; MENHAJ, M. B. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, v. 5, n. 6, p. 989–993, Nov 1994. ISSN 1045-9227.

[25] SETIONO, R.; HUI, L. C. K. Use of a quasi-newton method in a feedforward neural network construction algorithm. *IEEE Transactions on Neural Networks*, v. 6, n. 1, p. 273–277, Jan 1995. ISSN 1045-9227.

[26] COCHOCKI, A.; UNBEHAUEN, R. *Neural Networks for Optimization and Signal Processing*. 1. ed. [S.l.]: John Wiley and Sons Inc., 1993.

[27] GRAVES, A.; SCHMIDHUBER, J. Offline handwriting recognition with multidimensional recurrent neural networks. In: KOLLER, D. et al. (Ed.). *Advances in Neural Information Processing Systems 21*. Curran Associates, Inc., 2009. p. 545–552. Disponível em: <http://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks.pdf>.

[28] GRAVES, A.; MOHAMED, A. r.; HINTON, G. Speech recognition with deep recurrent neural networks. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. [S.l.: s.n.], 2013. p. 6645–6649. ISSN 1520-6149.

[29] SUTSKEVER, I.; MARTENS, J.; HINTON, G. E. Generating text with recurrent neural networks. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. [S.l.: s.n.], 2011. p. 1017–1024.

[30] ELMAN, J. L. Finding structure in time. *Cognitive science*, Wiley Online Library, v. 14, n. 2, p. 179–211, 1990.

[31] JORDAN, M. I. *Serial order: A parallel distributed processing approach*. [S.l.], 1986.

[32] XIE, H.; TANG, H.; LIAO, Y.-H. Time series prediction based on narx neural networks: An advanced approach. In: IEEE. *Machine Learning and Cybernetics, 2009 International Conference on*. [S.l.], 2009. v. 3, p. 1275–1279.

[33] LI, W.; ZHANG, D.; XU, Z. Palmprint identification by fourier transform. *International Journal of Pattern Recognition and Artificial Intelligence*, World Scientific, v. 16, n. 04, p. 417–432, 2002.

[34] LIM, J. S. Two-dimensional signal and image processing. *Englewood Cliffs, NJ, Prentice Hall, 1990, 710 p.*, 1990.

[35] SANDIFER, E. How euler did it. *Mathematical Association of America*, 2005.

[36] COOLEY, J. W.; TUKEY, J. W. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, JSTOR, v. 19, n. 90, p. 297–301, 1965.

[37] BLUESTEIN, L. A linear filtering approach to the computation of discrete fourier transform. *IEEE Transactions on Audio and Electroacoustics*, IEEE, v. 18, n. 4, p. 451–455, 1970.

[38] RADER, C. M. Discrete fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, IEEE, v. 56, n. 6, p. 1107–1108, 1968.

[39] YAVNE, R. An economical method for calculating the discrete fourier transform. In: ACM. *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. [S.l.], 1968. p. 115–125.

[40] NULL, L.; LOBUR, J. *The essentials of computer organization and architecture*. [S.l.]: Jones & Bartlett Publishers, 2014.

[41] APPLICATION Note 33: Fixed Point Arithmetic on the ARM. [S.l.]: Advanced RISC Machines Ltd (ARM), 1996. ARM DAI 0033A.

[42] KEHTARNAVAZ, N.; KIM, N. *Digital signal processing system-level design using LabVIEW*. [S.l.]: Elsevier, 2011.

[43] SOCIETY, I. C. Ieee standard definitions and concepts for dynamic spectrum access: Terminology relating to emerging wireless networks, system functionality, and spectrum management. *IEEE Std 1900.1-2008*, p. 1–62, Oct 2008.

[44] MAI, T. *Software Defined Radio*. 2012. `https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_sdr.html`. Online; Acesso: 2018-07-17.

[45] RESEARCH, E. *USRP N210*. 2012. `https://www.ettus.com/product/details/UN210-KIT`. Online; Acesso: 2018-07-17.

[46] INC., X. *XtremeDSP Development Platform: Spartan-3A DSP 3400A Edition*. [S.l.], 2008.

[47] ALTERA. 2018. `https://www.altera.com/`. Online; Acesso: 2018-07-17.

[48] NEWSROOM, I. *Intel Completes Acquisition of Altera*. 2015. `https://newsroom.intel.com/news-releases/intel-completes-acquisition-of-altera/`. Online; Acesso: 2018-11-26.

[49] XILINX. 2018. `https://www.xilinx.com/`. Online; Acesso: 2018-07-17.

[50] DILLIEN, P. *And the winner of best FPGA of 2016 is ...* 2017. `https://www.eetimes.com/author.asp?section_id=36&doc_id=1331443`. Online; Acesso: 2018-07-17.

[51] XILINX. *What is an FPGA?* 2018. `https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html`. Online; Acesso: 2018-07-17.

[52] INC., T. T. *DE2-115 User Manual*. [S.l.], 2010.

[53] KUROSE, R. *Computer Networking: A Top-Down Approach*. 6. ed. [S.l.]: Pearson, 2013.

[54] HAMMING, R. W. *Digital filters*. [S.l.]: Courier Corporation, 1998.

[55] INTEL. *Quartus*. 2018. `https://www.intel.com/content/www/us/en/programmable/downloads/download-center.html`. Online; Acesso: 2018-10-25.

[56] CENTER, I. K. *Compiled versus interpreted languages*. 2010. `https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zappldev/zappldev_85.htm`. Online; Acesso: 2018-12-03.

[57] MATHWORKS. *Vectorization*. 2018. `http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html`. Online; Acesso: 2018-12-03.

[58] FERNANDES, S. S. et al. An adaptive recurrent neural network model dedicated to opportunistic communication in wireless networks. *2018 International Joint Conference on Neural Networks (IJCNN)*, p. 01–08, 2018.