



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
DEPARTAMENTO DE TELEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Renan Gomes Vieira

**Processo de Prototipação de um Classificador Neural Embarcado
para Detecção de Falhas em Motores de Indução Trifásicos**

Fortaleza – CE

14 de dezembro de 2016

Renan Gomes Vieira

Processo de Prototipação de um Classificador Neural Embarcado para Detecção de Falhas em Motores de Indução Trifásicos

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Engenharia de Software

Orientador: Prof. Dr. Elias Teodoro da Silva Júnior

Coorientador: Prof. Dr. Cláudio Marques de Sá Medeiros

Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Programa de Pós-Graduação em Ciência da Computação (PPGCC)

Fortaleza – CE

14 de dezembro de 2016

Catálogo na fonte: Erika Cristiny Brandão F. Barbosa – CRB – 1099-3

V658p Vieira, Renan Gomes.
 Processo de prototipação de um classificador neural embarcado
 para detecção de falhas motores de indução trifásicos / Renan
 Gomes Vieira. - Fortaleza: IFCE, 2016.

114 f.

Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da
Computação – PPGCC do Instituto Federal de Educação, Ciência e
Tecnologia do Ceará – IFCE, 2016.

Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Elias Teodoro da Silva Júnior

1. COMPUTAÇÃO - DISSERTAÇÃO. 2. ENGENHARIA DE
SOFTWARE. 3. REDES NEURAIIS. 4. SISTEMAS EMBARCADOS I.
Título.

CDD 005.1

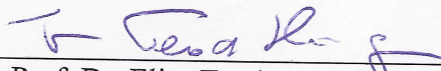
RENAN GOMES VIEIRA

Processo de Prototipação de um Classificador Neural Embarcado para Detecção de Falhas em Motores de Indução Trifásicos

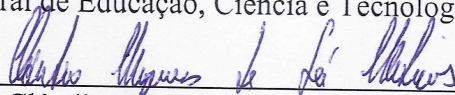
Dissertação submetida à Coordenação do Curso de Pós-graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação, área de concentração Ciência da Computação.

Aprovada em 14 / 12 / 2016.

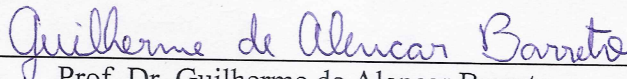
BANCA EXAMINADORA



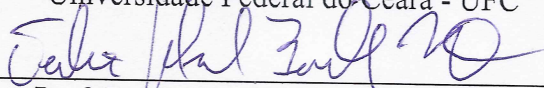
Prof. Dr. Elias Teodoro da Silva Júnior (Orientador)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE



Prof. Dr. Cláudio Marques de Sá Medeiros (Co-orientador)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE



Prof. Dr. Guilherme de Alencar Barreto
Universidade Federal do Ceará - UFC



Prof. Dr. Tobias Rafael Fernandes Neto
Universidade Federal do Ceará - UFC

Agradecimentos

Aos meus pais Dinha e Valmir, por sempre apoiarem minhas escolhas e incentivarem meu crescimento intelectual. Eu nada seria sem seus ensinamentos e exemplos de força, dedicação e resiliência. Estendo os agradecimentos aos meus irmãos, Lucas e Julyanna pelo momentos e palavras de incentivo.

Ao Robson que foi primo, irmão e amigo ao me auxiliar de todas as formas que estavam ao seu alcance.

Aos outros familiares que me apoiaram de forma direta e indireta como Tia Nenza, Tia Gracinha, Tia Neidinha e os primos Messias e Juliana.

À Beatriz pelo carinho, amizade, compreensão e palavras de apoio durante a maior parte deste trajeto.

Ao Professor Carlos Hairon pela confiança depositada durante o projeto Vendor S2, no LIT, o qual disponibilizou a bolsa de estudos durante meu mestrado. Agradeço também pelos conhecimentos compartilhados. Uma lembrança aos colegas de projeto Samir, Luan e Victor. Imensamente grato pela oportunidade.

Aos meus orientadores Elias Teodoro e Claudio Sá pelos imensuráveis conhecimentos adquiridos através das aulas, reuniões e orientações. São exemplos de ética, responsabilidade e dedicação que pretendo replicar e disseminar. Sinto-me privilegiado pela oportunidade de desenvolver esse trabalho com os senhores. Obrigado pela confiança.

Ao João e Rebeca que implementaram, respectivamente, grande parte dos códigos no FPGA e no processo de aquisição dos dados no dsPIC.

Aos colegas de mestrado, aos companheiros do LAMP/LES Lucas, Madson, Marcelo, Murilo, Renato e Victor com os quais compartilhei conhecimentos, alegrias, dificuldades, cafés e merendas. Em especial, Madson sempre prestativo a ajudar e dividir conhecimento e experiências de pesquisa; Marcelo e Hanna que me abrigaram em João Pessoa para minha participação no SBESC'16.

À todos os professores do PPGCC do IFCE pelas experiências vivenciadas; Ao programa e ao colegiado por prover meios para minha participação no WCCT'16.

Ao amigos: Dennis Sávio pelas diversas conversas, momentos de descontração, e pela ajuda para que minha participação no WCCT'16 fosse possível; Anderson, Kennedy e Milaynne pelo constante apoio e palavras de amizade.

A todos, meu sincero muito obrigado!

Resumo

Existem diversas aplicações embarcadas responsáveis por suprir demandas da sociedade. Há também vários estudos e aplicações de classificadores neurais para um grande conjunto de problemas em campo. Entretanto, poucas avaliações existem sobre como embarcar o classificador, assim como quais plataformas são as mais adequadas. Muitas vezes, há a necessidade de efetuar alguns ajustes nos classificadores ou nos dados a serem processados pelos mesmos para que seja possível o desenvolvimento de uma solução embarcada aplicável. Ao considerar limitações nas plataformas embarcadas durante a etapa de treinamento, o projetista do classificador pode tornar o processo de prototipação embarcada mais fácil e viável. Esta dissertação descreve o processo de desenvolvimento de um protótipo embarcado de um classificador neural treinado e validado por simulação para diagnosticar falha em motores de indução trifásicos, especificamente o curto-circuito entre espiras estatóricas. Uma série de adaptações nos sinais que compõem o conjunto de dados original utilizado para treinar e testar a rede neural são feitas com o intuito de possibilitar que a rede seja embarcada em diferentes arquiteturas. As mudanças nos dados ocorrem na taxa de aquisição dos sinais, período de aquisição e precisão numérica nos valores de entrada e pesos da rede neural artificial. Também são implementadas e comparadas duas implementações de tangente hiperbólica em Linguagem de Descrição de Hardware VHSIC (*VHSIC Hardware Description Language*, VHDL) para utilizar como função de ativação nos neurônios da rede neural artificial em um Arranjo de Portas Programável em Campo (*Field Programmable Gate Array*, FPGA). Como contribuição à detecção de falha entre espiras, um classificador capaz de detectar a evolução da intensidade da falha, a partir da condição normal, também é avaliado. Após definição e validação das propostas de classificadores e adaptações dos sinais que compõem o conjunto de dados para treinamento das redes, códigos em C e VHDL do classificador são embarcados respectivamente em um Controlador Digital de Sinais (*Digital Signal Controler*, DSC) e em um FPGA. Os resultados são avaliados através de algumas métricas (tempo de execução, memória ocupada, acurácia) a partir das quais são discutidas as principais vantagens e características de cada implementação. Os métodos utilizados empiricamente para desenvolvimento do protótipo são condensados e apresentados, servindo como proposta de um guia a ser seguido em problemas similares. Os resultados obtidos nas plataformas embarcadas são muito próximos obtidos no ambiente simulado em computador, validando os protótipos.

Palavras-chave: Sistemas Embarcados. Engenharia de Software. Redes Neurais. MLP. Curto circuito entre espiras. DSC. FPGA.

Abstract

There are several embedded applications responsible for supplying the society demands. There are several studies of neural classifiers applications for a wide range of problems in field. However, a few evaluations exists on how to embed a classifier, and which platforms are most appropriate to do it. Often, there is a necessity to perform some adjustments in the classifiers or in the data to be processed by the classifiers to be possible the development of an embedded solution. When considering limitations in embended plataforms during the training stage, the designer of the classifier can make the prototyping process easier and more viable. This work describes the process of developing an embedded prototype with a trained and validated neural classifier by simulation to identify faults in induction motors, specifically the interturn winding stator short circuit. A number of adaptations in the signals that compose the original data set used to train and test the neural network are performed in order to enable the network to be embedded in different architectures. The data changes occur in the signal acquisition rate, acquisition period and numerical accuracy in the input values and weights of the artificial neural network. Two differents hyperbolic tangent are implemented and compared in VHSIC Hardware Description Language (VHDL) to be used as activation function in the neurons of the artificial neural network in a Field Programmable Gate Array (FPGA). As a contribution to interturn fault detection, it is proposed a classifier which is able to detect the evolution of the fault. Codes in C and VHDL of the neural classifier are embedded, respectively, on a DSC and a FPGA. The results are evaluated through some metrics (execution time, memory footprint, accuracy) and the main advantages and characteristics of each implementation are discussed. The empirical methods used to develop the prototype are condensed and presented, serving as a proposal for a guide to be followed in similar problems. The results obtained in the embedded platforms are very close to the results of the computer simulated environment, which validates the prototypes.

Keywords: Embedded Systems. Software Engineering. Neural Networks. MLP. Interturn Winding Stator Short Circuit. Microcontroller. FPGA.

Lista de ilustrações

Figura 1 – Representação genérica de um perceptron	25
Figura 2 – Representação genérica de uma MLP com uma camada escondida . . .	26
Figura 3 – Etapas seguidas para desenvolvendo de um classificador neural	28
Figura 4 – Diagrama de caixa comparando a acurácia de diferentes classificadores projetados.	38
Figura 5 – Motor rebobinado acoplado ao freio de Foucault para aquisição do conjunto de dados	43
Figura 6 – Processo de aquisição das amostras do motor para composição do conjunto de dados	43
Figura 7 – Detalhamento das amostras - Número por fase de coleta, nível de carga e frequência de operação	44
Figura 8 – Leitura da corrente durante 0,4 segundos a 10kHz	45
Figura 9 – Valores da fundamental e das componentes de frequência obtidas após FFT	46
Figura 10 – Diagrama de composição do conjunto de dados	47
Figura 11 – Diagrama de caixa com os valores dos 50 treinamentos de cada projeto de classificador	50
Figura 12 – Novos conjuntos de dados com taxa de amostragem reduzida - Representação Gráfica	54
Figura 13 – Diagrama de caixa dos classificadores treinados com amostras coletadas em diferentes taxas de amostragem	55
Figura 14 – Intervalos das componentes visíveis em diferentes taxas de amostragem	56
Figura 15 – Acurácia Média para classificador de 3 classes por resolução no domínio da frequência	59
Figura 16 – Impacto da redução do período de aquisição nos valores das componentes de frequência	60
Figura 17 – Acurácia média dos classificadores treinados com conjuntos criados a partir de menores intervalos de coleta dos sinais	61
Figura 18 – Implementação de tangente hiperbólica utilizando retas	63
Figura 19 – Tangente hiperbólica aproximada através de 2 retas	64
Figura 20 – Erro da aproximação linear em comparação a original entre valores 0 e 8	65
Figura 21 – Comparativo entre solução 2 e tangente hiperbólica original	67
Figura 22 – Representação do protótipo embarcado para diagnóstico <i>on-line</i>	72
Figura 23 – Esquemático de um neurônio - Camada oculta	78
Figura 24 – Esquemático de um neurônio - Camada de saída	78

Figura 25 – Esquemático da rede neural implementada em VHDL com os compo- nentes mapeados	79
Figura 26 – EQM dos resultados do FPGA em comparação aos obtidos no Matlab .	81
Figura 27 – Processo para desenvolvimento de um classificador neural embarcado .	82
Figura 28 – Representação genérica de uma MLP com uma camada escondida . . .	96
Figura 29 – Ilustração da regra de parada antecipada baseada na validação cruzada	101

Lista de tabelas

Tabela 1 – Percentual dos principais componentes responsáveis por falhas em MIT	23
Tabela 2 – Parâmetros do modelo para RNA	48
Tabela 3 – Quantidade de amostras nos conjuntos de Treinamento, Validação e Teste para treinamento das RNAS nos diferentes projetos	49
Tabela 4 – Resultados obtidos nos treinamentos dos 2 projetos de classificadores .	51
Tabela 5 – Informações dos novos conjuntos de dados com taxa de amostragem reduzida	54
Tabela 6 – Resultados obtidos pelos classificadores treinados com o conjunto original (10 kHz) testando amostras coletadas em diferentes taxas de amostragem	55
Tabela 7 – Resultados das validações dos conjuntos de amostras de menor período de aquisição em classificadores treinados com os dados aquisitados a 10 kHz durante 10 segundos	57
Tabela 8 – Resultados obtidos com os novos classificadores treinados com conjuntos criados com intervalos menores de aquisição	58
Tabela 9 – Informações dos novos conjuntos de dados criados a partir de menores intervalos de coleta das amostras	61
Tabela 10 – Resultados dos treinamentos efetuados com o conjunto a 1 kHz durante duas coletas de 4 segundos	62
Tabela 11 – Equações das retas para aproximação da tangente hiperbólica	64
Tabela 12 – Representação da RALUT com suas respectivas entradas e saídas . . .	66
Tabela 13 – Comparativo dos resultados dos classificadores neurais com diferentes implementações de tangente hiperbólica	67
Tabela 14 – Resultados obtidos com diferentes precisões numéricas nos pesos dos classificadores	68
Tabela 15 – Resultados obtidos com diferentes precisões numéricas nas entradas dos classificadores	69
Tabela 16 – Resultados obtidos com diferentes precisões numéricas nas entradas e pesos do classificadores	69
Tabela 17 – Tempo de Execução	74
Tabela 18 – Quantidade de memória necessária por rotina de classificação do protótipo - DSC	74
Tabela 19 – Comparativo de acurácia: MATLAB <i>versus</i> dsPIC	75
Tabela 20 – Precisão dos dados no dsPIC	76
Tabela 21 – Recursos necessários por componente da RNA no FPGA	80
Tabela 22 – Comparativo dos resultados obtidos no FPGA e Matlab	80

Tabela 23 – Comparativo das saídas em 5 amostras: Matlab \times FPGA	80
Tabela 24 – Resultados médios obtidos de 30 realizações para o classificador binário	103
Tabela 25 – Resultados médios obtidos de 30 realizações para o classificador ternário	103
Tabela 26 – Resultados médios obtidos de 30 realizações para o classificador ternário com interpretação dos resultados como binário	104
Tabela 27 – Resultados dos classificadores com diferentes taxas de rejeição	105
Tabela 28 – Quantidade de amostras nos conjuntos de Treinamento, Validação e Teste para treinamento das RNAS nos classificadores binários e ternários	106
Tabela 29 – Resultados médios obtidos de 30 realizações para o classificador binário com adição de novos dados de conjunto normal	107
Tabela 30 – Resultados médios obtidos de 30 realizações para o classificador ternário com adição de novos dados de conjunto normal	107
Tabela 31 – Resultados médios obtidos de 30 realizações para o classificador ternário com interpretação dos resultados como binário	108

Lista de abreviaturas e siglas

ABS	Anti-lock Break System
ADALINE	Adaptive Linear Neuron
ADC	Analog-to-Digital Converter
AI	Alta Impedância
ASIC	Application Specific Integrated Circuits
ASSP	Application-Specific Standard Product
BI	Baixa Impedância
BP	Back-Propagation
CLB	Configurable Logic Blocks
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
cv	Cavalo-vapor
DSC	Digital Signal Controller
DSP	Digital Signal Processing
ELM	Extreme Learning Machine
EQM	Erro Quadrático Médio
FPGA	Field Programmable Gate Arrays
LCD	Liquid Crystal Display
LMA	Levenberg-Marquardt
LS-SVM	Least Squares Support Vector Machine
LUT	Look-Up Table
MCU	Microcontrolador
MIPS	Milhões de Instruções por Segundo

MIT	Motor de Indução Trifásico
MLM	Minimal Learning Machine
MLP	MultiLayer Perceptron
MNIST	Mixed National Institute of Standards and Technology
NNEP	Neural-Network-based Embedded system design Pattern
OPF	Optimum-Path Forest
RALUT	Range Addressable Look-Up Table
RBF	Radial Basis Function
RISC	Reduced Instruction Set Computer
RNA	Redes Neural Artificial
RNN	Recurrent Neural Network
SD	Secure Digital
SIRENS	Simple Reconfigurable Neural hardware Structure
SNN	Spiking Neural Networks
SoC	System-on-a-Chip
SOM	Self-Organizing Map
SRAM	Static Random Access Memory
SVM	Support Vector Machine
VHDL	VHSIC Hardware Description Language

Lista de símbolos

\mathbf{x}	Vetor de entrada qualquer
$u_i^{(h)}$	Ativação do i -ésimo neurônio da camada escondida na iteração t .
w_{ij}	Peso sináptico que conecta x_j ao i -ésimo neurônio.
$x_j(t)$	Componente qualquer do vetor de entrada $x(t)$.
$\theta_i^{(h)}$	Limiar do i -ésimo neurônio da camada escondida.
$\theta_k^{(o)}$	Limiar do k -ésimo neurônio da camada de saída.
Q	Número de neurônios da camada escondida.
P	Dimensão do vetor de entrada, sem o limiar.
M	Número de neurônios da camada de saída.
$y_i^{(h)}(t)$	Saída do i -ésimo neurônio da camada escondida na iteração t .
φ_i	Função de ativação do i -ésimo neurônio da camada escondida.
$y_k^{(o)}(t)$	Saída do i -ésimo neurônio da camada de saída na iteração t .
φ_k	Função de ativação do i -ésimo neurônio da camada de saída.
$u_k^{(o)}$	Ativação do i -ésimo neurônio da camada de saída.
$u_k^{(o)}(t)$	Ativação do i -ésimo neurônio da camada de saída na iteração t .
m_{ki}	Peso que conecta o i -ésimo neurônio da camada escondida ao k -ésimo neurônio da camada de saída.
$e_k^{(o)}(t)$	Valor do erro gerado por cada neurônio de saída no momento t .
$d_k(t)$	Valor desejado para a saída do k -ésimo neurônio da camada de saída.
ε	Erro quadrático médio.
J	Função objetivo, ou função custo.
$\delta_k^{(o)}(t)$	Gradiente local do neurônio k da camada de saída.
$\delta_i^{(h)}(t)$	Gradiente local do neurônio i da camada oculta.

$e_i^{(h)}(t)$	Sinal de erro retropropagado ou projetado para o i-ésimo neurônio da camada escondida.
η	Taxa de aprendizado
ξ	Termo de momento
r	Resolução no domínio da frequência
f_{max}	Frequência de Nyquist
N_p	Número de components obtidas no domínio da frequência
N_s	Número de pontos aqisitados

Sumário

1	INTRODUÇÃO	17
1.1	Motivação	19
1.2	Objetivos	20
1.2.1	Objetivo Geral	20
1.2.2	Objetivos Específicos	20
1.3	Produção Científica	21
1.4	Organização geral do texto	21
2	FUNDAMENTAÇÃO TEÓRICA	23
2.1	Curto-circuito entre espiras em motores de indução trifásico acionados por conversor de frequência	23
2.2	MultiLayer Perceptron (MLP)	24
2.2.1	Treinamento e Validação da MLP	27
2.3	Sistemas Embarcados e Redes neurais embarcadas	30
2.3.1	Conceitos Básicos	30
2.3.2	Arquitetura Geral	31
2.3.3	FPGA	32
2.3.4	DSP	33
2.3.5	Redes Neurais Embarcadas	33
2.4	Conclusão	35
3	TRABALHOS RELACIONADOS	36
3.1	Estado da Arte na detecção de Curto-circuito entre espiras utilizando redes neurais	36
3.1.1	Estado da Arte de aplicações de redes neurais embarcadas	39
3.2	Conclusão	41
4	PROCESSO DE AQUISIÇÃO DOS DADOS DE FALHA DE MOTOR E TREINAMENTO DOS CLASSIFICADORES	42
4.1	Descrição do processo de aquisição dos dados	42
4.2	Conjunto de Dados	46
4.3	Desenvolvimento dos Classificadores Neurais	48
4.4	Treinamentos Iniciais	48
4.5	Conclusão	50

5	ADEQUAÇÕES DO CONJUNTO DE DADOS E DO CLASSIFICADOR NEURAL PARA PROTOTIPAÇÃO DOS PROJETOS	52
5.1	Impacto da taxa de amostragem sobre a acurácia do classificador .	53
5.2	Impacto do período de aquisição dos dados sobre a acurácia do classificador	56
5.3	Discussão acerca das adaptações da captura dos dados	60
5.4	Alternativas de função de ativação para hardwares dedicados	62
5.4.1	Tangente hiperbólica baseada em aproximação de funções (Solução 1) . . .	63
5.4.2	Tangente hiperbólica baseada em aproximação de funções e RALUT (Solução 2)	63
5.4.3	Comparativo de acurácia das redes neurais utilizando as tangentes hiperbólicas alternativas	66
5.5	Impacto da precisão numérica nos pesos sinápticos e entradas da rede neural	67
5.6	Conclusão	69
6	IMPLEMENTAÇÃO DOS CLASSIFICADORES NEURAI EM PLATAFORMAS EMBARCADAS	71
6.1	Etapas executadas pelo protótipo proposto	71
6.2	Processo de prototipação em um DSC	71
6.3	Processo de prototipação em um FPGA	76
6.4	Métodos e Técnicas para prototipação de classificadores neurais em plataformas embarcadas	82
6.5	Conclusão	85
7	CONCLUSÃO	86
7.1	Trabalhos futuros	87
	REFERÊNCIAS	89
	APÊNDICE A – O ALGORITMO DE TREINAMENTO <i>BACKPROPAGATION</i>	96
A.1	Termo de momento	100
A.2	Validação Cruzada	100
	APÊNDICE B – TREINAMENTO DE NOVOS MODELOS NEURAI PARA O NOVO CONJUNTO DE DADOS .	102
B.1	Alteração do Número de Neurônios na camada oculta	102
B.1.1	Criação de novas amostras para o conjunto normal	106

APÊNDICE C – CÓDIGO DA REDE NEURAL NA LINGUAGEM	
C PARA EXECUÇÃO NO DSC	109

1 Introdução

Sistemas embarcados estão presentes em grande parte de equipamentos e produtos utilizados atualmente. Poucas vezes percebidos como computadores, esses sistemas estão presentes em telecomunicações, veículos, controle industrial, brinquedos e diversas outras categorias de uma extensa lista de produtos que já estão na rotina das pessoas e indústrias (MARWEDEL, 2011, pág. 1). Essas aplicações só se tornaram possíveis graças a evolução dos circuitos integrados que possibilitaram o desenvolvimento de componentes cada vez mais rápidos, menores e complexos (OLIVEIRA; ANDRADE, 2006, pág. 24).

A computação se beneficiou de forma significativa ao atingir esse estado de miniaturização e complexidade. Há muitas vezes a necessidade de computar ou processar informações em locais remotos (como dentro de indústrias, torres de telecomunicações ou veículos) (MARWEDEL, 2011, págs. 1-4) e não é prático, nem viável, ao avaliar métricas como tamanho do produto, custo, velocidade de processamento e consumo energético, a utilização de computadores desenvolvidos para propósitos gerais. Nesse cenário, surge uma série de soluções como microcontroladores (WOLF, 2008, pág. 3), hardwares reprogramáveis, como o Arranjo de Portas Programável em Campo (*Field Programmable Gate Arrays*, FPGA) e Dispositivo Lógico Complexo Programável (*Complex Programmable Logic Device*, CPLD) (MARWEDEL, 2011, pág. 152), Processador Digital de Sinais (*Digital Signal Processing*, DSP) (WOLF, 2008, pág. 52) e processadores desenvolvidos para aplicações específicas, como os Circuitos Integrados de Aplicação Específica (*Application Specific Integrated Circuits*, ASICs) e o Padrão de Produto para Aplicação Específica (*Application-Specific Standard Product*, ASSP) (MARWEDEL, 2011, pág. 135) como soluções para diversas demandas.

A heterogeneidade desse tipo de sistema expande-se à medida que há uma gama de possibilidades que esse tipo de produto pode atender. É essa diversificação que define umas das principais características dos sistemas embarcados: que eles são planejados e desenvolvidos para execução de uma ou mais tarefas específicas, utilizando-se inclusive dos requisitos exigidos do produto como forma de otimização do mesmo. Como exemplo, uma câmera filmadora só é necessária para aplicações que demandam algum tipo de processamento de imagens, e isso pode acarretar um maior consumo energético. Logo, no desenvolvimento do produto deve-se pensar em maneiras de suprir esse consumo, seja economizando em processamento e armazenamento, aumentando a autonomia da bateria (caso a aplicação não tenha fonte contínua de energia) ou quaisquer outras soluções que sejam viáveis a fim de atender os requisitos de prazo, custo ou até mesmo de requisitos como peso, tamanho e *design*.

Vistas as diversas aplicações citadas acima, é difícil estabelecer metodologias que funcionem para todos os tipos de sistemas embarcados, uma vez que há várias possibilidades de combinação de *hardware* e *software* nos diferentes produtos. Pesquisas desenvolvidas, por exemplo, por Kaisti et al. (2013) apontam o uso de metodologias ágeis como uma solução, assim como literatura especializada com padrões para aplicações com restrições em tempo real (DOUGLASS, 2008) e o uso de métodos já consolidados na engenharia de *software* se mostram aplicáveis e eficientes para o desenvolvimento dos sistemas embarcados.

Os *softwares* que são embarcados nesses *hardwares* dedicados são uma outra parte relevante dessa categoria de sistemas. Alinhado com o objetivo dos sistemas embarcados de solucionar problemas práticos e em campo, diversos algoritmos de Inteligência Computacional têm sido utilizados para solucionar os mais diversos problemas. Contudo, em busca de técnicas cada vez mais eficientes de aprendizado, junto a crescente capacidade de processamento e armazenamento dos computadores de propósito geral, tais algoritmos têm exigido cada vez mais processamento e espaço na memória para armazenar os aprendizados adquiridos. Alguns dos algoritmos mais utilizados em trabalhos contemporâneos da área de aprendizado de máquina como Floresta de Caminhos Ótimos (*Optimum-Path Forest*, OPF) (PAPA; FALCÃO, 2009), Máquina de Vetores de Suporte (*Support Vector Machine*, SVM) (HAYKIN, 2007, pag. 349) e Máquina de Vetores de Suporte por Mínimos Quadrados (*Least-Squares Support Vector Machine*, LS-SVM) (SUYKENS; VANDEWALLE, 1999) demandam mais recursos computacionais do que outros algoritmos mais simples disponíveis. Isso acaba indo de encontro a diversas restrições comumente relacionadas aos sistemas embarcados. Ao mesmo tempo, isso dificulta o processo de embarcar tais algoritmos em plataformas mais simples e de baixo custo, algo comumente desejável quando se procura desenvolver um produto dessa categoria.

Outro obstáculo que pode ser encontrado no desenvolvimento desse tipo de classificador, é a negligência de certas etapas como aquisição, pré e pós processamento dos dados ao se desenvolver esse tipo de algoritmo fora da arquitetura alvo. Algumas vezes, os classificadores são desenvolvidos sem ter em mente o produto final. No decorrer do desenvolvimento de um produto, principalmente nas etapas de design e implementação do protótipo, essas podem ser etapas que exigem recursos não antes considerados durante o treinamento dos classificadores. O resultado pode ser o desenvolvimento de um classificador que exige dados que não podem ser aquiridos em plataformas embarcadas ou que exigem muitos recursos computacionais, o que pode inviabilizar ou dificultar o desenvolvimento do projeto.

Um algoritmo com sua eficiência comprovada em diversas aplicações é a Perceptron Multicamadas (*Multilayer Perceptron*) ou, simplesmente, MLP. A MLP é uma rede neural capaz de lidar com problemas não lineares, abrangendo problemas de regressão, classificação e aproximação de funções. Nesse contexto de aplicações em campo e sistemas embarcados,

a rede MLP se mostra uma promissora solução na detecção de curto-circuito entre espiras estatóricas de motores de indução trifásicos acionados por conversor de frequência como descrito no trabalho de Oliveira (2014). Conforme indicado em Coelho et al. (2014) a rede MLP obtém bons resultados de classificação, com um menor custo computacional comparado a outros classificadores testados, como SVM e LS-SVM.

Com isso apresentado, esse trabalho baseia-se na pesquisa de Oliveira (2014), na qual o autor descreve a emulação de falhas e a coleta de dados do motor em curto-circuito. Após essa etapa, o mesmo autor define os parâmetros para o modelo neural através de tentativa e erro. Esta dissertação, a partir do classificador neural definido por Oliveira (2014), descreve as adaptações efetuadas no conjunto de dados e na rede neural artificial para prototipação embarcada. Os impactos dessas mudanças na acurácia do modelo são discutidos. Esta dissertação também discute brevemente uma nova forma de classificação dos padrões, avaliando outra forma mais detalhada de diagnóstico da falha presente no motor. Com o protótipo desenvolvido e validado, os métodos utilizados são condensados com intuito de funcionar como um conjunto de métodos a ser seguido para solução de problemas similares.

1.1 Motivação

As motivações do trabalho são detalhadas a seguir, em três tópicos.

Em relação à engenharia de *software* para sistemas embarcados, sendo essa a principal motivação do trabalho, busca-se descrever os métodos utilizados para embarcar uma rede neural definida em algum ambiente que não seja a plataforma alvo, onde não se leva em consideração as limitações que podem ser encontradas ao tentar embarcar o produto final. Existem diversas aplicações embarcadas assim como diversos estudos e aplicações de classificadores neurais para um grande conjunto de problemas em campo. Entretanto, poucas avaliações existem sobre como embarcar o classificador, assim como quais plataformas são as mais adequadas. Muitas vezes, há a necessidade de efetuar alguns ajustes nos classificadores ou nos dados a serem processados pelos classificadores para que seja possível o desenvolvimento de uma solução embarcada aplicável. Observa-se também, que etapas como pré-processamento e aquisição de dados, muitas vezes, são negligenciadas em termos de descrição em publicações similares. Ao considerar limitações nas plataformas embarcadas, o projetista do classificador pode tornar o processo de prototipação embarcada mais fácil e viável. Os métodos utilizados para prototipação são descritos com o intuito de serem replicados para problemas similares.

Em relação à contribuição para um produto funcional, a falha das espiras estatóricas em motores de indução é a segunda causa mais comum de falha nesse tipo de máquina elétrica (ALBRECHT et al., 1986, pág. 43). O estresse submetido ao bobinamento

estatístico do motor, pode chegar a ser dez vezes maior quando acionado por conversores de frequência (KAUFHOLD et al., 2002, págs. 27-28) em comparação aos que não são acionados, o que aumenta ainda mais a chance de ocorrerem curtos-circuitos. Geralmente, conversores de frequência já contém algum tipo de sistema embarcado em seu interior com pelo menos um sensor de corrente. Ao provar que o diagnóstico pode ser feito em uma plataforma embarcada, é possível que o classificador também seja inserido no sistema de comando e controle do conversor de frequência e aproveitar-se do *hardware* já estabelecido. Ao detectar o início da falha, o conversor pode não acionar a máquina antes da falha se agravar, poupando o custo ao evitar um curto-circuito mais severo em termos financeiros, de tempo de reparo e de parada de produção.

Já no quesito relacionado ao diagnóstico das falhas, todas as investigações anteriores a essa que utilizam-se do mesmo conjunto de dados, lidam com uma classificação binária, tentando diagnosticar o motor como em condições normais ou em condição inicial de falha. Neste trabalho, avalia-se um classificador que também identifique o agravamento da falha.

1.2 Objetivos

Segue uma descrição dos objetivo geral e específicos dessa dissertação:

1.2.1 Objetivo Geral

- Descrever o processo e métodos utilizados para embarcar um classificador neural em diferentes plataformas para detecção de curto-circuito entre espiras de motores de indução trifásicos acionados por conversor de frequência, comparando certas métricas como memória necessária, tempo de execução e acurácia do classificador.

1.2.2 Objetivos Específicos

- Avaliar o classificador neural capaz de identificar a evolução do curto-circuito a partir da condição normal que atenda as restrições de engenharia das plataformas embarcadas.
- Avaliar o impacto dos diversos parâmetros (taxa de amostragem de aquisição dos sinais, precisão numérica, função de ativação) sobre a acurácia do classificador e sobre os custos computacionais necessários no sistema embarcado, definindo valores que sejam aplicáveis em plataformas embarcadas de baixo custo e representativas para aprendizado do classificador.
- Adaptar o modelo neural definido para possibilitar o processo de desenvolvimento do sistema embarcado.

- Embarcar o classificador em um DSC e em um FPGA, descrevendo o processo das implementações.

1.3 Produção Científica

Durante o desenvolvimento desta dissertação, os seguintes artigos científicos foram publicados:

- Renan G. Vieira, Cláudio M. S. Medeiros, Elias T. Silva JR; Classification and sensitivity analysis to detect fault in induction motors using an MLP network; International Joint Conference on Neural Networks, WCCI '16, Vancouver, Canadá.
- Renan G. Vieira, Rebeca G. C. Cunha; Cláudio M. S. Medeiros, Elias T. Silva JR; Embedding a neural classifier to detect faults in a three-phase induction motor; VI Brazilian Symposium on Computing Systems Engineering, SBESC '16, João Pessoa, Brasil.

1.4 Organização geral do texto

No Capítulo 2, como fundamentação teórica, são apresentados os conceitos básicos utilizados no presente trabalho. Inicialmente, o problema de curto-circuito entre espiras estatóricas é apresentado. No segundo momento, a rede MLP é explicada, detalhando os principais conceitos que foram utilizados nesse trabalho. Na terceira parte, conceitos de sistemas embarcados são explorados, descrevendo também as principais plataformas para desenvolvimento.

É descrito no Capítulo 3 o estado da arte da detecção de curto-circuito utilizando algoritmos de inteligência artificial. Também é apresentado o estado da arte de redes neurais embarcadas em diferentes plataformas.

O Capítulo 4 descreve o processo de aquisição dos dados. O conjunto de dados também é explicado assim como duas possíveis formas de classificação do mesmo. O restante do capítulo detalha os treinamentos iniciais dos classificadores, descrevendo o processo e discutindo os resultados.

No Capítulo 5 o processo de adaptação do conjunto de dados no qual se reduz a taxa de amostragem e o período de aquisição são avaliados, onde se verifica também o impacto de tais modificações no modelo neural. O impacto das precisões numéricas dos pesos e entradas da rede neural na acurácia dos classificadores também é explorado. E, por fim, são implementadas e avaliadas duas funções de ativação alternativas para implementação em FPGA.

No Capítulo 6 os classificadores adaptados são embarcados em um FPGA e um Controlador Digital de Sinais (*Digital Signal Controller*, DSC), onde as métricas de avaliação são exibidas e discutidas. Ao fim, os métodos utilizados para embarcar o classificador são apresentados de forma generalizada, gerando um guia capaz de auxiliar projetos que visam embarcar algoritmos similares.

O Capítulo 7 finaliza o trabalho, onde se apresenta as conclusões da pesquisa e trabalhos futuros, discutindo as contribuições assim como as limitações encontradas.

2 Fundamentação Teórica

Neste capítulo são explorados os conceitos fundamentais utilizados nesse trabalho, limitando-se a abranger o que é mais pertinente ao entendimento dos experimentos efetuados.

2.1 Curto-circuito entre espiras em motores de indução trifásico acionados por conversor de frequência

Motores de indução trifásicos (MITs) têm um papel fundamental nas mais diversas aplicações industriais devido principalmente a sua robustez, simplicidade e eficiência (GHATE; DUDUL, 2010, pág. 3468) (ZAREI; TAJEDDINI; KARIMI, 2014, pág. 151). Seja na indústria petrolífera, química, de aparelhos domésticos, entre outras aplicações (LASHKARI; POSHTAN, 2015, pág. 275), os MITs costumam consumir cerca de 40% a 50% de toda capacidade energética gerada em um país industrializado (THOMSON; FENGER, 2001, pág. 26).

Com tamanha representatividade nas indústrias, é natural que os motores apresentem desgastes e falhas, seja devido ao seu envelhecimento, condições de instalação, aplicações inadequadas, falta de manutenção preventiva e estresse mecânico ou elétrico (GHATE; DUDUL, 2010, pág. 3468) (NANDI; TOLİYAT, 1999, págs. 198-199). As falhas mais comuns, conforme listadas em Nandi e Toliyat (1999, pág. 197), ocorrem nos rolamentos, nos isolamentos no rolamento do estator, através da abertura de barras do rotor ou rachadura nos anéis rotóricos e por excentricidade.

A maioria dos trabalhos relacionados à detecção de falha em motores trifásicos citam dois trabalhos da década de 1980 (IEEE, 1985; ALBRECHT et al., 1986) para indicar os componentes em motores que mais apresentam falha, sugerindo que não há uma pesquisa recente sobre o assunto. Os números divergem (conforme visto na Tabela 1), mas indicam que cerca de 26% a 36% das falhas que ocorrem em motores de indução são provenientes de curto-circuito entre espiras, categorizando esse tipo como a segunda maior causa de defeitos em motores, ficando atrás somente das falhas provenientes do rolamento (variando entre 41% a 44%).

O curto-circuito entre espiras estatóricas do rolamento geralmente ocorre devido a falhas no isolamento (D'ANGELO et al., 2011, pág. 179). O curto se inicia com uma falha de alta impedância entre algumas espiras (NATARAJA, 1989, pág. 590), gerando um aquecimento local que pode ocasionar em uma rápida expansão da falha no bobinamento (TALLAM et al., 2007, pág. 920). Quando alimentados por conversores de

Tabela 1 – Percentual dos principais componentes responsáveis por falhas em MIT

Principais Componentes	Percentual de falha	
	(IEEE, 1985)	(ALBRECHT et al., 1986)
Rolamento	44%	41%
Espiras	26%	36%
Rotor	8%	9%
Outros	22%	14%

Fonte: Adaptado de IEEE (1985) e Albrecht et al. (1986)

frequência, a tendência é a situação ser ainda pior devido às elevadas taxas de transição de tensão no tempo impostas pelos transistores do conversor, o que colabora ainda mais com a degradação do isolamento. Em Kaufhold et al. (2002, pág. 29) é indicado que o estresse (picos de tensão) no isolamento do bobinamento pode chegar a ser dez vezes maior em máquinas acionadas por conversor de frequência em comparação aos alimentados diretamente na linha. Em Contin (s.d.) o autor desenvolve um estudo sobre os efeitos do uso de conversores de frequência no isolamento, indicando que deve ser dada atenção a rapidez do crescimento dos pulsos gerados pelo conversor, bem como pela alta frequência com que esses picos são produzidos.

Há uma tendência na indústria em desenvolver soluções que detectem falhas em MITs conforme indicado em revisões bibliográficas como Nandi e Toliyat (1999) e Pandey, Zope e R. (2012). De acordo com Kaufhold et al. (2002, pág 27), as principais consequências de uma falha em motores é o reparo de equipamento de maior custo, parada de produção, o que gera prejuízos e problemas de saúde e segurança das pessoas próximas ao ocorrido. Ainda no trabalho de Kaufhold et al. (2002, pág 27), o autor alerta também que paradas frequentes e desnecessárias podem ser tão danosas quando paradas por falha, por gerar também perda de produção.

De acordo com Morsalin et al. (2014, pág. 1), as técnicas para detecção de falhas podem ser divididas em duas categorias: métodos invasivos, que são aqueles que incluem manutenções e monitoramentos regulares do sistema; e os não invasivos, que de forma acessível e de baixo custo, diagnosticam as condições do motor sem que haja a necessidade da abertura do mesmo.

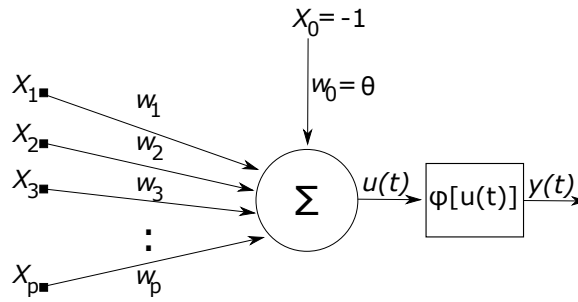
2.2 MultiLayer Perceptron (MLP)

A rede perceptron multicamadas é um dos mais conhecidos e consolidados algoritmos de uma categoria de modelos computacionais definida como redes neurais artificiais (RNAs). As RNAs são assim conhecidas por buscar inspiração na estrutura e funcionamento do cérebro humano, tendo como característica a capacidade de processar dados, simulando a

aprendizagem e generalização de um conhecimento (HAYKIN, 2007, págs. 27-28).

Esse algoritmo, como sugerido em seu nome, é conhecido assim por ser composto de múltiplas camadas de Perceptrons. Como definido em Haykin (2007, pág. 143), o perceptron é a forma mais simples de uma rede neural usada para classificação de padrões ditos linearmente separáveis. Na Figura 1 é mostrado a estrutura básica de um perceptron.

Figura 1 – Representação genérica de um perceptron



Fonte: O autor

Analisando a Figura 1, destacam-se:

- \mathbf{x} : valores que são as entradas a serem processadas pelos neurônios. O valor de x_0 é conhecido como viés (*bias*), assumindo comumente os valores de -1 ou 1. Em termos práticos, é comum considerar o valor do bias como uma entrada adicional ao neurônio, como exibido na Figura 1.
- w : são as sinapses ou pesos dos neurônios. Cada entrada do neurônio é multiplicado por um peso associada a tal entrada. Os valores de tais sinapses são definidos através da etapa de treinamento da rede, que será explicada posteriormente. O peso w_0 é conhecido como limiar do neurônio. De forma similar ao bias, é comum estabelecer tal valor como um peso adicional associado ao bias, definindo também seu valor na etapa de treinamento.
- $u(t)$: é valor do resultado do somatório (Σ) das multiplicações das entradas do neurônio (\mathbf{x}) com seus respectivos pesos (w).
- $y(t)$: representando a saída final do neurônio, após o valor de $u(t)$ ser submetido a uma função de ativação (φ).
- φ : a função de ativação pode assumir diferentes formas, como visto em Engelbrecht (2007, pág. 18), mas, comumente, assume a forma de função

$$\varphi[u(t)] = \frac{1}{1 + \exp[-u(t)]} \quad (\text{Logística}) \quad (2.1)$$

ou

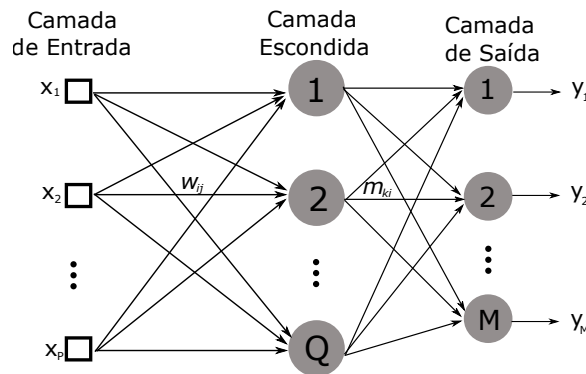
$$\varphi[u(t)] = \frac{1 - \exp[-u(t)]}{1 + \exp[-u(t)]} \quad (\text{Tangente Hiperbólica}). \quad (2.2)$$

enquanto que os neurônios da camada de saída em geral são lineares ou mesmo como as supracitadas.

Algumas variáveis estão em função da variável t por assumirem valores dependentes do padrão que está sendo apresentada à rede. Isto é, estes valores são resultados de operações realizadas com diferentes padrões submetidos a rede em momentos distintos.

A MLP é composta por camadas com um ou mais perceptrons. Tal composição torna a rede MLP uma ferramenta poderosa, sendo altamente atrativa para mapeamentos não-lineares. A Figura 2 expõe a arquitetura básica deste tipo de RNA, auxiliando os conceitos apresentados a seguir, onde além de outros componentes, exhibe as três camadas que normalmente compõem uma MLP: entrada, escondida e saída. Os conceitos apresentados a seguir se limitam ao uso da MLP de uma única camada e como classificador de padrões.

Figura 2 – Representação genérica de uma MLP com uma camada escondida



Fonte: O autor

A camada de entrada é composta pelos sinais a serem processados pela rede neural. Esses sinais são os atributos (ou características) da amostra (também conhecido como padrão) que se busca classificar. Logo, cada amostra, por sua vez, é um vetor de atributos. A camada de entrada da RNA é composta por x elementos mais um, onde o valor de x é quantidade de atributos de cada amostra e a entrada adicional é o viés (*bias*). Dito isso, o objetivo da MLP como um classificador de padrões, é dados um vetor de atributos, que são as entradas da rede, a saída da RNA ser um rótulo indicando qual classe é correspondente à amostra apresentada.

Como dito anteriormente, tanto a camada oculta quanto a de saída são compostas por unidades de processamento chamadas de neurônios artificiais ou simplesmente, perceptrons. Apesar da Figura 2 só conter uma camada oculta, é possível desenvolver MLPs com mais de uma camada. Alguns trabalhos como (HORNIK; STINCHCOMBE; WHITE, 1989; CYBENKO, 1989; FUNAHASHI, 1989) citam que com uma única camada, e certas configurações de rede, é possível aproximar de forma adequada qualquer função

contínua arbitrariamente. Em Maiorov e Pinkus (1999), os autores chegam a conclusões similares, utilizando, contudo, mais de uma camada oculta, onde é possível inclusive aproximar funções descontínuas. Logo, é pouco comum RNAs apresentarem mais do que duas camadas escondidas.

Com os conceitos e notações citados, a expressão matemática que representa a saída do k -ésimo neurônio da camada de saída de uma rede já treinada, apresentada na Figura 2 é dada por

$$y_k^{(o)} = \varphi_k \left[\sum_{i=0}^Q m_{ki} \varphi_i \left(\sum_{j=0}^P w_{ij} x_j(t) \right) \right]. \quad (2.3)$$

Em que:

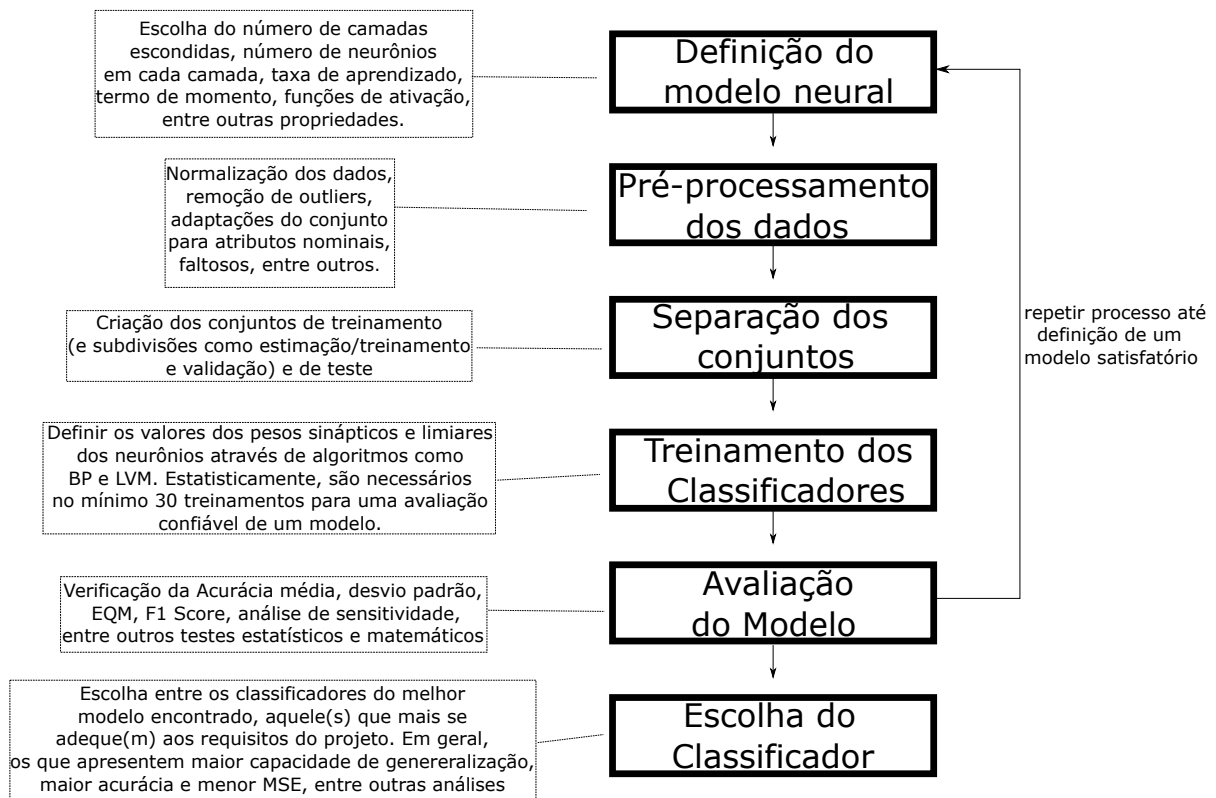
- $y_k^{(o)}$ é a saída do k -ésimo neurônio da camada de saída.
- φ_k é a função de ativação do k -ésimo neurônio da camada de saída.
- Q é o número de neurônios da camada escondida.
- m_{ki} é o peso que conecta o i -ésimo neurônio da camada escondida ao k -ésimo neurônio da camada de saída.
- φ_i é a função de ativação do i -ésimo neurônio da camada escondida.
- P é a dimensão do vetor de entrada, sem o limiar.
- w_{ij} é o peso sináptico que conecta x_j ao i -ésimo neurônio.
- $x_j(t)$ é o j -ésimo componente (atributo) do vetor de entrada $x(t)$.

2.2.1 Treinamento e Validação da MLP

A Figura 3 exibe os passos mais comumente realizados para o treinamento e validação da rede MLP, com um breve resumo de cada etapa ao lado, onde tais passos da figura são detalhados a seguir, junto de algumas referências para detalhes mais aprofundados.

- **Definição do modelo neural:** A definição do modelo neural é feita ao escolher os diversos parâmetros da RNA: número de camadas ocultas, número de neurônios em cada camada, taxa de aprendizado, termo de momento, funções de ativação dos neurônios, critério de parada, valores de inicialização dos pesos, entre outras propriedades. Tais valores podem ser definidos por tentativa e erro, onde o projetista define valores iniciais para tais propriedades da rede e, a partir dos resultados obtidos, vai ajustando-os até encontrar um modelo adequado; ou através de técnicas como busca em grade que faz combinações dos parâmetros de forma semi-automatizada em busca de um modelo ideal. Em Medeiros e Barreto (2015, pág. 55) é possível encontrar diversos procedimentos para definição dos parâmetros. Em Engelbrecht (2007, pág. 93), o autor descreve o impacto e a importância de cada um dos parâmetros da rede MLP.

Figura 3 – Etapas seguidas para desenvolvendo de um classificador neural



Fonte: O Autor

- Pré-processamento dos dados:** É muito comum efetuar algum tipo de pré-processamento dos dados antes dos mesmos serem submetidos à execução da rede neural. O pré-processamento mais comum, é a normalização dos dados no intervalo das funções de ativação que são aplicadas aos neurônios. Como exemplo, ao utilizar a tangente hiperbólica, os dados são normalizados entre o intervalo de -1 a 1. Outros tipos de pré-processamentos como remoção de *outliers* (amostras muito discrepantes daquelas pertencentes a sua mesma classe), adaptação de atributos nominais, adição de atributos ausentes em algumas amostras podem ser feitas, caso necessário. O trabalho desenvolvido por Batista (2003) trata de diversos aspectos de pré-processamento de dados de forma detalhada.
- Separação dos conjuntos:** O conjunto de dados normalmente é separado em dois conjuntos disjuntos: de treinamento e de teste. O conjunto de treinamento, pode ainda ser dividido em um conjunto de estimação (ou treinamento), que é utilizado para definir os valores dos pesos sinápticos e limiares dos neurônios; e o de validação, que é utilizado para definir o momento de parada do treinamento (na validação cruzada com parada prematura, para evitar *overfitting*) ou auxiliar na definição do modelo (em técnicas como *k-fold*). Já o conjunto de teste é

utilizado para verificar a capacidade de generalização do classificador, sendo esse teste o mais importante na avaliação dos resultados. O número de amostras para cada conjunto, assim como a forma de separá-los, é relativo a cada projeto, dependendo muito da quantidade de amostras que o projetista tem em mãos. Em Engelbrecht (2007, pág. 99) são exibidos mais detalhes sobre a relevância desta etapa. Já em Batista (2003, pág. 141) são descritas técnicas para lidar com classes desbalanceadas.

- **Treinamento dos Classificadores:** A definição dos valores dos pesos sinápticos (representados na Figura 2 pelas letras w e m) da rede MLP é dada através do treinamento da rede neural. O algoritmo de treinamento utilizado nessa etapa pode variar de projeto a projeto. Os mais comuns são o Levenberg-Marquardt (LMA) (MORÉ, 1978) e o algoritmo de Gradiente Descendente (também conhecido comumente por Back-Propagation) (HAYKIN, 2007, pag. 138) (MEDEIROS; BARRETO, 2015, pág. 51), algoritmo este utilizado nos experimentos do trabalho e descrito no Apêndice A. Estatisticamente, é comum efetuar ao menos 30 realizações para conclusões mais confiáveis a respeito dos resultados. Em cada um destes treinamentos, mantém-se os parâmetros definidos inicialmente e altera-se a composição dos conjuntos de treinamento, validação (quando aplicável) e teste. Ou seja, em cada realização, todas as amostras do conjunto de dados são redistribuídas de forma aleatória nos conjuntos. No primeiro momento é importante avaliar o modelo neural, verificando assim se a seleção dos parâmetros da rede são adequadas para o conjunto de dados como um todo. E isso é feito através da avaliação do modelo neural.
- **Avaliação do Modelo neural:** Após cada realização, é feita uma análise sobre os resultados obtidos pelos classificadores. A análise é feita em termos de: verificação da acurácia média dos classificadores (com as amostras do conjunto de teste), o desvio padrão ou variância de tais resultados, o Erro Quadrático Médio obtido em comparação com os resultados esperados, *F1 Score* para análise de classificadores binários, uma análise de sensibilidade, recursos computacionais necessários, assim como outros testes estatísticos. O ideal é comparar diferentes modelos e definir um mais adequado para o problema. Por exemplo, pode ser que haja dois modelos onde um tem uma acurácia média um pouco maior que outro modelo. Contudo, o modelo com maior acurácia tem um número significativamente maior de neurônios e exija mais recursos computacionais, como memória e tempo de execução. Logo, dependendo da aplicação e da diferença dos resultados, pode ser mais interessante escolher um classificador com uma menor acurácia mas que exija menos recursos. Ao fim da avaliação e/ou comparação de diferentes modelos, caso a configuração ainda não seja adequada, novos treinamentos com diferentes composições de rede podem ser realizados.

Ao encontrar um que satisfaça os requisitos desejados, um classificador específico (ou uma seleção de alguns) deve ser escolhido para ser utilizado na aplicação final.

- **Escolha do classificador:** A escolha do classificador passa por um processo similar à escolha do modelo, onde os resultados dos testes estatísticos, matemáticos e computacionais - acurácia, EQM, análise de sensibilidade, recurso computacional, entre outros - de cada classificador do modelo escolhido são analisados individualmente. Dessa forma, é possível escolher um classificador mais adequado para o projeto, observando principalmente a capacidade de generalização do classificador.

2.3 Sistemas Embarcados e Redes neurais embarcadas

Nesta seção são abordados alguns conceitos básicos de sistemas embarcados, focando principalmente nos utilizados nesse trabalho. Há também uma subseção descrevendo uma série de características inerentes às aplicações de redes neurais embarcadas.

2.3.1 Conceitos Básicos

Ao longo dos anos, a forma de processamento da informação vem se tornando gradualmente cada vez mais compacta. Dos grandes *mainframes* que chegavam a ocupar salas inteiras aos *smartphones* utilizados atualmente, a perda não é somente de tamanho desses equipamentos, mas sim, em algumas vezes, da percepção geral do que é ou não é um computador. Com conceitos como computação ubíqua cada vez mais explorados, os sistemas embarcados se mostram um conceito fundamental ao lado de outros como a interação humano-computador, contexto computacional e inteligência ambiental (MARWEDEL, 2011, prefácio, xi).

Em termos conceituais básicos, Wolf (2008, pág. 1) define um sistema de computador embarcado como qualquer dispositivo que contém um computador programável mas que não é desenvolvido para ser computador de propósito geral. Entre as aplicações mais comuns estão os eletrodomésticos (geladeira, forno de microondas), telecomunicações (*smartphones*), veículos (freio ABS, computadores de bordo) e, como na aplicação do presente trabalho, em controle industrial (robôs, sistemas de monitoramento e diagnóstico de maquinário). Adicionalmente, Marwedel (2011, prefácio, xiii) define sistemas embarcados como sistemas de processamento de informação em produtos fechados. Ou seja, todos os produtos citados contêm, além de outros componentes, um computador que foi planejado, desenvolvido ou programado para desempenhar um conjunto específico de tarefas.

Conforme dito por Silva (2008, pág. 13), apesar de haver certas características em comum nos sistemas embarcados, não há a homogeneidade que pode ser encontrada em

computadores de propósito geral. Isso se deve às restrições únicas referentes aos requisitos de diferentes projetos que podem exigir módulos distintos de *hardware* e *software*. Ao comparar os exemplos anteriores como o freio ABS e uma geladeira, por exemplo, é lógico imaginar o quão diferentes esses projetos são em termos de requisitos.

Apesar de não serem exigências para categorizar um sistema embarcado, certas características comuns permeiam grande parte desta categoria de produtos. Marwedel (2011, pág. 5) destaca a importância de um sistema embarcado ser confiável ao atender os seguintes requisitos:

- confiabilidade: a probabilidade que o sistema não irá falhar;
- manutenibilidade: a probabilidade de ser reparado em tempo hábil mediante falha do sistema;
- disponibilidade: a probabilidade do sistema estar disponível para executar suas funções;
- seguro: a característica que descreve que o sistema não irá causar danos ou perigo;
- segurança: a propriedade que descreve que informações confidenciais presentes no sistema permaneçam confidenciais;

Outra característica fundamental é que ele deve ser eficiente. Há um consenso nos trabalhos relacionados a sistemas embarcados em determinar a eficiência dos mesmos através de certas métricas. Muitas vezes essas métricas são utilizadas para comparação entre diferentes plataformas e soluções (sendo as três primeiras na lista abaixo as mais utilizadas), que estão também alinhadas com o que é descrito em Marwedel (2011, pág. 5) como requisitos de eficiência:

- energia: a quantidade de energia exigida para executar as tarefas;
- tempo de execução: o tempo gasto para executar as tarefas;
- tamanho e qualidade do código: a pequena quantidade de memória é uma limitação recorrente em sistemas embarcados e um código de qualidade é determinante na maioria dos projetos;
- peso: sistemas portáteis devem ser leves, sendo essa uma característica importante em certos tipos de produtos, como *smartphones*;
- custo: para produtos de larga escala, o custo é um requisito primordial para aumentar a competitividade no mercado;

2.3.2 Arquitetura Geral

É possível dividir um sistema embarcado em duas partes: uma composta pelo *hardware*, onde estão todos os componentes físicos do sistema, e outra composta pelo *software* que é o código que contém a lógica de uso desses componentes para alcançar

o objetivo do projeto. Essa divisão é muito mais didática do que prática, uma vez que como apontado por Marwedel (2011, pág. 11) a integração e elaboração dessas duas partes em paralelo é muito importante para alcançar a excelência do sistema. Isso se deve aos requisitos específicos de cada projeto que são determinantes para definir quais tecnologias devem ser utilizadas para atender a demanda do produto.

Ao determinar o *hardware* necessário para o sistema que está sendo desenvolvido, um dos componentes mais importantes é a unidade de processamento. Tal unidade pode ser um microprocessador de propósito geral, um ASIC, ser descrita em uma linguagem de descrição de *hardware* e ser implementada em FPGAs ou CPLDs e, para soluções mais voltadas para processamentos de sinais, ser um DSP. A definição da solução a ser utilizada está diretamente relacionada aos requisitos de projeto como tempo de resposta do processamento, custo e prazos.

Interfaces e dispositivos de entrada e saída também são essenciais nos produtos embarcados. Os dispositivos de entrada são responsáveis por receber os dados a serem processados. Já os de saída, é o meio por onde são exibidos ou encaminhados os resultados de tais processamentos. Esses tipos de dispositivos são os mais variados pois estão relacionados diretamente às necessidades do produto. É comum encontrar teclados, sensores, microfones e câmeras, entre outros, como tipo de dispositivos de entradas em sistemas embarcados. Já os dispositivos de saída abrangem *displays*, alertas sonoros e interfaces de comunicação para encaminhar os resultados para outras fontes como outros dispositivos e computadores.

Há uma associação direta de microcontroladores com sistemas embarcados por sua extensa utilização. Os microcontroladores são categorizados como SoC (*System-on-a-Chip*) onde todos ou uma grande parte dos componentes citados acima são integrados em um único *chip*. Basicamente os microcontroladores são compostos por um núcleo de processador, memória, conversores de sinal e periféricos configuráveis de entrada e saída (OLIVEIRA; ANDRADE, 2006, pág.34). Outros componentes bastante comuns como *timers* e contadores também podem ser encontrados nesse tipo de sistema. Esses dispositivos são amplamente utilizados em automação industrial, residencial, eletrodomésticos e brinquedos, principalmente devido seu relativo baixo custo, robustez e simplicidade em comparação a outras soluções. De forma similar, há também uma família de dispositivos que alia as vantagens dos SoC com a alta capacidade de processamento de sinais dos DSP, os DSC.

Nas próximas subseções são explorados de forma mais detalhada as tecnologias e as respectivas placas de desenvolvimentos utilizadas no presente trabalho: FPGA e DSP.

2.3.3 FPGA

O *Field Programmable Gate Arrays* (FPGA) é uma unidade de *hardware* totalmente configurável, sendo a mais conhecida desta categoria de sistemas. Como indicado no nome, são dispositivos que são configurados em campo após sua fabricação, de forma personalizada para atender à demanda específica da aplicação.

Em geral, os FPGAs são utilizados como solução quando o uso de *hardwares* totalmente customizados, como ASICs, não é viável. Ao se falar de implementações em *hardware* de RNAs, Sharma (2007, pág. 2) aponta duas características dos FPGAs que são importantes para a escolha da plataforma para implementação:

- A rede MLP é um algoritmo naturalmente paralelo, uma vez que multiplicações e somatórios de cada neurônio pode ocorrer de forma paralela e independente, corroborando com o uso do FPGA, uma arquitetura também paralela, uma vez que os processadores de propósito geral operam de forma sequencial.
- As operações das redes neurais requerem operações matemáticas simples, com baixa precisão que podem ser executadas mais rápidas em *hardwares* baratos e de baixa precisão. Uma vez que a tendência dos *hardwares* é de se tornarem mais baratos, os *hardwares* customizáveis podem ser construídos para desempenhar computações cada vez mais complexas.

2.3.4 DSP

Digital Signal Processors (DSP, que também pode designar *Digital Signal Processing*) são microprocessadores especificamente desenvolvidos para executar tarefas de processamento digital de sinais. Computadores, em geral, são extremamente capazes de realizar duas tarefas: manipular dados e calcular operações matemáticas (SMITH, 1997, pág. 503). Todos os microprocessadores são capazes de executar ambas as tarefas, mas é difícil desenvolvê-los de forma otimizada para ambas. Dessa forma, é mais comum que processadores tradicionais sejam primariamente direcionados para lidar com dados. Similarmente, os DSPs são desenvolvidos de forma otimizada para efetuar cálculos matemáticos (SMITH, 1997, pág. 504).

Devido principalmente à natureza das aplicações em que são normalmente utilizados, DSPs são desenvolvidos de forma que seja possível estimar o tempo de execução de suas tarefas (SMITH, 1997, pág. 506). Smith (1997, pág. 506) cita também que a maioria dos DSPs são utilizados em aplicações onde o processamento é contínuo, quando não há uma definição clara de começo e final dos sinais. Essa continuidade dos sinais está diretamente relacionada ao custo do projeto, consumo energético e memória necessária do projeto.

A principal diferença entre os DSPs e dos microprocessadores e microcontroladores, é que tanto sua arquitetura quanto suas bibliotecas de funções nativas são desenvolvidas e

otimizadas para o processamento de sinais. Como indício de seu uso e relevância, em Wolf (2008, pág. 57) é citado que a grande maioria dos smartphones contém ao menos um DSP.

2.3.5 Redes Neurais Embarcadas

A maioria das aplicações comerciais de redes neurais é implementada apenas em programas de alto nível, ou seja, RNAs que são executadas em computadores, *notebooks* ou *workstations*. Nessas situações, não há uma limitação de recursos computacionais como memória e velocidade de processamento. Contudo, algumas aplicações exigem o uso de uma RNA embarcada seja pelo custo do projeto, velocidade de processamento e local do seu uso, entre outras características de projeto (MISRA; SAHA, 2010, pág. 239).

É possível encontrar projetos de RNAs em *hardware* utilizando diversos tipos de tecnologias e arquiteturas como digitais (BERMAK; MARINEZ, 2003), analógicas (BROWN; YU; GARVERICK, 2004), híbridas (LEHMANN; BRUUN; DIETRICH, 1996), baseadas em FPGA (NEDJAH; MOURELLE, 2007) e até não eletrônicas, como implementações óticas (fotônica) (MOERLAND; FIESLER; SAXENA, 1996). A implementação de RNAs em *hardware* implica, geralmente, em adicionais esforços relacionados ao desenvolvimento de componentes que auxiliam o funcionamento e execução da RNA. Sensores e sistemas para aquisição de dados, pré ou pós-processamento das entradas e saídas do algoritmo, por exemplo, são necessidades que, muitas vezes, estão atreladas ao projeto de RNAs embarcadas (BOSQUE; CAMPO; ECHANOBÉ, 2014, pág. 286). As redes neurais que são embarcadas, geralmente são treinadas em computadores diferentes da plataforma alvo. Por isso tais componentes não são necessários pois, muitas vezes, os dados utilizados no algoritmo já estão pré-processados e armazenados em arquivos. Em muitas aplicações reais em campo, o intervalo para aquisição de dados é pequeno e determinante para definição da plataforma a ser utilizada no projeto, sendo também considerada a quantidade de memória necessária para armazenar e pré-processar os dados no formato necessário para execução da rede (KOZIEL et al., 2015, pág.61). Quando implementadas em SoC (microcontroladores), há uma extensa gama de dispositivos com diferentes configurações variando desempenho, memória disponível, potência dissipada, custo, entre outras. A escolha de qual dispositivo utilizar deve também atender as necessidades do projeto, seja dos recursos necessários para execução e resposta da rede, assim como o recurso disponível para desenvolvimento do projeto.

São muitas as especificações a serem escolhidas ao embarcar uma rede neural. Em relação ao algoritmo a ser embarcado, há diversos tipos de redes neurais como *Adaptive Linear Neuron* (ADALINE), *Spiking Neural Networks* (SNN), *Radial Basis Function network* (RBF), *Perceptron*, *MultiLayer Perceptron* (MLP), *Recurrent Neural Network* (RNN), *Neural Gas*, *Self-Organizing Map* (SOM), entre diversas outras, além de variações e combinações entre as redes. Em se tratando da topologia da rede, devem

ser definidas as funções de ativação dos neurônios (tangente hiperbólica, função sinal, entre outras). O algoritmo de aprendizado (Gradiente Descendente/Back-Propagation, Levenberg-Marquardt, entre outros) tem uma importância ainda maior para problemas onde a etapa de treinamento é feita em tempo real, já na plataforma embarcada. Há a necessidade também de definir o número e tipo de entradas e saídas, número de neurônios, número de camadas de neurônios e taxas de aprendizado (MISRA; SAHA, 2010, pág. 241). Ao se tratar do *hardware*, são muitas as opções de tecnologia a serem utilizadas (já citadas), além das decisões que envolvem a representação dos dados (ponto fixo ou ponto flutuante), armazenamento dos pesos, bits de precisão, conexões programáveis ou via fio (*hardwire connections*), aprendizado no *hardware*, reaprendizado ou aprendizado *off-line*, implementação das funções de ativação ou utilização de tabelas de memórias (MISRA; SAHA, 2010, pág. 241) (BOSQUE; CAMPO; ECHANOBÉ, 2014, pág. 286). Todas essas decisões acabam sendo determinantes no desenvolvimento do projeto (MISRA; SAHA, 2010, pág. 240). As questões de engenharia (tempo de projeto, recurso financeiro disponível, tempo de resposta exigida da rede, restrições de projeto) são necessárias para a escolha de qualquer uma das especificações citadas acima. Tais escolhas podem ser determinantes para o sucesso do projeto ou até mesmo a decisão do uso da rede neural para a questão a ser resolvida.

Visto o grande impacto da escolha da topologia ao projetar uma RNA embarcada, alguns estudos surgem para tentar otimizar o processo. Em Esmaeilzadeh et al. (2007), os autores propõem o padrão de projeto *Neural-Network-based Embedded system design Pattern* (NNEP). Conforme o nome já sugere, é um padrão para projetos embarcados que utilizam redes neurais que visa diminuir o *time-to-market* e maximizar o reuso, conceito primordial na engenharia de *software* de sistemas embarcados. Comuns também são implementações de co-processadores, geralmente em FPGA, que executam especificamente redes neurais. Configuráveis, esses co-processadores são uma solução para a falta de flexibilidade e escalabilidade de algumas redes neurais em hardware (AKLAH; ANDREWS, 2015, pág. 427) que são projetadas somente para uma função. Em Aklah e Andrews (2015) é proposto um co-processador reconfigurável para execução de redes MLPs. No trabalho de Eickhoff, Kaulmann e Rückert (2006) é proposto um *hardware* reconfigurável de neurônios, chamado de *Simple Reconfigurable Neural hardware Structure* (SIRENS). Escrito em VHDL, com a proposta dos autores é possível representar diversos tipos de neurônios para diversas aplicações.

2.4 Conclusão

Este capítulo descreve, no primeiro momento, o curto-circuito entre espiras em motores de indução trifásico. Esta falha representa a segunda causa mais comum de falha em MITs. A parada de funcionamento dos MITs acarretam prejuízos para as indústrias,

assim como representa uma ameaça a segurança das pessoas que estão próximo ao ocorrido. No segundo momento, o capítulo descreve a rede MLP como classificador de padrões, algoritmo esse utilizado neste trabalho para diagnosticar a falha em MITs. No terceiro momento, conceitos sobre sistemas embarcados são apresentados, onde as plataformas onde o classificador neural para detecção de curto-circuito entre espiras em MIT proposto no trabalho são exploradas.

3 Trabalhos Relacionados

Neste Capítulo são descritos os trabalhos relacionados à detecção de falha de motor utilizando redes neurais, de redes neurais embarcadas com aplicações comerciais e industriais, assim como a pesquisas que utilizam o mesmo conjunto de dados deste trabalho usando outros algoritmos de inteligência computacional.

3.1 Estado da Arte na detecção de Curto-circuito entre espiras utilizando redes neurais

Diversas soluções foram propostas para detecção de curto-circuito entre espiras em MIT. Em Morsalin et al. (2014), é proposto um classificador neural treinado com algoritmo Levenberg-Marquardt, que tem como entradas a corrente do motor, tensão da alimentação e velocidade do rotor. Como modelo para aquisição e teste, os autores utilizam um motor de 0,5 cv, de única fase operando a 50 Hz e a vazio (sem carga). Os resultados validam, de forma satisfatória, o uso de redes neurais para o problema apresentado. Contudo, o problema é limitado a somente uma configuração de motor (monofásico, sem carga, operando a 50 Hz) e necessita de pelo menos três sensores para as entradas.

Em Palácios et al. (2015) dois classificadores neurais (MLP e RBF) são utilizados para detecção de falha. Os dados são adquiridos a uma taxa de amostragem de 25 kHz no intervalo de 5 a 15 segundos, alterando o comprometimento das espiras (3%, 5% e 10%) de um motor de 1 cv, variando também para cada severidade de curto o conjugado da carga aplicado ao eixo do motor. Os dados são classificados como ‘condição normal’ ou ‘condição de falha’. Utilizando a técnica de discretização do sinal da corrente, com entradas de 30 características por padrão, os autores conseguem acurácias que variam de 75% a 98,55% utilizando a rede RBF e 98,34% a 98,96% com a rede MLP. Após aplicação de PCA e a redução para 7 características por amostra, as acurácias variam entre 87,71% a 89,79% e 83,13% a 85,63% para as redes RBF e MLP, respectivamente.

Em Bouzid e Champenois (2013) uma rede MLP treinada com o algoritmo *Back-Propagation* é utilizada pra detectar curto-circuito no bobinamento. Para testes, é utilizado um modelo simplificado acoplado com falha (original “*A faulty simplified multiple coupled circuit model*”) de 1,1 kW. Nesse caso, os autores tentam classificar diferentes tipos de falhas: o curto entre espiras, entre fases e entre fase e terra. Como entrada, utilizam as componentes simétricas para extrair os 4 atributos para alimentar a rede neural: a magnitude e o ângulo da fase das correntes de sequência zero e negativa. Os resultados para teste são de 100% de acurácia para todos os problemas apresentados. Contudo, os

valores são adquiridos de um modelo e não de um motor real, operando em uma única frequência de operação (50 kHz). Para aquisição das 4 entradas, é necessário um sensor para leitura das correntes de cada fase o que dificulta também a implementação industrial.

Já no trabalho apresentado por Lashkari e Poshtan (2015), uma rede MLP treinada com *Back-Propagation* é utilizada para detectar a falha no bobinamento e indicar a fase na qual a falha ocorre, assim também quando ocorre algum tipo de desbalanceamento de tensão entre as fontes de alimentação. Utilizando-se de um modelo matemático, o autor simula o motor de 5 hp operando sem falha, com falha de até 20% do bobinamento em cada uma das fases e sob condições de desbalanceamento de tensão entre as fases. Para detectar as falhas, a diferença das três fases e da corrente e tensão de cada fase são utilizadas. Treinando e validando com amostras em diferentes condições de falhas (diferentes números de espiras em falha, entre 1% a 20% das espiras) e diferentes níveis de desbalanceamento de tensão entre as fases (1% a 5%), os resultados indicam que a técnica é bastante eficiente. Contudo, a solução não se mostra muito prática para construção de um protótipo comercial de baixo custo, visto a necessidade de um número significativo de sensores para coleta dos atributos para rede neural.

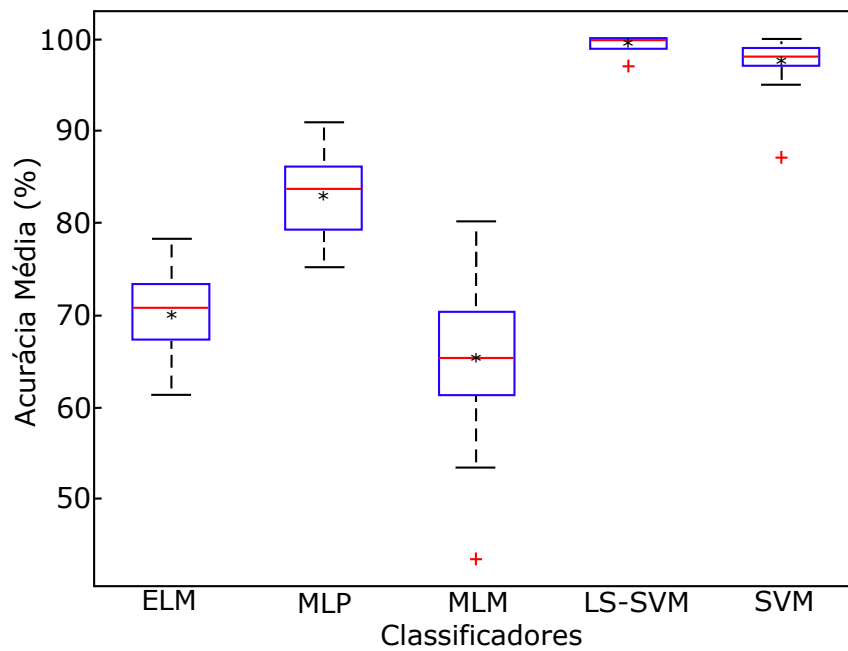
Analizando os trabalhos citados, não se nota nenhuma menção a algum projeto ou intenção de criar um classificador embarcado. Ou pelo menos, não há uma preocupação perceptível de criar-se um classificador que atenda possíveis restrições de um projeto embarcado como tempo de computação, período de aquisição e resposta de diagnóstico, entre outras características. Há alguns protótipos que detectam outros tipos de falha de motor como em Sobrinho et al. (2015) para detecção de quebra de barra, mas não de curto-circuito entre espiras.

Observa-se também que a maioria dos trabalhos tentam identificar a falha em somente uma condição de operação do MIT, ou seja, operando em uma única frequência ou nível de carga e sem muitas descrições das condições das falhas. Um bobinamento contém diversas espiras e a falha pode estar presente entre um número diferente de espiras, assim como intensidades diferentes de falhas. A falha pode estar presente também em diferentes fases do motor. A abordagem apresentada no presente trabalho e também pesquisas anteriores dos quais ele se baseia, pretende suprir essas questões ao adquirir amostras em variadas condições e treinar o classificador de forma a generalizar a falha, independente da configuração de operação do motor.

O problema de detecção de falha de curto circuito tem sido objeto de investigação por outros autores que utilizam o mesmo conjunto de dados apresentados neste trabalho. Publicações anteriores (COELHO et al., 2014; COELHO; MEDEIROS, 2013; OLIVEIRA; PONTES; MEDEIROS, 2013) abordam o tema com foco na avaliação do desempenho de classificadores com diferentes paradigmas de aprendizado de máquina. Dentre os classificadores investigados, os autores destacam as redes neurais artificiais MLP,

ELM e Self-Organizing Map (SOM) (KOHONEN; SCHROEDER; HUANG, 2001), os classificadores fundamentados na Teoria do Aprendizado Estatístico tais como SVM e LS-SVM, e um algoritmo recente chamado Minimal Learning Machine (MLM) (SOUZA et al., 2013). Os resultados alcançados na classificação binária, ou seja, distinção da condição de falha em relação à condição normal, são apresentados na Figura 4.

Figura 4 – Diagrama de caixa comparando a acurácia de diferentes classificadores projetados.



Fonte: Coelho et al. (2014)

Observa-se que os classificadores ELM e MLM apresentam taxas médias de acerto bem inferiores a 80%, o que é considerado como insatisfatório. Já nos classificadores SVM e LS-SVM, suas taxas médias de acerto estão bem próximas a 100%, com taxas máximas atingindo 97,5% e 99,5%, respectivamente. Entretanto, o número médio de vetores de suporte necessários para garantir tal desempenho ao classificador SVM é da ordem de 60% do número total de amostras (aproximadamente 141 vetores suporte). Já o LS-SVM requer que todas as amostras do conjunto de treinamento sejam vetores de suporte (cerca de 235 vetores suporte). Dado o objetivo de desenvolver um protótipo embarcado de algum classificador para aplicação industrial no monitoramento de falhas em motores de baixa potência e, portanto, de baixo custo, a demanda excessiva por memória pode requerer um aumento na complexidade do *hardware* e inviabilizar economicamente a aplicação.

Ao observar os resultados, estima-se que a MLP é o algoritmo que propicia a melhor relação custo-benefício. Sua taxa média de acerto é acima de 80%, com taxa máxima acima de 90%, além da baixa demanda de recursos computacionais. A topologia utilizada

apresenta cinco neurônios na camada oculta e um na camada de saída, requerendo o armazenamento de apenas 41 parâmetros (pesos) em memória, justificando assim, a sua escolha para tentativa de elaboração do protótipo neste trabalho.

3.1.1 Estado da Arte de aplicações de redes neurais embarcadas

A seguir são apresentados trabalhos que descrevem ou tem como objetivo algum tipo de aplicação real (comercial ou industrial) das redes neurais. Procura-se chamar atenção não só para o processo de treinamento das redes neurais mas sim as adaptações necessárias para que as mesmas sejam aplicáveis nos projetos embarcados. Outros detalhes como plataforma utilizada, comparação entre plataformas, erros obtidos em comparação aos resultados originais, aquisição dos dados e pré-processamento são levantados, quando disponíveis.

Com aplicação também em MITs, em Graciola et al. (2016), uma rede neural MLP treinada com Levenberg-Maquardt de única camada oculta com 15 neurônios é embarcada em um DSC para estimativa de velocidade do rotor em MIT. Como entrada para rede, os autores utilizam os valores das três correntes adquiridos em um motor de 1 HP, operando a 60 Hz, simulado através de um modelo matemático. Após treinamento em um computador convencional, a rede escolhida é embarcada. Três sensores de corrente são conectados à placa de desenvolvimento, assim como um circuito de filtro passa-baixa e outro de condicionamento de sinal. Nos testes, a velocidade estimada pelo DSC é exibida em um *display*. Em comparação aos resultados obtidos na máquina onde é efetuado o treinamento original, os autores apontaram um pequeno aumento do valor estimado da velocidade e do erro relativo com os valores de, respectivamente, de 0,093-0,37% no dsPIC contra 0,022-0,057% obtido no computador original.

No trabalho elaborado por Basterretxea, Echanobe e Campo (2014) uma MLP com duas camadas ocultas é utilizada para reconhecer movimentos humanos e é embarcada em diferentes composições de rede (números de neurônios nas camadas ocultas e diferentes atributos para classificação) em um FPGA. A pesquisa aponta e descreve o extenso trabalho de aquisição e pré-processamento de dados que envolve Análise de Componentes Principais (PCA, do inglês *Principal component analysis*), o que demanda grande custo computacional. Como resultado final, todo processo de aquisição e pré-processamento dos dados é implementado para ser executado no *soft-processor* MicroBlaze, um processador de 32 bits, enquanto a rede neural é implementada em *hardware*. Para testes de comparação, todo o processo é também embarcado para ser executado pelo *soft-processor* da placa. O resultado aponta que a velocidade do uso do pré-processamento via *soft-processor* e a execução da RNA em *hardware* é mais econômico energeticamente e mais rápido do que a implementação total em *software* para execução no *soft-processor*. A perda de acurácia média em comparação aos resultados obtidos originalmente em um computador

de propósito geral é de somente 0,5%.

Já no trabalho apresentado em Saldanha e Bobda (2015), uma MLP com 300 atributos de entrada, 100 neurônios na única camada oculta e 10 neurônios na camada de saída é implementada em *hardware* utilizando o *soft-processor* MicroBlaze em uma Spartan 6. Para validar a implementação, os autores utilizam o conjunto de dados MNIST que consiste de imagens de dígitos escritos a mão. A saída de 10 neurônios é referente exatamente aos 10 dígitos (0-9) onde a saída de cada neurônio quando ativada pela saída '1' representa um respectivo dígito. Os autores apontam como dificuldade uma considerável necessidade de memória, no escopo de sistemas embarcados, para armazenamento dos pesos como um dos problemas. Um *speed-up* de $\times 3$ foi atingido em comparação à execução em uma CPU de propósito geral. Além disso, os autores avaliam o impacto sobre a remoção de pesos muito pequenos dentro de diferentes intervalos pré-estabelecidos. Em um dos testes, ao remover os pesos com valores dentro do intervalo de -0,001 a 0,001, cerca de somente 4,34% dos pesos originais são mantidos, com uma redução de 59,9% do tempo de execução e sem perda de acurácia.

O estudo feito por Koziel et al. (2015) detalha de forma bastante significativa o processo para embarcar em um MCU uma rede neural desenvolvida originalmente em uma CPU de propósito geral. O objetivo do trabalho é embarcar uma RNA para estimar a força muscular aplicada por ciclistas durante a prática da atividade física. Os autores levantam as dificuldades para alcançar as restrições de tempo real, assim como as adaptações necessárias da rede neural original, como a divisão da mesma em duas redes menores e diferentes, assim como a substituição das funções de saída dos neurônios por interpolação de funções polinomiais de segundo grau. Outro problema identificado, é a dificuldade de aplicar os filtros necessários para pré-processamento dos dados. A limitação de desempenho do microcontrolador e a necessidade de respostas em tempo real são os maiores obstáculos citados pelos autores. Após embarcar as etapas de pré-processamento e as redes em um ARM Cortex M4F, um microcontrolador de 32 bits com instruções de DSP acoplado, algumas adaptações em termos de quantidade de dados para execução da rede e precisão do pré-processamento são feitos para atingir as restrições do projeto. Nenhuma citação sobre os impactos dessas alterações na acurácia dos classificadores é feita.

No artigo de Husin et al. (2012) é apresentada a ideia e processo de desenvolvimento de um sistema embarcado portátil para identificar folhas de plantas. É proposta uma rede neural com 4800 entradas referentes a resolução das imagens de entrada (60 x 80 *pixels*) com 20 neurônios na camada oculta e 20 na camada de saída. Ambas camadas utilizam tangentes hiperbólicas como função de ativação. O processo de diagnóstico consiste inicialmente na captura das imagens das folhas por uma câmera CMOS. Após isso, é efetuado o pré-processamento que consiste em converter os dados para escala de cinza, limiarização, detecção das bordas das folhas, extração das características e, por fim, a

execução na rede neural para um diagnóstico que é exibido em um display LCD. A placa onde é implementada o protótipo é composta por um microcontrolador DS89C450 (uma implementação do clássico microcontrolador 8051 de 8 bits da Intel), duas interfaces seriais, um *keypad* 4x4, 128kB de SRAM, *slot* de cartão SD e display LCD. O tempo de resposta para cada processamento das imagens desde a captura até a classificação é de 30 segundos. Os testes no próprio protótipo com cerca de 1400 folhas de plantas correspondentes aos 20 tipos de espécies que o classificador é capaz de reconhecer apontam uma taxa 98,9% de acurácia.

3.2 Conclusão

Este capítulo descreve o estado da arte da detecção de falhas em MIT utilizando algoritmos de inteligência artificial, assim como de aplicações embarcadas que utilizam de redes MLP. De forma geral, é mais comum encontrar implementações de redes neurais, seja em MCU, DSC e principalmente FPGAs, que não são direcionadas a alguma aplicação industrial ou comercial. As pesquisas tendem, principalmente, validar a rede neural na plataforma embarcada e verificar a quantidade de recurso utilizado, tempo de execução e o impacto do processo de adaptação dos classificadores na acurácia. Isso acaba negligenciando a importância de processos como pré e pós processamento. Nesse trabalho, estas questões também são apresentadas em contraponto aos trabalhos encontrados.

4 Processo de aquisição dos dados de Falha de Motor e Treinamento dos Classificadores

Neste capítulo, é descrita a metodologia usada na bancada de ensaio da aquisição dos sinais que compõem o conjunto de dados utilizado para definição dos modelos neurais, tais etapas realizadas originalmente por Oliveira (2014). Após esta etapa, são exibidos os resultados preliminares de dois projetos para diferentes formas de classificação dos dados. É importante observar que algumas etapas do processo é omitida com o intuito de alinhar as partes mais relevantes com o objetivo do trabalho. É possível encontrar maiores detalhes em relação aos equipamentos e processo de aquisição em Oliveira (2014, pág. 46) e sobre seleção dos atributos em Oliveira (2014, pág. 64). Vale alertar que todo processo de aquisição das amostras de corrente foi realizado pelo autor Oliveira (2014), assim como a definição do modelo neural utilizado no final deste capítulo para treinamento do classificadores.

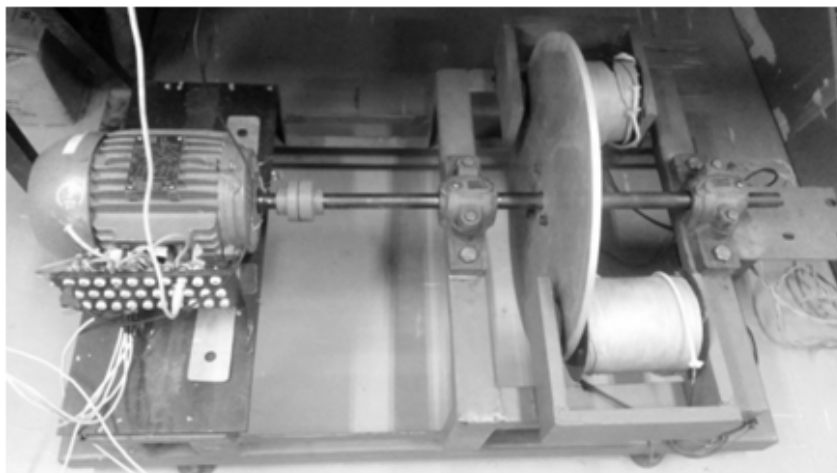
4.1 Descrição do processo de aquisição dos dados

Para treinamento da rede neural a ser embarcada, são utilizadas informações de um motor real especialmente adaptado para a emulação da falhas entre espiras. Para aquisição dos dados, uma bancada composta por um motor de indução trifásico conectado em delta, um conversor de frequência e um freio magnético para aplicação de carga são utilizados. A Figura 5 exibe a composição da bancada e na Figura 6 o processo de aquisição.

São aplicados diferentes níveis de carga durante aquisição dos dados com o uso de um Freio de Foucault. Nesse ensaio, 3 níveis de carga são aplicados: sem carga, 50% de carga e 100% de carga (a plena carga). O conversor de frequência é utilizado para aplicar diversas frequências de operação no motor. Os valores aplicados nesse ensaio são: 30 Hz, 35 Hz, 40 Hz, 45 Hz, 50 Hz, 55 Hz e 60 Hz. Para leitura das correntes, 3 sensores de efeito hall são utilizados, em cada fase ligada ao motor.

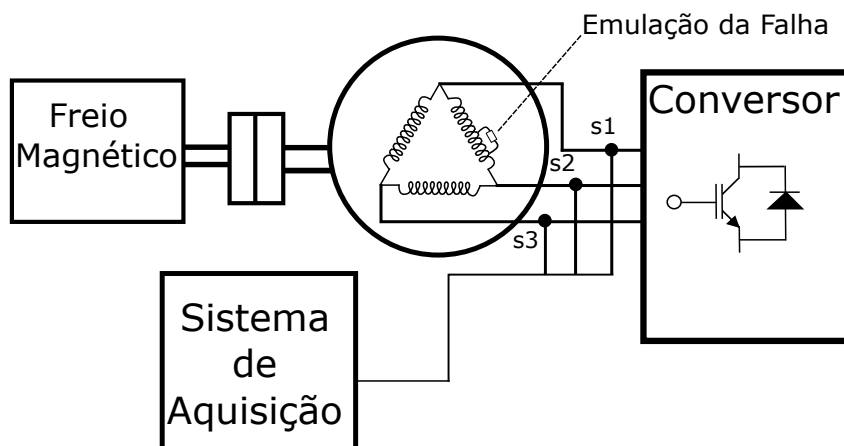
Durante a aquisição dos dados, são capturadas diferentes combinações e situações de falha, uma vez que durante o processo, foram emuladas diferentes intensidades e extensões de falha. Três níveis de extensão de falha são aplicadas sobre as espiras representando em percentual os valores de 1,41%, 4,81% e 9,26% das espiras em curto circuito. Dois níveis de intensidade de falhas são emuladas também, indicados nesse trabalho por Alta Impedância (AI) e Baixa Impedância (BI). De uma forma simplificada, as amostras de AI

Figura 5 – Motor rebobinado acoplado ao freio de Foucault para aquisição do conjunto de dados



Fonte: CUNHA (2016)

Figura 6 – Processo de aquisição das amostras do motor para composição do conjunto de dados



Fonte: O autor

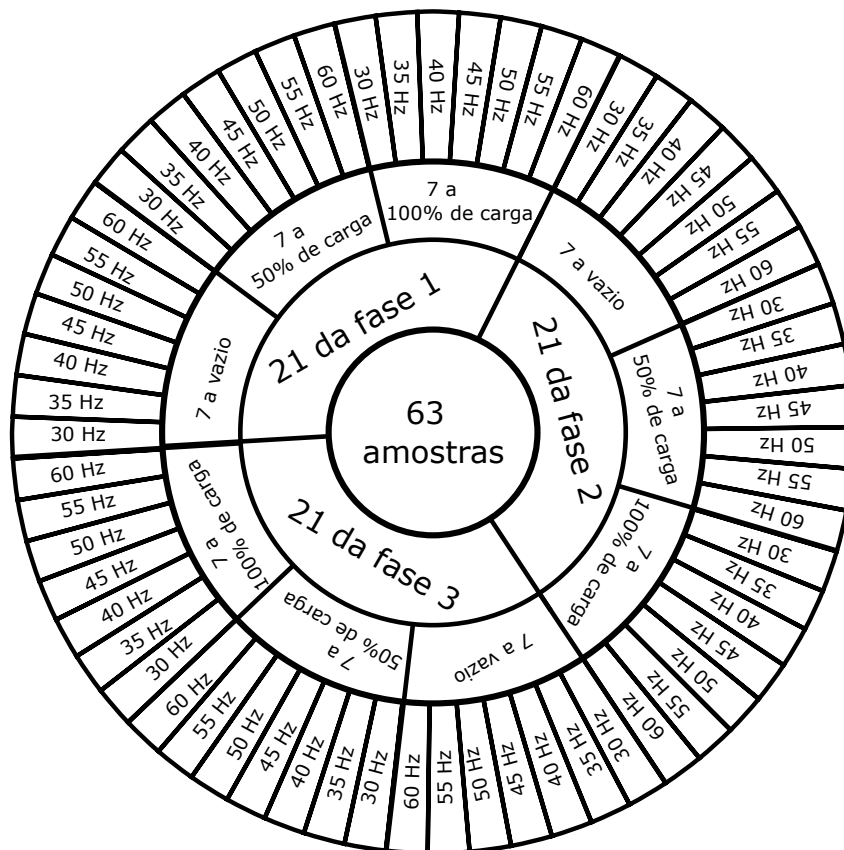
representam falhas em um estágio inicial, sendo consideradas mais “leves”. Já as amostras de BI, representam curto-circuitos num estágio mais avançado, sendo consideradas como falhas mais “severas”. Durante o processo, há uma limitação de corrente de curto-circuito para que o motor não seja danificado.

O resultado é um conjunto de dados representando diversas condições operacionais do motor (diferentes frequências e níveis de carga), com valores da corrente capturados em cada fase do motor por sensores de efeito hall. No total, são representados 6 tipos diferentes de composições de falhas:

- AI em 5 espiras, 1,41% do total (listadas a partir daqui como AI1 ou AI nível 1)
- AI em 17 espiras, 4,81% do total (listadas a partir daqui como AI2 ou AI nível 2)
- AI em 32 espiras, 9,26% do total (listadas a partir daqui como AI3 ou AI nível 3)
- BI em 5 espiras, 1,41% do total (listadas a partir daqui como BI1 ou BI nível 1)
- BI em 17 espiras, 4,81% do total (listadas a partir daqui como BI2 ou BI nível 2)
- BI em 32 espiras, 9,26% do total (listadas a partir daqui como BI3 ou BI nível 3)

O conjunto completo de dados é composto por 7 classes (1 de motor operando sem falha e 6 diferentes condições de falha). A quantidade de amostras de cada classe deriva das combinações de fase de aquisição da amostra \times frequência de operação do motor \times nível de carga do motor. É ilustrado na Figura 7 a organização de cada classe do conjunto de dados.

Figura 7 – Detalhamento das amostras - Número por fase de coleta, nível de carga e frequência de operação

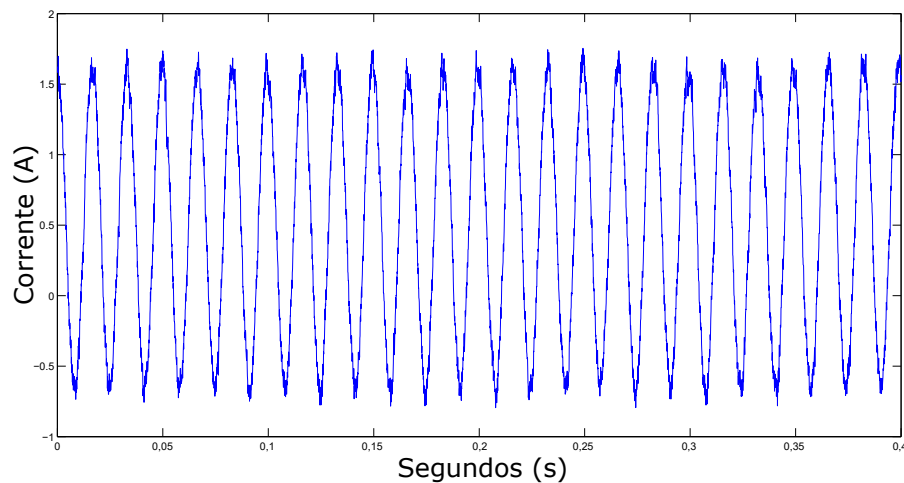


Fonte: O autor

Enquanto o motor é submetido a cada uma dessas situações em específico, é efetuada a leitura da corrente em cada uma das fases a uma taxa de 10 kHz durante 10

segundos. Esses valores adquiridos durante os ensaios são armazenados, gerando para cada amostra 100.000 pontos correspondente a taxa de amostragem \times o tempo de coleta. Para ilustrar parte do resultado, é exibida na Figura 8 a leitura da corrente de uma fase do motor operando a vazio, sem falha, a 60 Hz, durante 0,4 segundos.

Figura 8 – Leitura da corrente durante 0,4 segundos a 10kHz

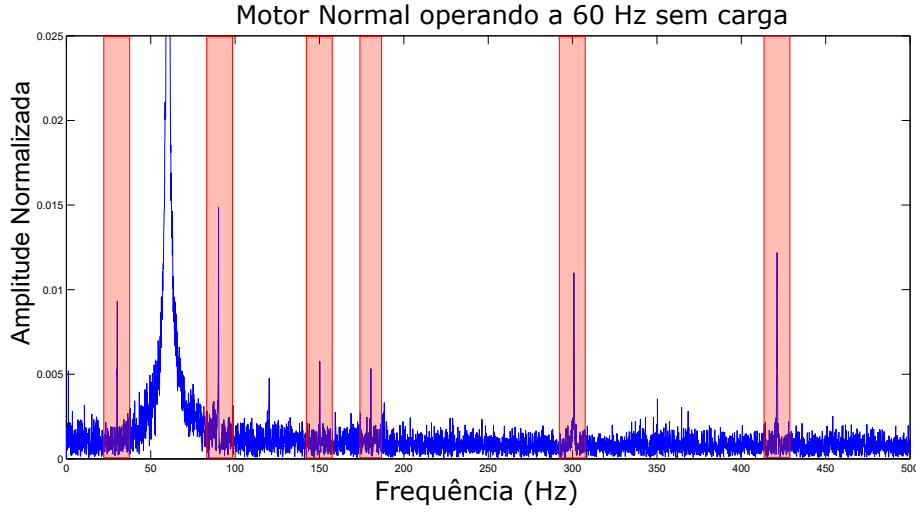


Fonte: O autor

Esses 100.000 pontos são submetidos à Transformada Rápida de Fourier (*Fast Fourier Transform*, FFT) onde os sinais são transformados para o domínio da frequência para que seja possível a análise de assinatura da corrente e a seleção dos atributos. O espectro de frequência é retornado na faixa entre 0 Hz e 5 kHz, com a resolução de 0,1 Hz. Os atributos selecionados para detecção de falha, que são as entradas da RNA, são derivados a partir da componente fundamental (f) de operação do motor com os valores de: $0,5f$, $1,5f$, $2,5f$, $3f$, $5f$ e $7f$. A escolha dessas componentes é baseada no trabalho de Penman et al. (1994) e através de análise estatística apresentada em Oliveira (2014, pág. 64). São exemplificados na Figura 9 os valores obtidos após a FFT do mesmo sinal mostrado na Figura 8.

Observando a Figura 9, é possível ver o valor da componente fundamental de frequência (60,2Hz) e as outras componentes de interesse, que são utilizadas para o diagnóstico do motor. Um detalhe importante que será mais explorado em capítulos posteriores é a resolução no espectro da frequência (Figura 9). A resolução é o intervalo dos valores que se pode obter no espectro de frequência. Como dito anteriormente, para aquisições de 10 segundos a 10 kHz, a resolução é de 0,1 Hz. Após o processo de FFT, a faixa de valores retornados encontra-se entre 0 a 5 kHz a cada 0,1 Hz. Essa resolução está relacionada à taxa de aquisição e o número total de pontos aquisitados. A equação 4.1

Figura 9 – Valores da fundamental e das componentes de frequência obtidas após FFT



Fonte: O autor

é utilizada para obter a resolução de acordo com a configuração de aquisição dos sinais, como visto a seguir.

$$r = \frac{f_{max}}{Np} \quad (4.1)$$

em que:

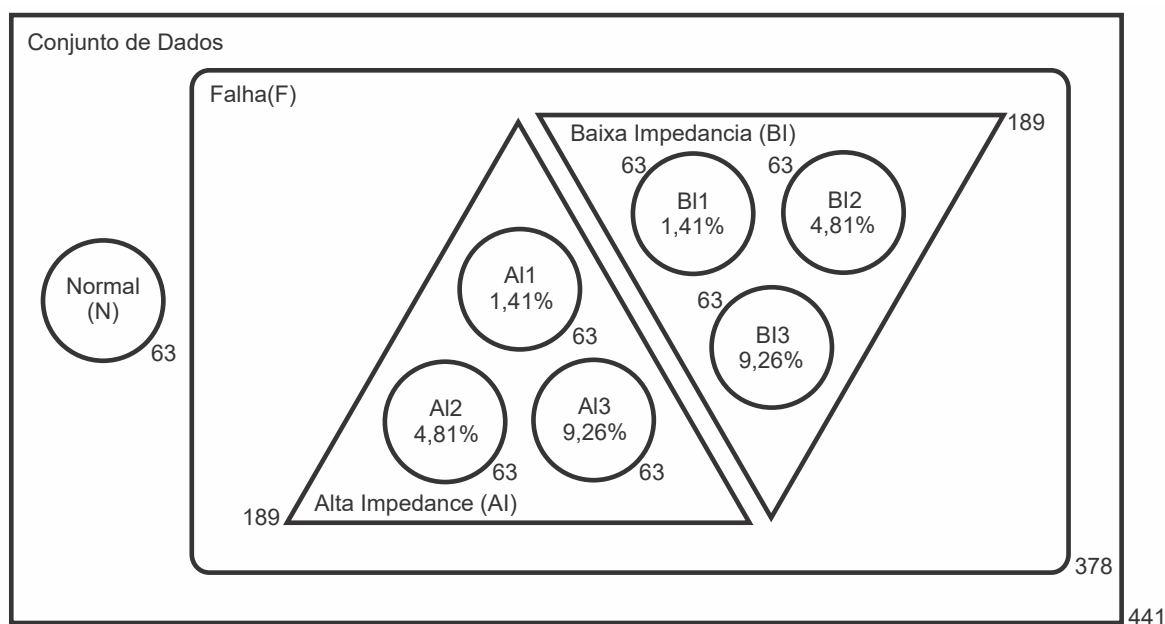
- f_{max} é a frequência de Nyquist, a máxima componente do espectro da frequência, obtido simplesmente dividido a taxa de aquisição por 2.
- Np é o número de componentes obtidas, obtido por $N_s/2$.
- N_s é o número de pontos originalmente aquisitados.

Ainda na Figura 9, o estudo e a observação da resolução é importante pois a frequência da fundamental não é exatamente 60 Hz, assim como as componentes de frequência que derivam da mesma. Logo, os valores utilizados nos treinamentos são aproximações da fundamental e das componentes. Por isso que quanto mais preciso for o espectro, mais os dados são representativos na tentativa de detectar a falha.

4.2 Conjunto de Dados

É exibida na Figura 10 a organização dos dados e as diferentes classes formadas pelas amostras adquiridas durante a operação do motor sem falha e em 6 combinações das falhas. Os números próximos a cada forma geométrica representam o número total de amostras disponíveis no conjunto de dados.

Figura 10 – Diagrama de composição do conjunto de dados



Fonte: O autor

Ao observar o diagrama da Figura 10, é possível perceber que o problema pode ser tratado de diferentes formas. Trabalhos anteriores (COELHO; MEDEIROS, 2013; OLIVEIRA; PONTES; MEDEIROS, 2013; COELHO et al., 2014) fazem uso dos dados de uma forma binária, ou seja, separando os dados em duas classes: normal ou com falha. Nessa abordagem, os classificadores são desenvolvidos desconsiderando que tipo ou nível de falha o motor apresentava, considerando todas as falhas como uma única classe de “falha”.

Outra maneira de separar os dados de forma mais detalhada, quanto a condição do motor, é tentando identificar quando o motor está operando sem falha (N), com uma falha inicial (AI) ou quando está apresentando uma falha mais severa (BI), sendo esse um classificador ternário.

O problema pode ser tratado também na tentativa de identificação da extensão da falha, ou seja, identificando a quantidade de espiras em curto circuito (1,41%, 4,81% ou 9,26%) e não a intensidade (AI ou BI) da falha. Nesse caso, o classificador é desenvolvido tentando identificar 4 tipos de classe: Normal, Nível 1 de falha (AI1&BI1), Nível 2 de falha (AI2&BI2) e Nível 3 de falha (AI3&BI3). De forma mais completa e detalhada, todas as 7 classes podem ser utilizadas para a treinamento de classificadores mais complexos. Em Vieira, Medeiros e Silva (2016) é realizado um estudo sobre essas diferentes formas de separação dos dados.

4.3 Desenvolvimento dos Classificadores Neurais

Neste seção, o processo de elaboração dos classificadores neurais é detalhado. É importante observar que esse trabalho é baseado na pesquisa descrita em Oliveira (2014), onde todo o processo de definição do modelo e parâmetros do classificador é desenvolvido. Neste trabalho, procura-se propor uma evolução em termos de diagnóstico do classificador binário proposto em trabalhos anteriores. Para isso, uma nova forma de classificação é avaliada, que tenta diagnosticar as falhas mais detalhadamente.

4.4 Treinamentos Iniciais

Após o processo de definição de modelo descrito por Oliveira (2014), os parâmetros como descritos na Tabela 2 são utilizados.

Tabela 2 – Parâmetros do modelo para RNA

Parâmetro	Valor
Rede	6-5-1
Entrada da Rede	0,5f, 1,5f, 2,5f, 3f, 5f, 7f
Nº Máx. de épocas	5000
Critério de parada	Parada prematura (Validação Cruzada)
Função de ativação	Tangente hiperbólica
Taxa de aprendizado	0,009 a 0,004, decaimento exponencial
Taxa de momento	0,8
Normalização	1 - Remoção da média e divisão pelo desvio padrão 2 - Normalização entre os valores -1 e + 1
Rotulação	Com falha: -0,98 Sem falha +0,98
Separação dos Dados	80% para treinamento (onde 10% são usados para validação cruzada) Restante pra teste

Fonte: O autor

Nos treinamentos descritos a partir daqui, o conjunto contém amostras de corrente coletadas das três fases. Isso não ocorre nos trabalhos anteriores relacionados à classificação do mesmo conjunto de dados que trabalharam somente com amostras de corrente e duas fases, onde uma das fases está conectada diretamente à espira onde a falha é emulada e outra fase não. Mais detalhes sobre o impacto dessa abordagem e possíveis formas de separação para classificação dos padrões pode ser visto em Vieira, Medeiros e Silva (2016).

Com os parâmetros definidos na Tabela 2, dois projetos de classificadores são elaborados onde a diferença de cada um está na forma de classificar os dados, tal que:

- Projeto 1: os dados são separados em duas classes, motor em condições normais de operação e em condições de falha, sendo a partir daqui referenciado como classificador binário.
- Projeto 2: os dados são separados em 3 classes, motor em condições normais de operação, falha de alta impedância e falha de baixa impedância, sendo esse um classificador que busca identificar a intensidade da falha. A partir daqui, será referenciado como classificador ternário.

Em todas as realizações, 70% dos dados normais (44 amostras) são utilizados para treinamento, 10% para validação cruzada (6 amostras) e o restante para teste da rede (13 amostras). Ao se tratar das amostras com falha, o mesmo valor é utilizado para os conjuntos de treinamento (44 amostras) e validação (6 amostras) para cada classe de falhas relativas do projeto, sendo o restante utilizado para teste. A Tabela 3 detalha a separação de dados em cada projeto.

Tabela 3 – Quantidade de amostras nos conjuntos de Treinamento, Validação e Teste para treinamento das RNAS nos diferentes projetos

Projeto	Conjunto de Treinamento	Conjunto de Validação	Conjunto de Teste
1 - N, F	44 Normais 44 Falha	6 Normais 6 Falha	13 Normais 328 Falha
2 - N, AI, BI	44 Normais 44 AI 44 BI	6 Normais 6 AI 6 BI	13 Normais 139 AI 139 BI

Fonte: O autor

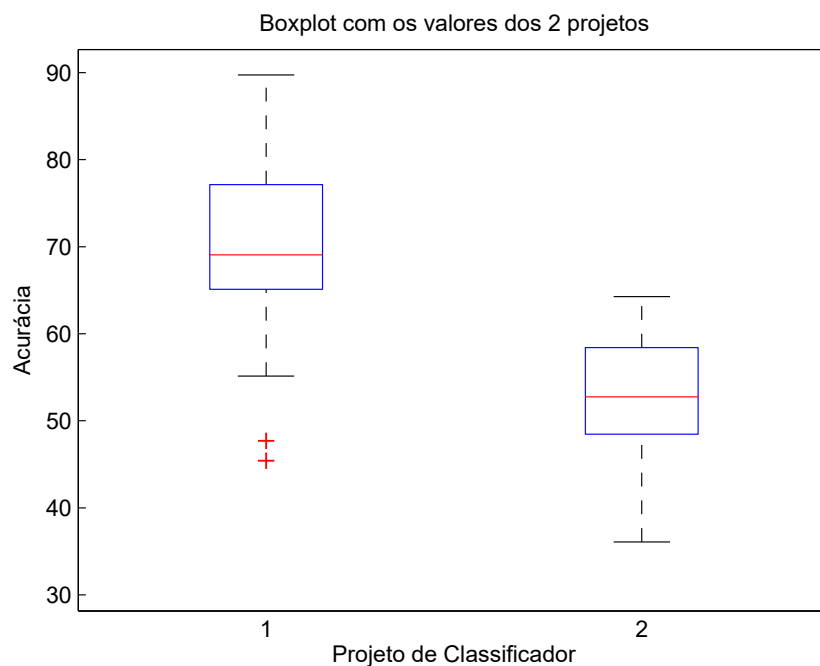
Essa forma de separação é adotada na tentativa de atender uma restrição determinante para o protótipo embarcado em tempo real: é muito importante que o classificador evite falsos positivos. Conforme indicado em Oliveira (2014), e em testes preliminares do autor, a separação aleatória dos conjuntos favorece a acurácia final mas prejudica a detecção do conjunto de amostras normais.

Outro ponto a ser registrado, é que dentro dos conjuntos de falha dos projetos busca-se representar igualmente as subclasses das falhas. Logo, por exemplo, em todos os treinamentos no projeto que classifica a falha pela sua intensidade (AI e BI), há uma distribuição igualitária de amostras de diferentes extensões de falha (falha em 1,41%, 4,81% e 9,26% das espiras). Representando isso em valores exibidos na Tabela 3, das 44 amostras de treinamento, aproximadamente, um terço das amostras contém falhas em 1,41%, outro terço em 4,81% e o restante das amostras representando falhas em 9,26% das espiras. A mesma ideia de proporcionalidade se repete nos outros conjuntos de validação e

teste, assim como no classificador binário.

São efetuadas 50 realizações para cada projeto, em que os resultados do conjunto de teste podem ser vistos no diagrama de caixa da Figura 11. Na Tabela 4 são mostrados os valores de acurácia média (AM), desvio padrão (σ), e erro quadrático médio (EQM) para os conjuntos de treinamento (tr), validação cruzada (vl) e teste (ts). A acurácia média normal (AMn) indica a acurácia somente das amostras normais, sendo esse um valor importante a ser avaliado por motivos já discutidos anteriormente.

Figura 11 – Diagrama de caixa com os valores dos 50 treinamentos de cada projeto de classificador



Fonte: O autor

Os resultados apresentados em Oliveira (2014) dos classificadores binários, são um pouco inferiores em comparação aos exibidos na Tabela 4. Isso pode ser justificado com a adição dos dados adquiridos da corrente da fase que, até então, não havia sido utilizado em trabalhos relacionados anteriores.

4.5 Conclusão

Este capítulo descreve o processo de aquisição dos dados, originalmente feito por Oliveira (2014). O modelo neural definido para a classificação binária dos dados

Tabela 4 – Resultados obtidos nos treinamentos dos 2 projetos de classificadores

Conjunto		Projeto de Classificador	
		Binário	Ternário
TR	Acm (%)	82,64 \pm 8,22	67,74 \pm 6,14
	Amn (%)	85,68 \pm 14,81	75,50 \pm 17,24
	eqm	0,0913	0,2639
VL	Acm (%)	86,00 \pm 10,71	65,56 \pm 9,46
	Amn (%)	90,67 \pm 16,32	74,00 \pm 17,00
	eqm	0,1071	0,2428
TS	Acm (%)	68,96 \pm 11,48	52,84 \pm 6,86
	Amn (%)	80,00 \pm 16,61	65,54 \pm 17,63
	eqm	0,3135	0,4171

Fonte: O autor

também desenvolvido por Oliveira (2014), onde o mesmo modelo é testado para uma classificação ternária, onde os resultados são exibidos.

Observando os resultados, projeto 2 (classificador ternário) é escolhido para dar continuidade ao processo de prototipação. Mesmo com resultados modestos, a análise a ser feita a partir daqui é sobre o impacto das adaptações necessárias no classificador e na configuração de aquisição de dados para desenvolvimento do projeto embarcado. Utilizando os classificadores ternários, a avaliação é feita sobre até onde é aplicável alterar a configuração de aquisição de dados (como taxa de amostragem e período de aquisição) e das RNAs (precisão numérica, função de ativação) sem perda significativa dos resultados obtidos originalmente. Definir um modelo ideal para a nova forma de classificação dos dados, nessa etapa do projeto, se mostra uma opção inviável. A configuração atual para obtenção dos sinais (10 kHz a 10 segundos) é impraticável, principalmente nas plataformas de baixo custo escolhidas para prototipação. Ao definir uma configuração de aquisição de sinais com a qual seja possível desenvolver o protótipo embarcado, um modelo ideal para a nova forma de classificação dos dados deve ser definido.

Em adição ao citado, no trabalho de Vieira, Medeiros e Silva (2016) é indicado que mesmo que o classificador ternário, que classifica N, AI e BI, diagnosticando a intensidade da falha, não consiga resultados tão bons quanto o dos classificadores binários, ele ainda se mostra uma metodologia viável por conseguir identificar as falhas mais severas. Resumidamente, a maioria dos erros ocorrem em falhas pouco severas (AI) e presentes em poucas espiras (1,41% das voltas). A natural evolução da falha pode ocorrer durante minutos ou até horas, e a partir de monitoramentos constantes feitos pelo sistema, pode ser detectada antes de atingir um estado degradativo do bobinamento.

5 Adequações do conjunto de dados e do classificador neural para prototipação dos projetos

Neste capítulo, descreve-se os procedimentos adotados para reduzir a necessidade de recursos computacionais na plataforma embarcada. As explorações das hipóteses para atingir tais reduções ocorrem em quatro graus do projeto:

- na taxa de amostragem de leitura da corrente do motor;
- no tamanho da amostra (quantidade de pontos) e a influência desta quantidade na precisão dos valores que compõem as entradas da rede;
- na precisão numérica dos pesos e dos atributos de entrada da rede;
- a avaliação de formas alternativas para implementação da função de ativação dos neurônios da rede neural.

No projeto explorado nesta dissertação, as seguintes etapas são necessárias, com as quais tais hipóteses supracitadas estão relacionadas:

- digitalização dos dados, através de um conversor analógico-digital, sendo essa etapa relacionada a taxa de amostragem;
- pré-processamento dos dados e extração dos atributos, onde essa etapa está relacionada ao tamanho das amostras;
- execução da rede neural, relacionado a precisão numérica dos dados e da função de ativação.

Os impactos das alterações nos quatro graus do projeto, são comparados com os resultados obtidos com os classificadores treinados com o conjunto original dos dados. Os treinamentos efetuados nesse capítulo, a não ser quando indicado, utilizam o mesmo modelo neural já definido em treinamentos anteriores (conforme mostrado na Tabela 2, Capítulo 4), com 50 realizações para cada modelo, com uma única alteração nos números de neurônios na camada oculta (7 ao invés de 5), uma vez que o aumento dos neurônios se mostrou um melhor modelo para classificação.

Nesse capítulo, os resultados dos novos treinamentos são avaliados da seguinte maneira. Quando apontado a acurácia do classificador "ternário", os resultados são para as classificações dos dados em normal (motor sem falha), alta impedância (AI, falha leve) e baixa impedância (BI, falha mais severa); em "binário", é utilizado o mesmo classificador com uma mudança na interpretação dos resultados. Nesse caso, os erros entre classes de falha (AI e BI) são ignorados, considerando erro de classificação somente

quando uma mostra normal é dita como falha (classe AI ou BI) ou quando uma falha (AI ou BI) é classificada como normal. Essa abordagem apresentou melhores resultados do que o classificador binário. Os resultados para "normal", apontam os valores obtidos ao apresentar somente amostras de condição normal (sem falha) para os classificadores. Como já discutido anteriormente, essa avaliação é importante pois um classificador que não aponte falhas quando o motor opera normalmente, é um dos requisitos do projeto. As colunas 'EQM' representam os valores do erro quadrático médio explicado no Capítulo 2. Os resultados são calculados utilizando o conjunto de teste.

5.1 Impacto da taxa de amostragem sobre a acurácia do classificador

Nos ensaios para aquisição dos dados para treinamento, as amostras de corrente são coletadas a uma taxa de amostragem de 10 kHz durante 10 segundos, gerando 100.000 pontos para cada amostra por fase do motor. Para o projeto embarcado, isso é um problema ao se considerar a memória necessária para armazenar tais números para cada amostra coletada em plataformas com baixa capacidade de memória.

Conforme indicado em Oliveira (2014, pág. 52), as componentes de frequência de interesse situam-se dentro da faixa de 0 a 499 Hz, logo a taxa de amostragem mínima para aquisição de dados é de 1 kHz, segundo o teorema de Nyquist (NYQUIST, 1928). Com o intuito de reduzir a quantidade de memória necessária para armazenamento do espectro de frequência, novos conjuntos de dados com menor taxa de amostragem são gerados para treinamento das RNAs, a partir dos dados originais adquiridos nos ensaios. Como exemplo, para obter uma taxa de amostragem de 5 kHz a partir de uma aquisição realizada com uma taxa de amostragem de 10 kHz basta compor o novo conjunto com o descarte de uma amostra sim e outra não. Para 2,5 kHz, a cada quatro pontos, somente um é utilizado, e assim sucessivamente. A Tabela 5 contém o resumo do processo e das conversões realizadas, assim como o período de aquisição dos pontos, número total de pontos após a conversão a partir do conjunto original e a resolução das amostras dos novos conjuntos no domínio da frequência. Para ilustrar o processo de forma visual, a Figura 12 exibe o processo de conversão.

Com os novos conjuntos de dados, novos treinamentos são feitos. É ilustrado na Figura 13 os diagramas de caixa dos classificadores binários e ternários dos treinamentos efetuados com cada um dos novos conjuntos de dados.

Como mais uma forma de validar a abordagem e os novos conjuntos de dados, utiliza-se os primeiros classificadores treinados com dados obtidos a partir da taxa de amostragem de 10 kHz a 10 segundos para validar as mesmas amostras dos novos conjuntos obtidos a taxas de amostragem menores. Ou seja, as amostras dos novos conjuntos em

Tabela 5 – Informações dos novos conjuntos de dados com taxa de amostragem reduzida

Taxa de Amostragem	Período de Aquisição	Valores Considerados	Número de Pontos	Resolução (Domínio da Frequência)
10 kHz	10 segundos	Todos	100.000	0,1 Hz
5 kHz	10 segundos	1 a cada 2	50.000	0,1 Hz
2,5 kHz	10 segundos	1 a cada 4	25.000	0,1 Hz
2 kHz	10 segundos	1 a cada 5	20.000	0,1 Hz
1,25 kHz	10 segundos	1 a cada 8	12.500	0,1 Hz
1 kHz	10 segundos	1 a cada 10	10.000	0,1 Hz

Fonte: O autor

Figura 12 – Novos conjuntos de dados com taxa de amostragem reduzida - Representação Gráfica

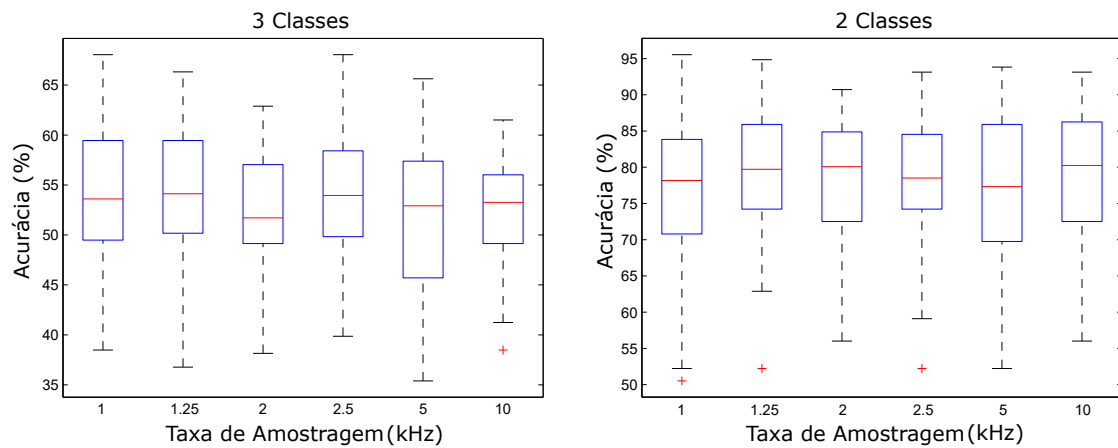
<p>Conjunto Original 10 kHz: $\{x_1, x_2, x_3, x_4, x_5 \dots x_{99.996}, x_{99.997}, x_{99.998}, x_{99.999}, x_{100.000}\}$</p> <p>Conjunto Reduzido 5 kHz: $\{x_1, x_3, x_5, x_7, x_9 \dots x_{99.991}, x_{99.993}, x_{99.995}, x_{99.997}, x_{99.999}\}$</p> <p>Conjunto Reduzido 2,5 kHz: $\{x_1, x_5, x_9, x_{13}, x_{17} \dots x_{99.981}, x_{99.985}, x_{99.989}, x_{99.993}, x_{99.997}\}$</p> <p>Conjunto Reduzido 2 kHz: $\{x_1, x_6, x_{11}, x_{16}, x_{21} \dots x_{99.976}, x_{99.981}, x_{99.986}, x_{99.991}, x_{99.996}\}$</p> <p>Conjunto Reduzido 1,25 kHz: $\{x_1, x_9, x_{17}, x_{25}, x_{33} \dots x_{99.961}, x_{99.969}, x_{99.977}, x_{99.985}, x_{99.993}\}$</p> <p>Conjunto Reduzido 1 kHz: $\{x_1, x_{11}, x_{21}, x_{31}, x_{41} \dots x_{99.951}, x_{99.961}, x_{99.971}, x_{99.981}, x_{99.991}\}$</p>
--

Fonte: O autor

menores taxas de amostragem (5 kHz, 2,5 kHz, 2 kHz, 1,25 kHz e 1 kHz) são submetidos aos classificadores treinados com o conjunto de dados original (10 kHz). Uma vez que as componentes no domínio da frequência não são alteradas, acredita-se que não há mudanças drásticas na acurácia dos classificadores. Os resultados podem ser observados (junto ao resultado obtido com o conjunto original de dados) na Tabela 6

Logo, a partir dos resultados aproximados exibidos na Figura 13, os valores obtidos

Figura 13 – Diagrama de caixa dos classificadores treinados com amostras coletadas em diferentes taxas de amostragem



Fonte: O autor

Tabela 6 – Resultados obtidos pelos classificadores treinados com o conjunto original (10 kHz) testando amostras coletadas em diferentes taxas de amostragem

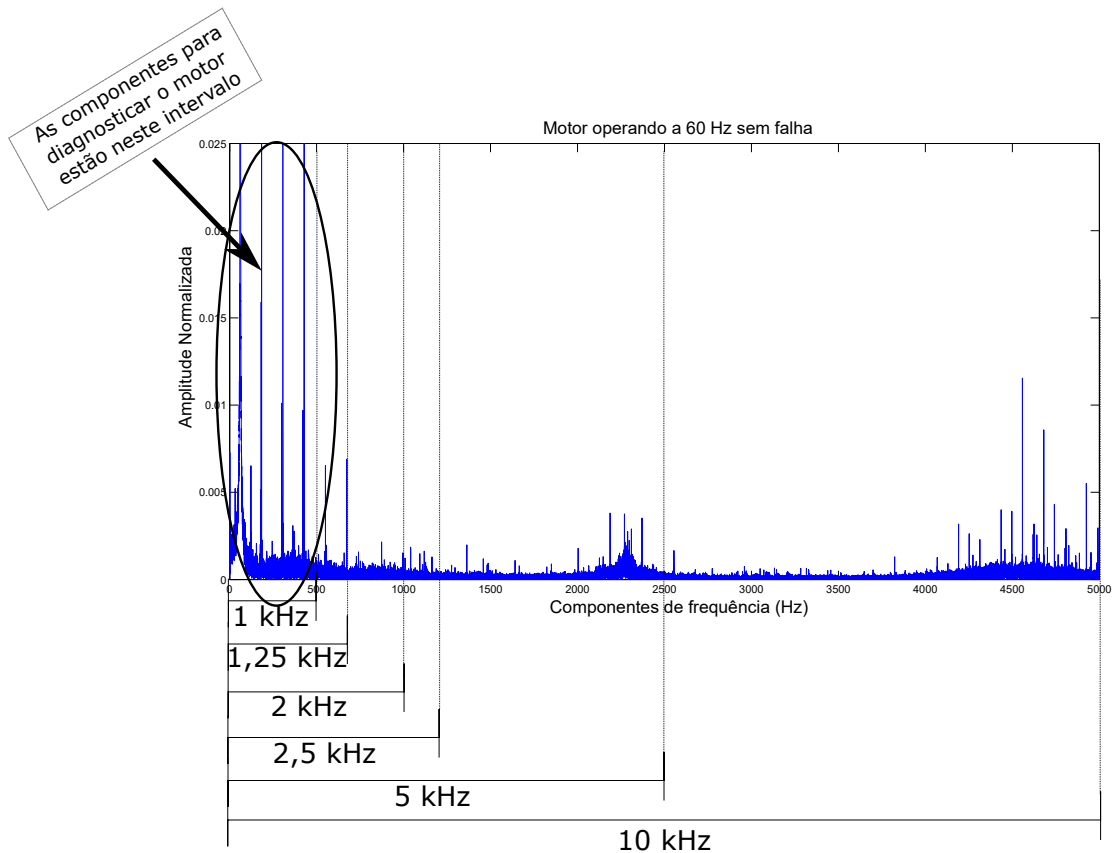
Taxa de Amostragem (kHz)	Resolução (Hz) (Domínio da Frequência)	Acurácia Média (%) / Desvio Padrão (σ)			EQM
		Ternário	Binário	Normal	
10	0,1	53,40 \pm 7,70	77,13 \pm 11,14	68,00 \pm 18,91	0,3471
5	0,1	53,56 \pm 7,57	77,36 \pm 11,16	67,69 \pm 19,23	0,3809
2,5	0,1	52,83 \pm 7,51	76,32 \pm 11,33	69,85 \pm 18,96	0,3947
2	0,1	53,79 \pm 7,45	77,83 \pm 11,18	65,08 \pm 17,67	0,3946
1,25	0,1	52,47 \pm 7,70	76,93 \pm 11,47	65,54 \pm 19,98	0,3994
1	0,1	53,39 \pm 7,56	78,27 \pm 11,26	66,77 \pm 18,24	0,4029

Fonte: O autor

exibidos na Tabela 6, e com o teorema de Nyquist, é possível concluir que qualquer uma das taxas de amostragem testadas são igualmente representativas para treinamento e validação da rede. A Figura 14 mostra de forma gráfica os intervalos das componentes visíveis em diferentes taxas de amostragem, reafirmando de forma gráfica as conclusões e resultados obtidos.

Tal redução é importante, pois, ao reduzir a taxa de 10 kHz para 1 kHz, há uma redução de 90% dos dados necessários para pré-processamento, o que diminui o tamanho de memória necessária para armazenar os dados no momento da criação do protótipo embarcado. A redução na taxa de amostragem também tende a reduzir o custo do conversor Analógico-Digital usado para aquisição dos dados. Entretanto, nesta faixa de frequência esse benefício é imperceptível, uma vez que a diferença de 10 kHz para 1 kHz é pouco

Figura 14 – Intervalos das componentes visíveis em diferentes taxas de amostragem



Fonte: O autor

significativa ao se analisar o conversor Analógico-Digital necessário para adquirir ambas faixas de frequência.

5.2 Impacto do período de aquisição dos dados sobre a acurácia do classificador

O intervalo de coleta das amostras de 10 segundos também não se mostra ideal na tentativa de desenvolvimento do protótipo. É interessante que o protótipo seja capaz de diagnosticar a situação do motor em menor intervalo de tempo tão pequeno quanto possível. Além da relativa demora de resposta (diagnósticos a cada 10 segundos) quanto maior o período de aquisição dos sinais, maior também o espaço na memória necessário para armazenamento dos dados adquiridos. De forma similar à seção anterior, novos conjuntos de dados foram criados a partir do conjunto original. Dessa vez, manteve-se a taxa de amostragem (10 kHz) mas reduziu-se o período de aquisição.

Para representar esse menor período de aquisição, parte dos sinais originais são

excluídos na tentativa de simular aquisições por menos tempo, gerando assim novos conjuntos de dados. Para simular uma coleta de cinco segundos, por exemplo, dos 100.000 pontos originais, coletados a 10 segundos, metade desses pontos são utilizados. Para simular uma coleta de 2 segundos, um quinto dos valores são utilizados. Essa ideia de proporcionalidade se repete em outros períodos de aquisição, criando assim novos conjuntos com intervalos menores de coleta (5, 4, 3, 2, 1,25, 1, 0,5 e 0,1 segundo). Com os conjuntos criados, de forma similar a já apresentada ao fim da seção anterior, os classificadores treinados com os dados aquisitados a 10 kHz a 10 segundos são utilizados para validar comparativamente os novos conjuntos. O intuito desse teste é verificar se ainda é possível utilizar-se das RNAs treinadas com o conjunto de dados original para classificar dados coletados em outra configuração de aquisição. Os resultados podem ser observados na Tabela 7, onde é possível ver o período de aquisição dos sinais em segundos, o intervalo de coleta em comparação ao do conjunto de dados original de 10 segundos, a resolução no domínio da frequência e as acurácias médias e desvio padrão para duas e três classes, assim como para as amostras normais.

Tabela 7 – Resultados das validações dos conjuntos de amostras de menor período de aquisição em classificadores treinados com os dados aquisitados a 10 kHz durante 10 segundos

Período de Aquisição (segundos)	Resolução (Hz) (Domínio da Frequência)	Acurácia Média (%) / Desvio Padrão (σ)		
		Ternário	Binário	Normal
10	0,1	58,83 \pm 6,06	78,34 \pm 8,61	74,79 \pm 16,16
5	0,2	53,53 \pm 5,38	73,07 \pm 8,93	75,94 \pm 11,69
4	0,25	52,63 \pm 4,44	71,94 \pm 7,36	72,98 \pm 14,84
3	0,33	44,80 \pm 3,88	61,31 \pm 6,58	83,30 \pm 9,92
2,5	0,4	46,28 \pm 4,01	66,94 \pm 8,60	59,59 \pm 15,49
2	0,5	44,95 \pm 2,97	62,59 \pm 4,99	66,16 \pm 12,99
1,25	0,8	37,99 \pm 3,17	53,39 \pm 6,47	60,89 \pm 9,90
1	1	43,83 \pm 5,02	62,66 \pm 9,25	40,19 \pm 16,35
0,5	2	43,60 \pm 2,34	71,21 \pm 4,33	21,75 \pm 8,36
0,1	10	39,07 \pm 2,35	74,36 \pm 5,96	14,32 \pm 10,46

Fonte: O autor

É possível ver a constante diminuição de acurácia a medida que a resolução no domínio da frequência piora, invalidando os classificadores anteriores. Uma vez que os classificadores originais não são mais eficientes em classificar os novos dados, novos classificadores são desenvolvidos com os novos conjuntos de intervalo menor numa tentativa de melhorar os resultados. Os valores dos novos treinamentos podem ser vistos na Tabela 8, que segue a mesma organização da Tabela 7.

É visível a queda da acurácia dos treinamentos ao reduzir o intervalo de aquisição

Tabela 8 – Resultados obtidos com os novos classificadores treinados com conjuntos criados com intervalos menores de aquisição

Período de Aquisição (segundos)	Resolução (Hz) (Domínio da Frequência)	Acurácia Média (%) / Desvio Padrão (σ)		
		Ternário	Binário	Normal
10	0,1	53,40 \pm 7,70	77,13 \pm 11,14	68,00 \pm 18,91
5	0,2	53,66 \pm 6,92	78,67 \pm 10,97	65,38 \pm 17,87
4	0,25	49,26 \pm 6,18	72,04 \pm 10,58	68,85 \pm 19,08
3	0,33	49,44 \pm 5,58	74,03 \pm 9,55	70,00 \pm 18,45
2,5	0,4	46,32 \pm 6,87	69,65 \pm 12,19	59,08 \pm 20,40
2	0,5	44,89 \pm 6,30	66,63 \pm 10,83	49,85 \pm 24,67
1,25	0,8	42,56 \pm 7,43	63,93 \pm 14,05	48,46 \pm 23,27
1	1	41,11 \pm 9,74	59,99 \pm 16,99	52,92 \pm 21,75
0,5	2	42,91 \pm 8,37	65,40 \pm 14,44	39,54 \pm 22,09
0,1	10	33,29 \pm 10,02	65,87 \pm 20,72	26,15 \pm 19,45

Fonte: O autor

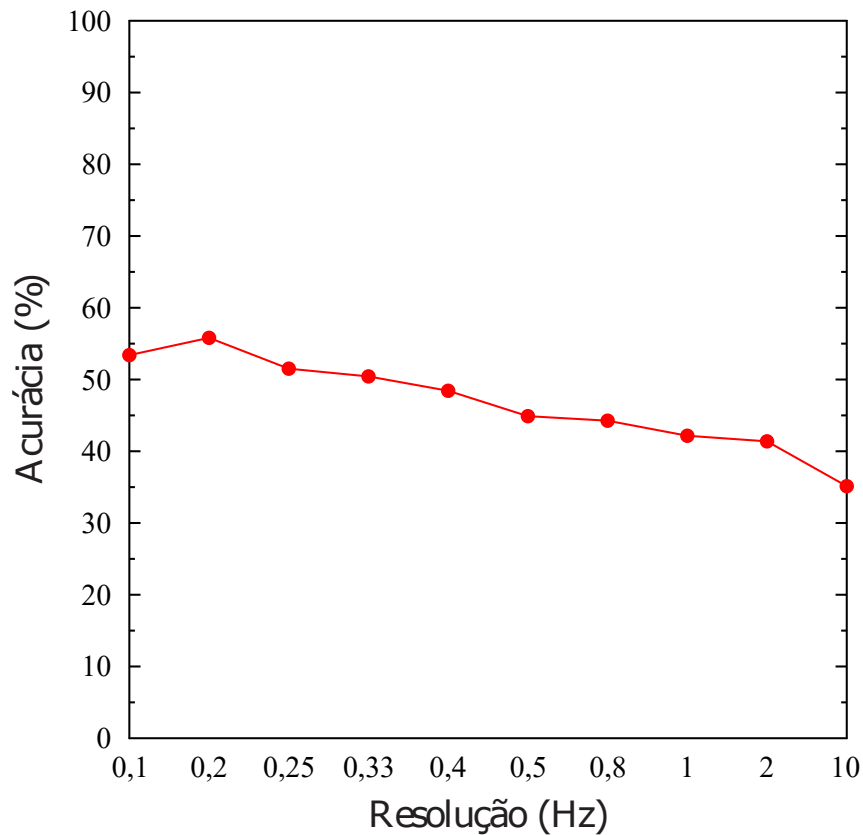
dos sinais. Contudo, esses testes são importantes para verificação de até onde é possível diminuir a resolução sem redução significativa da acurácia. O gráfico na Figura 15 exibe a relação da acurácia média (para o classificador ternário) com a resolução no domínio da frequência.

É possível estabelecer algumas considerações após esses testes preliminares com os novos conjuntos de dados. Há uma relação direta da acurácia dos classificadores com a resolução no domínio da frequência. Logo, em adição a equação 4.1, é possível concluir que:

- a resolução (r) se dá pela equação 4.1;
- a taxa de aquisição está relacionada ao conversor analógico-digital disponível na plataforma;
- o número de amostras é limitado pela quantidade de memória da plataforma;
- a resolução é limitada pela acurácia desejada.

Observando a suave mas contínua queda de acurácia da Figura 15, fica difícil inferir qual é o melhor valor de resolução para desenvolvimento dos classificadores. Naturalmente, quanto mais preciso os valores para treinamento, melhor para detecção das falhas. Contudo, tal decisão é limitada pela quantidade de memória disponível da plataforma embarcada. A Figura 16 exibe o impacto da redução do período de aquisição nos valores visíveis das componentes no domínio da frequência. Acima de cada gráfico das componentes, há o período de aquisição dos sinais e resolução no domínio da frequência ao lado. Observa-se que a partir de 3 segundos os sinais começam a se diferenciar de forma significativa do sinal original.

Figura 15 – Acurácia Média para classificador de 3 classes por resolução no domínio da frequência

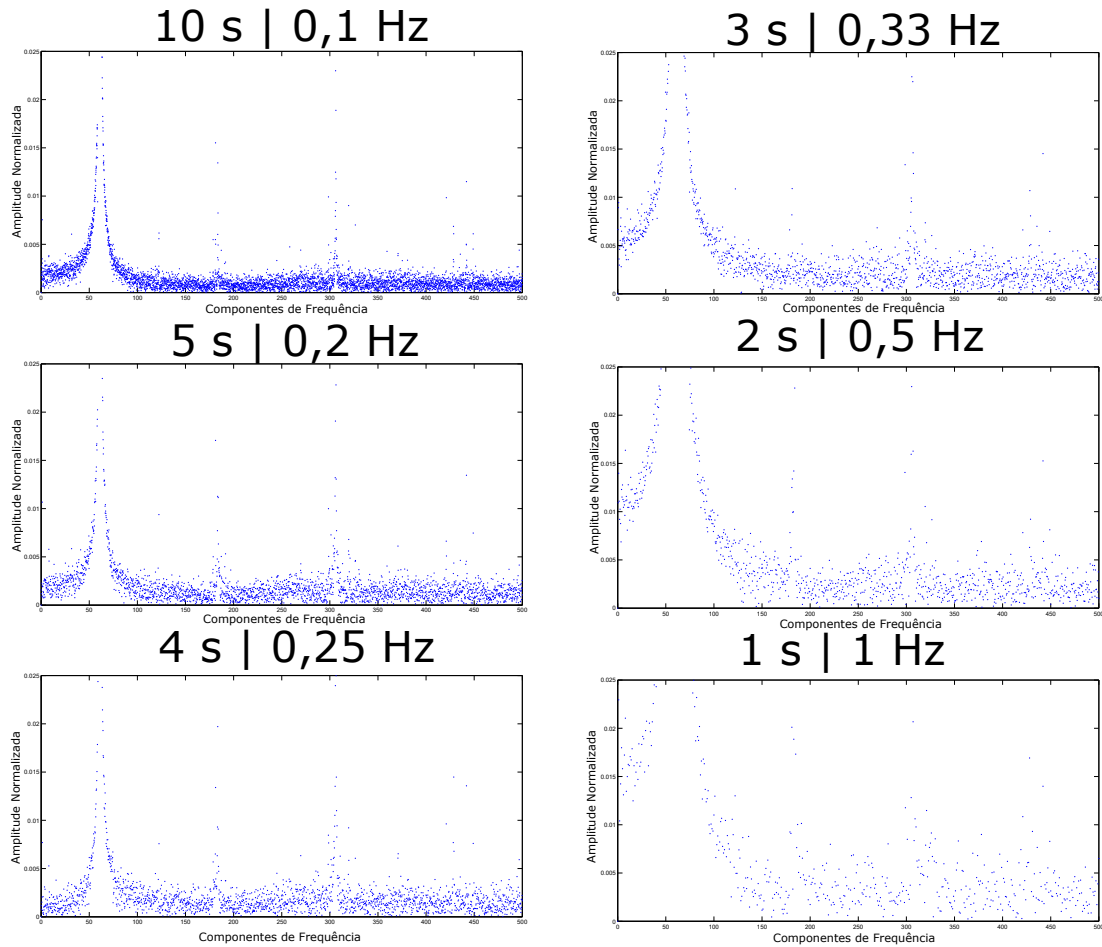


Fonte: O autor

Para tentar contornar a redução da qualidade dos dados, buscou-se verificar se a adição de mais dados, mesmo de pior qualidade, poderia auxiliar na melhoria dos resultados. Para tal, novos testes foram feitos ainda reduzindo o período de coleta mas utilizando todas as partes do conjunto original como diferentes coletas. Como exemplo, dos 100.000 pontos originais, representando 10 kHz a 10 segundos, dois conjuntos de 50.000 pontos são criados para representar dois períodos de coleta de 5 segundos a 10 kHz. A mesma ideia é aplicada em intervalos menores. Os detalhes dos novos conjuntos criados com essa metodologia, é exibido na Tabela 9, onde a coluna 'número de pontos' corresponde ao número de valores adquiridos para criar uma única amostra; 'intervalo de coleta' representa a quantidade de coletas possíveis a partir dos 10 segundos originais; 'quantidade de amostras' agrupa o total de amostras obtidas a partir da nova aquisição de dados e por fim, a resolução no domínio da frequência.

Mais uma vez, novos treinamentos foram realizados com cada um dos 8 conjuntos criados reduzindo o período de coleta exibidos na Tabela 9. Os resultados médios de tais treinamentos são mostrados no gráfico da Figura 17.

Figura 16 – Impacto da redução do período de aquisição nos valores das componentes de frequência



Fonte: O autor

Apesar dos resultados pouco conclusivos, é perceptível que a adição de novos dados, mesmo com a resolução piorada, permitem resultados similares aos obtidos com os dados originais. Acredita-se que apesar da qualidade dos dados cair ao diminuir o tempo de coleta, a maior quantidade de dados dos novos conjuntos comparado ao conjunto original colaboram para um classificador algumas vezes melhor (no caso dos 5 segundos e no de 1,25 segundos para 3 classes) ou de desempenho equiparável (no caso da coleta de 2,5 segundos).

5.3 Discussão acerca das adaptações da captura dos dados

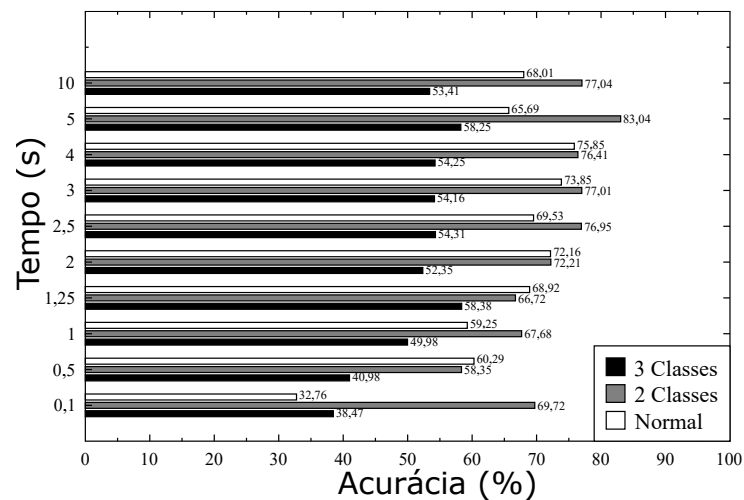
Os ajustes realizados no conjunto de dados e no processo de captura dos sinais são essenciais para o processo de desenvolvimento do protótipo. A redução de taxa de aquisição diminui significativamente o número de pontos necessários para o treinamento das redes

Tabela 9 – Informações dos novos conjuntos de dados criados a partir de menores intervalos de coleta das amostras

Período de Aquisição (segundos)	Número de Pontos	Intervalo de Coleta	Quantidade de Amostras	Resolução (Hz) (Domínio da Frequência)
10	100.000	1	441	0,1
5	50.000	2	882	0,2
4	40.000	2	882	0,25
3	30.000	3	1.323	0,33
2,5	25.000	4	1.764	0.4
1,25	12.500	8	3.528	0,8
1	10.000	10	4.410	1
0,5	5.000	20	8.820	2
0,1	1.000	100	44.100	10

Fonte: O autor

Figura 17 – Acurácia média dos classificadores treinados com conjuntos criados a partir de menores intervalos de coleta dos sinais



Fonte: O autor

neurais. Teste preliminares nas plataformas disponíveis para prototipação indicam que o número máximo de pontos que pode ser armazenados é de aproximadamente 4000 pontos. A configuração de aquisição que atende essa restrição é a de 4 segundos a 1 kHz de taxa de amostragem. Observando os resultados na Figura 17, a coleta de 4 segundos oferece uma boa acurácia ao classificar amostras normais e acurácia aproximada nos classificadores binários e ternários, comparados com os classificadores desenvolvidos com as amostras coletadas durante 10 segundos. Ao escolher esse intervalo, há uma redução de 60 % no

tempo de resposta de diagnóstico, uma vez que a captura original requeria 10 segundos. Junto a redução da taxa de aquisição de 10 kHz para 1 kHz, há uma redução de 96% dos pontos necessários para diagnosticar o motor, sem perda de acurácia comparado aos modelos construídos com o conjunto original. Com essa nova configuração, um novo conjunto de dados é desenvolvido com dois períodos de aquisição de 4 segundos a 1 kHz derivado do conjunto original de 10 kHz a 10 segundos, utilizando as mesmas técnicas já explicadas. O resultado médio dos treinamentos efetuados com esse conjunto é exibido na Tabela 10 onde é possível ver que os resultados são bastante similares aos obtidos com o conjunto original, com o desvio padrão ainda menor, o que define esse modelo mais confiável para essa nova configuração de dados. Dado os resultados, essa configuração de aquisição então é escolhida para desenvolvimento do protótipo.

Tabela 10 – Resultados dos treinamentos efetuados com o conjunto a 1 kHz durante duas coletas de 4 segundos

Período de Aquisição (segundos)	Resolução (Hz) (Domínio da Frequência)	Acurácia Média (%) / Desvio Padrão (σ)		
		Ternário	Binário	Normal
10	0,1	53,40 \pm 7,70	77,13 \pm 11,14	68,00 \pm 18,91
4	0,25	53,45 \pm 4,37	77,31 \pm 7,08	69,96 \pm 13,23

Fonte: O autor

5.4 Alternativas de função de ativação para hardwares dedicados

Funções de ativação não-lineares como tangente hiperbólica e sigmoide são responsáveis pela capacidade dos classificadores neurais de tratar de problemas de classificação com classes não linearmente separáveis. Diversos trabalhos como Deotale e Dole (2014, pág. 2), Braga et al. (2010, pág.1629) e Namin et al. (2009, pág. 2117) apontam os multiplicadores e as funções de ativação com as principais responsáveis pelas dificuldades (seja de implementação, consumo energético e de espaço, precisão, entre outros) em projetos de redes neurais em plataformas reprogramáveis (FPGA).

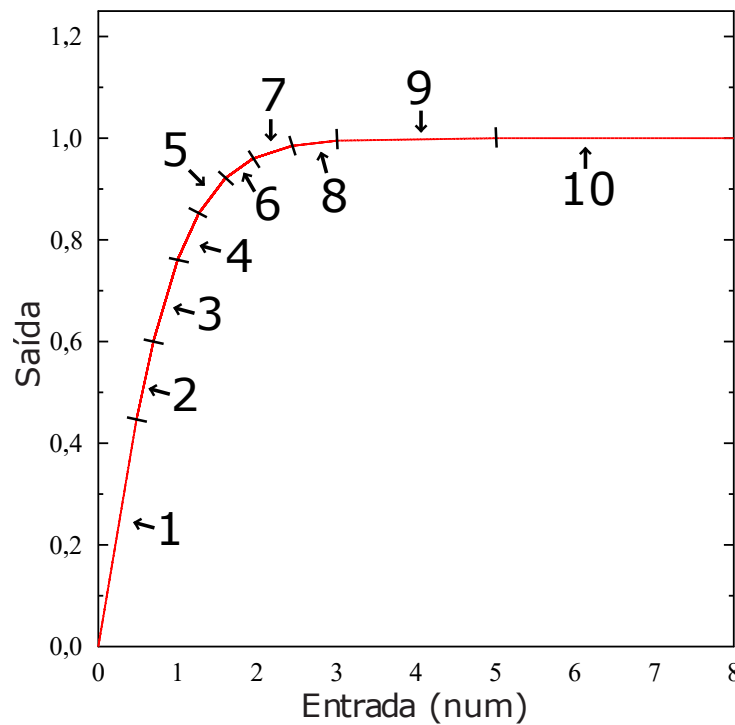
Entre as soluções propostas mais utilizadas, cita-se o frequente uso de *look-up table* (LUT) substituindo as funções de ativação (DEOTALE; DOLE, 2014, pág. 2) (AKLAH; ANDREWS, 2015, pág. 429), memory look up table (MLUT) (KAISTI et al., 2013, pág. 6), métodos de aproximação de função com trechos lineares (BRAGA et al., 2010, pág.1631), utilizando-se de polinômios quadráticos obtidos através de mínimos quadrados (NEDJAH; SILVA; MOURELLE, 2012, pág. 9194), e combinações entre LUTs e aproximações de função com trechos lineares (NAMIN et al., 2009, pág. 2118).

Para o presente trabalho, duas soluções são avaliadas: métodos de aproximação de função com trechos lineares e uma combinação da aproximação linear com LUT, sugerido no trabalho (NAMIN et al., 2009). Neste primeiro momento, as soluções propostas são avaliadas comparando os resultados da tangente hiperbólica original com as propostas no ambiente Matlab, observando somente a acurácia. Outras questões como consumo de memória e tempo de execução são discutidas no capítulo que descreve as implementações na plataforma reprogramável.

5.4.1 Tangente hiperbólica baseada em aproximação de funções (Solução 1)

Para essa proposta, dez retas são utilizadas para aproximar a função de ativação. Para simplificação do projeto, somente o módulo do valor de entrada da função é considerado, com o sinal da entrada replicado na saída. O resultado da aproximação é observado na Figura 18.

Figura 18 – Implementação de tangente hiperbólica utilizando retas



Fonte: O autor

De forma manual, as dez retas são sugeridas tentando atingir a menor diferença possível da função original. As equações das retas foram condensadas na Tabela 11.

Tabela 11 – Equações das retas para aproximação da tangente hiperbólica

Reta	Intervalo da entrada (num)	Saída
1	0 a 0,4806	num * 0,9298
2	0,4806 a 0,6976	(num * 0,7193) + 0,1010
3	0,6976 a 1	(num * 0,5248) + 0,2367
4	1 a 1,264	(num * 0,3431) + 0,4184
5	1,264 a 1,6	(num * 0,2068) + 0,5907
6	1,6 a 1,9401	(num * 0,1114) + 0,7431
7	1,9401 a 2,434	(num * 0,0510) + 0,8605
8	2,434 a 3	(num * 0,0181) + 0,9407
9	3 a 5	(num * 0,0024) + 0,9879
10	Maior que 5	1

Fonte: O autor

5.4.2 Tangente hiperbólica baseada em aproximação de funções e RALUT (Solução 2)

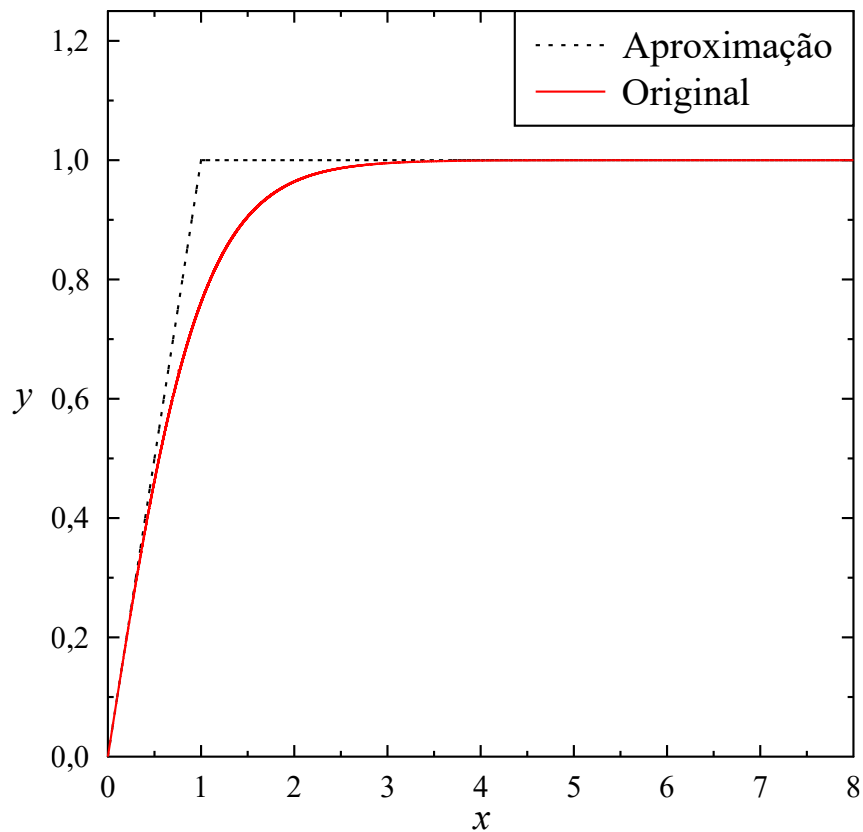
Nessa abordagem baseada em (NAMIN et al., 2009), a função inicialmente é simplificada com o uso de duas retas conforme visto na Figura 19.

Essa aproximação pode ser expressa por:

$$y = \begin{cases} x & 0 \leq x \leq 1, \\ 1 & 1 \leq x \leq 8. \end{cases} \quad (5.1)$$

Após isso, é calculada a diferença entre o valor real da tangente hiperbólica original e o valor aproximado definido pelas retas da equação 5.1 para os valores entre 0 e 8. Essas diferenças entre as saídas das duas funções são armazenadas para desenvolvimento de uma LUT. A ideia é fazer o ajuste da saída sugerida pela aproximação linear (Figura 19) através do erro armazenado na LUT referente ao valor da entrada. Ou seja, dada uma entrada \mathbf{x} , o valor da tangente é calculado conforme indicado na equação 5.1. Após isso, o valor sugerido é ajustado de acordo com o valor de correção presente na LUT. Conforme indicado em Namin et al. (2009, pág. 2118) as principais vantagens dessa abordagem é o não uso de multiplicadores que ocupam significativamente mais espaço de memória e linhas de programação. A abordagem apresentada também consegue gerar LUTs menores em comparação as tradicionais implementações de tangente hiperbólica em LUT sem nenhum tipo de otimização. Isso porque o tipo de memória utilizada é a *Range Addressable LookUp Table* (RALUT). Uma RALUT corresponde a uma LUT onde um intervalo de entradas corresponde a uma única saída. Logo, ao invés de armazenar todos os ajustes necessários para os valores de entrada (neste caso 0 a 8), intervalos que apresentam valor de correção similares são agrupados.

Figura 19 – Tangente hiperbólica aproximada através de 2 retas



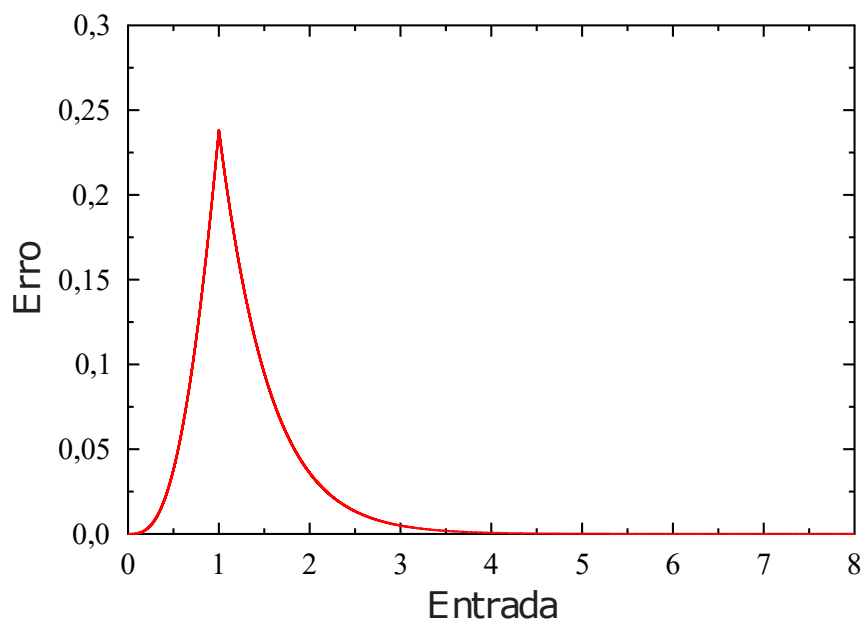
Fonte: O autor

Como modificação à ideia original, busca-se simplificar mais o projeto para conseguir um menor número de ajustes a serem armazenados na RALUT. Para melhor entendimento da lógica aplicada, os erros das duas retas originalmente propostas em relação à função original são plotados no gráfico na Figura 20.

Com o eixo x representando os valores de entrada (0 a 8) e o y representando o valor do erro, é possível perceber uma certa simetria entre os valores de erro. Para fazer benefício dessa observação, e para níveis de teste iniciais, baseado também no trabalho de Namin et al. (2009), define-se que o maior erro que a função proposta pode cometer é do valor de 0,04. Desta forma, busca-se intervalos de entrada onde a média do erro desses intervalos seja de no máximo 0,04. Esses intervalos são as entradas da RALUT e a saída é o erro médio desse intervalo que deve ser subtraído da saída da equação 5.1. A Tabela 12 exemplifica o processo com os valores reais.

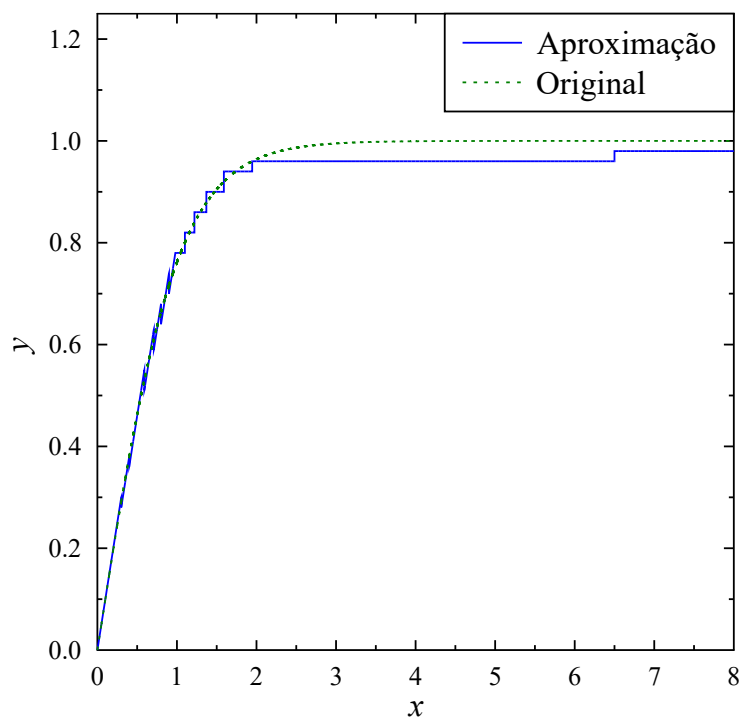
A função é plotada na Figura 21 com o comparativo com a tangente hiperbólica original.

Figura 20 – Erro da aproximação linear em comparação a original entre valores 0 e 8



Fonte: O autor

Figura 21 – Comparativo entre solução 2 e tangente hiperbólica original



Fonte: O autor

Tabela 12 – Representação da RALUT com suas respectivas entradas e saídas

Entradas da RALUT (Intervalos)	Saída da RALUT (erro médio do intervalo)	Saída da Função (y)
1. $0,9 \geq x \leq 1,1$	0,2	$x - 0,2$
2. $0,8 \geq x \leq 0,9$ 3. $1,1 \geq x \leq 1,22$	0,16	$x - 0,16$
4. $0,71 \geq x \leq 0,8$ 5. $1,22 \geq x \leq 1,37$	0,12	$x - 0,12$
6. $0,59 \geq x \leq 0,71$ 7. $1,37 \geq x \leq 1,59$	0,08	$x - 0,08$
8. $0,4 \geq x \leq 0,59$ 9. $1,59 \geq x \leq 1,946$	0,04	$x - 0,04$
10. $0,3 \geq x \leq 0,4$ 11. $1,946 \geq x \leq 6,5$	0,02	$x - 0,02$
Valores Restantes	0	$x - 0$

Fonte: O autor

5.4.3 Comparativo de acurácia das redes neurais utilizando as tangentes hiperbólicas alternativas

Para avaliar as funções propostas, são utilizados os classificadores treinados com o conjuntos de dados formado a partir de amostras obtidas com uma taxa de amostragem de 1 kHz durante 4 segundos. Nesses classificadores já treinados, a função de ativação original do Matlab 2014a é substituída uma vez pela aproximação de função através de retas (solução 1) e depois pelo uso de duas retas mais correção de erro com uma RALUT (solução 2) na saída dos neurônios. As redes são novamente submetidas aos dados dos conjuntos de teste e os resultados comparativos podem ser observados na Tabela 13.

Tabela 13 – Comparativo dos resultados dos classificadores neurais com diferentes implementações de tangente hiperbólica

Função de Ativação	Acurácia Média (%) / Desvio Padrão (σ)		EQM
	Ternário	Binário	
tanh (Matlab)	$53,45 \pm 4,37$	$77,31 \pm 7,08$	0,3800
Solução 1	$53,37 \pm 4,36$	$77,19 \pm 7,79$	0,3791
Solução 2	$54,35 \pm 4,03$	$79,19 \pm 6,58$	0,3746

Fonte: O autor

Os números indicam que as duas alternativas conseguem resultados muito próximos ao original, o que sugere que ambas são soluções válidas para o problema. A escolha de qual solução será utilizada irá depender de qual é a mais viável em termos de implementação

em linguagem de descrição de hardware. Contudo, é importante observar que a solução 2 não utiliza multiplicação como a solução 1, o que já indica seu potencial de uso.

5.5 Impacto da precisão numérica nos pesos sinápticos e entradas da rede neural

Outro aspecto a ser avaliado é a precisão numérica dos valores dos pesos das redes neurais e dos valores das harmônicas que servem como entrada para o classificador. O *software* utilizado para gerar tais valores é o Matlab, que opera com 15 casas decimais. Ao embarcar tais valores em plataformas embarcadas, algumas vezes esses números são convertidos para a base binária. Com esse número de casas decimais, os valores binários ficam muito grandes, aumentando assim a complexidade do projeto e das operações. É desejável então, reduzir o número de casas decimais, com o intuito de atender a limitações de precisões de certas plataformas e até mesmo aumentar a velocidade de computação.

Em busca de tal objetivo, procura-se avaliar, de forma empírica, o impacto da redução gradativa do número de casas decimais na acurácia dos classificadores. O teste é realizado inicialmente sobre os pesos das redes neurais. Uma redução binária é utilizada diminuindo sempre metade das casas decimais até observar uma queda na acurácia do classificador. A partir desse ponto a redução de precisão é feita removendo uma casa decimal por vez. Após os resultados serem armazenados, novamente as casas decimais são reduzidas e o processo se repete até chegar a uma casa decimal. Os resultados dos classificadores treinados com o conjunto de dados com a configuração de aquisição de 1 kHz a 4 segundos, calculados com diferentes precisões numéricas podem ser vistos na Tabela 14. Verifica-se que a alteração dos resultados começa a partir da terceira casa decimal.

Tabela 14 – Resultados obtidos com diferentes precisões numéricas nos pesos dos classificadores

Precisão	Acurácia Média (%) / Desvio Padrão (σ)		EQM
	Ternário	Binário	
16 casas (64 bits)	53,45 \pm 4,37	77,31 \pm 7,08	0,3800
8 casas (32 bits)	53,45 \pm 4,37	77,31 \pm 7,08	0,3800
4 casas (16 bits)	53,45 \pm 4,37	77,31 \pm 7,08	0,3800
3 casas (12 bits)	53,45 \pm 4,37	77,32 \pm 7,08	0,3800
2 casa (8 bits)	53,35 \pm 4,39	77,16 \pm 7,03	0,3802
1 casa (4 bits)	52,51 \pm 4,54	75,59 \pm 7,38	0,3856

Fonte: O autor

De forma similar, a precisão dos valores das amostras é alterada. Utilizando a

mesma metodologia aplicada nos pesos da rede, as casas decimais dos valores de entrada da rede (os padrões) são reduzidas e submetidas às redes com os pesos sem alteração (com as 15 casas decimais originais). O resultado de processo pode ser observado na Tabela 15.

Tabela 15 – Resultados obtidos com diferentes precisões numéricas nas entradas dos classificadores

Precisão	Acurácia Média (%) / Desvio Padrão (σ)		EQM
	Ternário	Binário	
16 casas (64 bits)	53,45 \pm 4,37	77,31 \pm 7,08	0,3800
8 casas (32 bits)	53,45 \pm 4,37	77,31 \pm 7,08	0,3800
4 casas (16 bits)	53,45 \pm 4,38	77,31 \pm 7,08	0,3800
3 casas (12 bits)	53,44 \pm 4,36	77,27 \pm 7,07	0,3801
2 casas (8 bits)	53,42 \pm 4,34	77,18 \pm 7,10	0,3802
1 casa (4 bits)	53,02 \pm 4,26	76,12 \pm 6,87	0,3853

Fonte: O autor

Observa-se que o impacto é maior sobre os pesos da RNA em comparação ao impacto sobre as entradas da rede. Nas entradas da rede, a partir da terceira casa decimal a acurácia começa a cair, mas ainda de forma pouco relevante.

Para trabalhar com números com a mesma precisão decimal, a redução das casas decimais é aplicada simultaneamente nas entradas da rede e os pesos da rede. Os resultados utilizando a mesma metodologia são observados na Tabela 16.

Tabela 16 – Resultados obtidos com diferentes precisões numéricas nas entradas e pesos do classificadores

Precisão	Acurácia Média (%) / Desvio Padrão (σ)		EQM
	Ternário	Binário	
16 casas (64 bits)	53,45 \pm 4,37	77,31 \pm 7,08	0,3800
8 casas (32 bits)	53,45 \pm 4,37	77,31 \pm 7,08	0,3800
4 casas (16 bits)	53,45 \pm 4,38	77,31 \pm 7,08	0,3800
3 casas (12 bits)	53,45 \pm 4,38	77,30 \pm 7,09	0,3800
2 casas (8 bits)	53,30 \pm 4,32	77,14 \pm 7,09	0,3803
1 casa (4 bits)	52,66 \pm 4,45	75,74 \pm 7,55	0,3877

Fonte: O autor

Importante observar que a redução das casas decimais interfere o desempenho de forma desigual em classificadores diferentes dado que observa-se, por exemplo, valores de acurácia iguais mas com desvios padrões diferentes. Quando o classificador a ser embarcado for escolhido, sugere-se observar o impacto das reduções da precisão que pode ser maior ou menor do que em outros classificadores. Fica a cargo do projetista decidir até onde é

viável a redução de casas decimais para atender as restrições do projeto, seja de memória, acurácia ou velocidade de processamento.

5.6 Conclusão

Este capítulo descreve os estudos desenvolvidos para reduzir recursos computacionais na etapa de aquisição e pré-processamento dos sinais utilizados para diagnosticar o MIT, assim como na precisão numérica dos pesos e entradas da rede neural e da função de ativação dos neurônios da rede MLP.

Os resultados indicaram que é possível aquisitar os dados a uma taxa de 1 kHz durante 4 segundos, sem perda de acurácia dos classificadores. A nova configuração para aquisição dos dados reduz em 96% o número de pontos para serem armazenados para diagnóstico da falha e em 60% o tempo de resposta entre diagnósticos, comparado a aquisição original (10 kHz durante 10 segundos). Duas tangentes hiperbólicas são avaliadas para implementação em FPGA, apresentando resultados muito próximos a função de ativação original. Uma análise sobre o impacto da redução da precisão numérica indicou também que é possível trabalhar com 4 casas decimais, tanto nos pesos da rede MLP quando das entradas da rede neural, sem perda de acurácia nos classificadores.

6 Implementação dos classificadores neurais em plataformas embarcadas

Neste capítulo é descrito o processo de prototipação de um classificador em duas plataformas diferentes: DSC e FPGA. São descritos os processos e métodos utilizados para embarcar o classificador. Os recursos computacionais de ambos os protótipos são avaliados através de métricas como tempo de computação e consumo de memória, assim como a acurácia obtida nas plataformas em comparação ao classificador no ambiente Matlab. Dentro dos classificadores treinados com a configuração de duas coletas de quatro segundos a 1 kHz, um classificador binário com acurácia de 88,3% e ternário de 61,51% foi escolhido para ser embarcado em ambas as plataformas. O classificador utilizado tem sete neurônios na camada oculta e três na de saída.

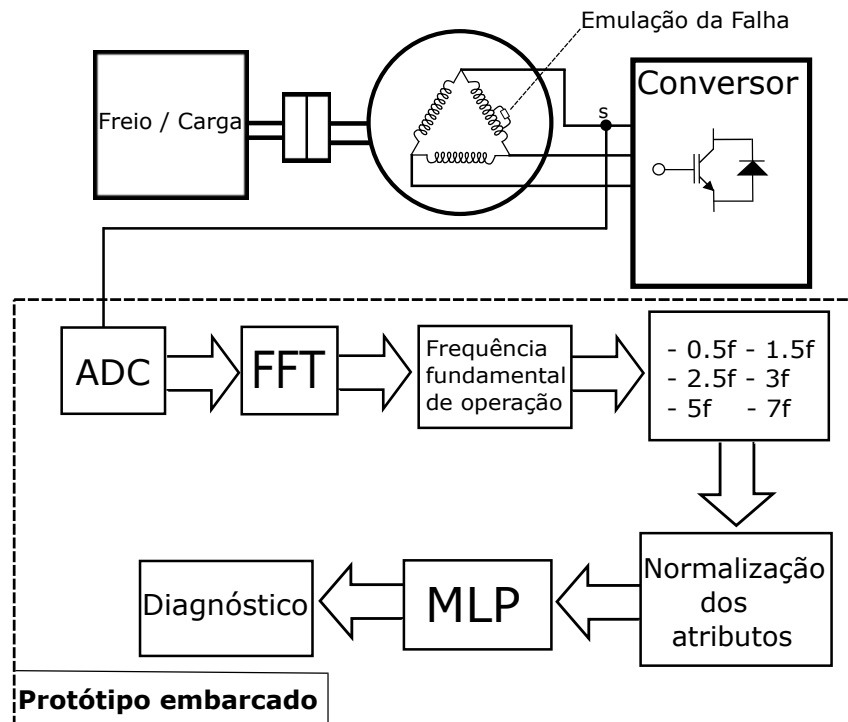
6.1 Etapas executadas pelo protótipo proposto

O protótipo embarcado proposto no trabalho segue a sequência de passos a seguir (todas ilustradas dentro do retângulo de borda quadriculada da Figura 22). Após a obtenção dos sinais através do sensor de corrente (representado pela letra 's' na Figura 22) o qual é convertido através de um ADC, os pontos são submetidos a FFT. Com os valores transformados para o domínio da frequência, a amplitude da fundamental (f) é encontrada, onde a partir dela, são calculados os 6 atributos da rede neural ($0,5f, 1,5f, \dots, 7f$). O próximo passo é a normalização dos atributos por duas técnicas: forma z-score e transformação dos valores para os intervalos de -1 a +1. Após a normalização, a RNA é executada, onde por fim o motor é diagnosticado.

6.2 Processo de prototipação em um DSC

O DSC escolhido para prototipação é o dsPIC33EP512MU810. O dsPIC é um Controlador Digital de Sinais (DSC) da Microchip que consiste em um dispositivo de 16 bits com arquitetura *Harvard* modificada. Essa família combina as vantagens de um microcontrolador (MCU) de 16 bits com um Processador Digital de Sinais (DSP), fornecendo assim uma solução completa para aplicações como controle de motores, conversão de potência, sensores de alta velocidade, processamento de sinais de áudio e voz, entre outros (BRAGA, 2014).

Podendo alcançar 30 MIPS (Milhões de Instrução por Segundo) de processamento, os dsPICs podem ser programados em linguagem C, além de possuírem memória *flash*

Figura 22 – Representação do protótipo embarcado para diagnóstico *on-line*

Fonte: O autor

interna de programa, EEPROM de dados, SRAM de dados, periféricos diversos, além de várias bibliotecas de softwares que permitem ao desenvolvedor criar soluções embutidas com facilidade (BRAGA, 2014). Conforme indicado no *site* da fabricante (MICROCHIP, 2016), os dsPICs oferecem o desempenho de um DSP com a simplicidade de um MCU.

Ainda em Microchip (2016), os DSPs, através de um ADC, em geral mais rápidos dos que os convencionais, convertem os sinais analógicos rapidamente para o formato digital, processando-os e entregando-os seja digitalmente ou analogicamente, podendo ainda serem utilizados para controle de dispositivos externos. Os DSPs já possuem um MCU apropriado para esse tipo de tarefa. No entanto, a maior vantagem do dsPIC está em sua arquitetura modificada e recursos que os tornam mais fáceis de usar do que os DSPs convencionais, principalmente para aqueles já familiarizados com a família PIC.

O DSC escolhido tem 52 kB de memória RAM, executa até oito operações por instrução, conversor analógico digital de 10 e de 12 bits de resolução e que pode realizar conversões em frequências de até 1,1 Msps. A plataforma foi escolhida por possuir os periféricos necessários para aquisição e armazenamento dos sinais, além de possuir características de um microcontrolador e também um processador otimizado para processamento digital de sinais.

Os dados da corrente são adquiridos usando o conversor analógico-digital integrado

no dsPIC, com 12 bits de resolução. Conforme avaliado anteriormente, a taxa de amostragem é de 1 kHz, durante 4,096 segundos, ao invés de 4 segundos, para atender a restrição do conversor analógico-digital embutido no dsPIC. No total, essa configuração de coleta de dados, são aquisitados 4096 pontos para composição de uma amostra. Esse é o limite de pontos que são possíveis armazenar nesta plataforma devido o tamanho da memória disponível. O pré-processamento dos dados brutos é a parte mais complexa do protótipo, em termos computacionais, sendo também a mais difícil de implementar. Para aumentar a eficiência e reduzir o tempo de prototipação, a biblioteca de DSP disponível no compilador do fabricante é utilizado. Tais bibliotecas são escritas em assembly, de forma otimizada a alcançar máximo desempenho do *hardware*.

A etapa de pré-processamento é igual ao apresentado na Figura 22 mas com algumas modificações para atender demandas do dsPIC. Após aquisitados, os dados são escalonados para evitar *overflow* durante o cálculo da FFT. O algoritmo utilizado para implementar a FFT é o "Radix-2 Decimation in Frequency", da biblioteca do dsPIC, que requer um vetor de entrada de tamanho compatível com base binária (por isso os 4,096 segundos e não 4 segundos exatos). A saída da função é apresentada em ordem reversa, sendo necessário a reordenação do vetor. Todo o restante do processo segue conforme discutido anteriormente.

O resultado da classificação é exibido através de 3 leds: o led verde simbolizando condição normal de operação, laranja representando falha de alta impedância e a vermelha indicando que uma falha de baixa impedância foi detectada. Para verificar o tempo de execução de cada etapa, os dados brutos armazenados em um computador de propósito geral são enviados ao DSC através da porta serial, onde toda a etapa de pré-processamento e execução do classificador neural ocorre. O tempo de execução de cada etapa executado no protótipo é visto na Tabela 17.

O processo de aquisição é o que consome mais tempo, seguido pelo pré-processamento dos sinais. A execução da rede neural, como visto, é medida com e sem a função de ativação. A tangente hiperbólica, a função de ativação utilizada no classificador para o processo, consome 69% do tempo de execução da rede. Uma vez que redes neurais podem ser treinadas com diferentes funções de ativação ou com funções modificadas e otimizadas, o uso de tais funções podem reduzir ainda mais o tempo de computação dessa etapa. Contudo, no presente caso analisando o tempo total, esse tipo de abordagem fornece uma vantagem irrisória.

A quantidade de memória utilizada pelo protótipo embarcado é de 40982 bytes de RAM e 31542 bytes de ROM. O detalhamento do consumo de memória por cada etapa é descrita na Tabela 18.

A primeira coluna descreve cada etapa do processo de diagnóstico no protótipo, e a segunda indica a memória consumida, assim como o tipo de memória do DSC. O processo de aquisição indica os pontos da leitura da corrente. Na linha de pré-processamento está

Tabela 17 – Tempo de Execução

Rotina	Subrotina	Tempo de Execução
Aquisição e Armazenamento do Dados	-	4,096 s
Pré-Processamento	Escalonar Vetor, Cálculo e Remoção da Média	71 ms
	FFT, Reordenação em Ordem Normal	12 ms
	Espectro de Potência e Obtenção da Fundamental	3,4 ms
	Amplitude das Harmônicas e Normalização dos Dados	16 ms
RNA	Com tangente hiperbólica na função de ativação	1,24 ms
	Sem tangente hiperbólica na função de ativação	388 us

Fonte: O autor

Tabela 18 – Quantidade de memória necessária por rotina de classificação do protótipo - DSC

Rotina	Memória Consumida
Processo de Aquisição	16384 bytes (RAM)
Pré-Processamento	24600 bytes (RAM)
Código	10374 bytes (ROM)
Pesos do Classificador Neural	438 bytes (ROM)
Valores para normalização dos dados	144 bytes (ROM)
Constantes	33 bytes (ROM)
TOTAL	41040 bytes RAM (78,92%) 19365 bytes ROM (9,00%)

Fonte: O autor

condensado a quantidade de memória necessária para execução das funções conforme descrita na Tabela 17. O código representa o algoritmo da rede MLP escrito em C; os valores para normalização dos dados são os valores das médias, desvio padrão, mínimos e máximos de cada uma das harmônicas para normalização, valores esses obtidos dos dados durante o treinamento do classificador; e a linha de constantes, que representa o consumo das mesmas no código.

O maior responsável pelo consumo da memória de dados também é a etapa de aquisição junto ao pré-processamento, totalizando quase 80% da memória disponível do DSC. Neste cenário, um intervalo de coleta maior invalida a plataforma por falta de memória para armazenamento dos dados. Por outro lado, os pesos do classificador consomem pouco espaço, assim como o código da rede neural. Isso indica que é possível embarcar RNAs maiores, ou até mesmo outros tipos de classificadores.

Para comparar a performance obtida na plataforma embarcada com a do MATLAB, 260 amostras do conjunto original de 882 amostras aquisitadas a 1 kHz durante 4 segundos, são selecionadas aleatoriamente e são submetidas ao mesmo classificador nas duas plataformas. Os pesos do classificador são carregados nos dois *softwares* e as amostras são submetidas as redes. Para enviar as amostras para o dsPIC, é utilizada uma porta serial, onde os resultados são retornados pelo mesmo canal. Os resultados obtidos em ambas plataformas são comparados e exibidos na Tabela 19

Tabela 19 – Comparativo de acurácia: MATLAB *versus* dsPIC

	Acurácia Matlab	Acurácia dsPIC
Ternário	61,53%	60,79%
Binário	83,07%	82,69%

Fonte: O autor

Esta avaliação mostra que uma pequena queda de acurácia acontece no dsPIC em comparação ao resultado originalmente obtido no MATLAB em ambas as formas de classificação. Acredita-se que essa diferença é causada pelas conversões dos valores de *float* para ponto fixo, ao escalar os pontos para evitar o overflow durante a etapa do FFT e da diferença de precisão decimal entre o MATLAB e dsPIC. Para verificar isso, mesmo sendo uma pequena diferença que não invalida implementação, uma breve investigação sobre a razão dessa redução da acurácia é feita. O MATLAB usa os dados no formato *double* com 64 bits de precisão, com 15 casas decimais. Na Tabela 20 é exibido os diferentes tipos de dados utilizados nas etapas da implementação do dsPIC.

Tabela 20 – Precisão dos dados no dsPIC

Etapa do Processo	Tipo de Dado	Precisão Numérica
Aquisição, FFT e Cálculo da Frequência Fundamental	Fractional	Ponto fixo de 16 bit
Amplitude das Harmônicas Cálculos, Normalização e Rede Neural	Float	Ponto flutuante de 32 bit

Fonte: O autor

Nas etapas de aquisição até o cálculo da frequência fundamental, um tipo de ponto fixo de 16 bits chamado de "fractional" é utilizado, uma vez que esse formato é requerido para utilizar as funcionalidades da biblioteca do DSP. Do cálculo das amplitudes das harmônicas até a execução da rede neural, o dsPIC utiliza do tipo *float* de 32 bits, com 6 casas decimais de precisão. Logo, as mudanças de tipo e suas precisões diferentes, junto aos processos de normalização, obtenção de média, desvio padrão, valores mínimos e máximos (utilizados na normalização) e os pesos todos truncados a 6 casas decimais, simbolizando diferentes valores dos utilizados no MATLAB, justificam a diferença dos resultados.

6.3 Processo de prototipação em um FPGA

A implementação do classificador neural no FPGA é composta, basicamente, pelo desenvolvimento dos componentes que compõem a rede MLP em uma linguagem de descrição de hardware, sendo a utilizada neste trabalho a VHDL. Como visto no Capítulo 2, onde é ilustrado a composição de uma rede MLP, para executar um classificador é necessário o desenvolvimento de um multiplicador (para multiplicar as entradas dos neurônios com seus respectivos pesos), um somador (para somar o resultados de cada uma dessas multiplicações) e uma função de ativação (no caso do trabalho, a tangente hiperbólica). A partir do desenvolvimento de cada um desses componentes, é possível o compor a unidade de processamento chamado, neste trabalho, de "neurônio", que representa um perceptron.

Um Virtex-5 XCVLX110-3 é utilizado para prototipação em FGPA. Conforme indicado na especificação do produto (XILINX, 2015), o dispositivo contém 17,280 *slices* (com 4 LUTs e 4 Flip-Flops cada, totalizando 69,120 de cada), 64 multiplicadores e 4,608 blocos de RAM (kB). De forma simplificada, uma LUT é um conjunto de portas lógicas interconectadas. Ao desenvolver o circuito no FPGA em uma linguagem de descrição, as LUTs armazenam uma lista predeterminada de saídas para uma dada combinação de entradas, representando a lógica definida pelo engenheiro. Os *slices* são uma combinação de recursos lógicos (normalmente LUTs, flip-flops e multiplexadores) que compõem um bloco lógico configurável, que são a base dos FPGAs. Cada FPGA implementa LUTs e *slices* de formas diferentes. No caso do Virtex-5, cada *slice* contém 4 LUTs e 4 flip-flops. Mais detalhes podem ser encontrados em (MARWEDEL, 2011, pág. 152).

O componente "neurônio" é a base para construção da rede na plataforma. Cada neurônio é composto por outros componentes: somador, multiplicador e tangente hiperbólica. Todos os componentes foram testados via simulador Active-HDL 8.1 (ALDEC, 2016) antes de serem sintetizados de fato na placa, tendo todos funcionado, em suas versões finais, conforme o esperado.

Como explorado no capítulo anterior, trabalhar com até quatro decimais não reduz

a acurácia do classificador, sendo essa a precisão utilizada na implementação em VHDL do classificador. Com essa precisão, todos os pesos do classificador escolhido para prototipação é convertido para o formato ponto fixo Q15, assim como as amostras do conjunto de teste do classificador através de um *script* em Python. A partir da definição do formato dos números, o primeiro componente a ser desenvolvido é o bloco somador binário de números ponto fixo com sinal, visto a natureza dos pesos e entradas da rede. No segundo momento, é utilizado um multiplicador adquirido em (OPENCORE.ORG, 2009) que efetua somente multiplicações de valores positivos. Com a adição de um novo componente que realiza complemento de dois, um novo multiplicador é criado e validado, sendo capaz de efetuar multiplicações de valores negativos também.

A próxima etapa é a implementação do bloco para cálculo das tangentes hiperbólicas. Neste cenário, as duas soluções apresentadas no Capítulo 5 foram desenvolvidas. Após definir o uso da segunda solução (linearização + redução do erro através de uma RALUT) devido ao seu menor consumo de *hardware* por não necessitar de multiplicadores, a mesma é implementada em VHDL e testada. O resultado de todas as implementações com os devidos mapeamentos de entradas e saídas entre os componentes, geram o 'neurônio', onde na Figura 23 ilustra o *schematic* do componente neurônio da camada oculta. Os componentes com o '+' correspondem aos somadores, os com '*' aos multiplicadores e com 'tanh' a tangente hiperbólica. Os 'inputs' de 1 a 6 correspondem às 6 componentes da entrada da rede; os 'synapses' de 1 a 6 aos pesos do neurônio relacionados a cada entrada de mesmo número. Com o *bias* da rede neural com o valor de '-1', o valor do limiar ($\theta_i^{(h)}$) representando pela 'synapse0' no componente referente ao neurônio já é armazenado no componente em VHDL com o sinal invertido, funcionando como uma entrada.

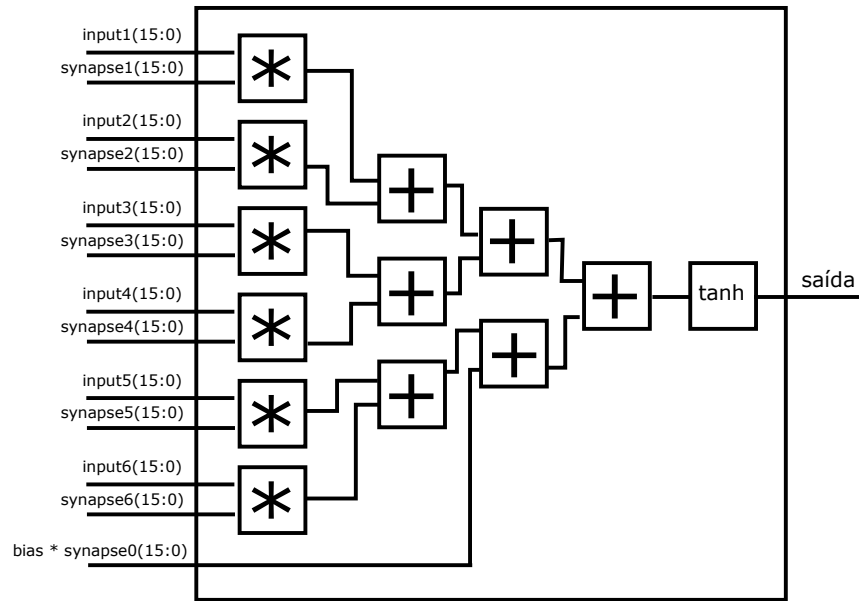
O neurônio da camada de saída é ilustrado na Figura 24. Os 'outputn' de 1 a 7 referem-se as saídas dos sete neurônios da camada de saída. O restante, segue a mesma representação da Figura 23.

A próxima etapa consiste em estabelecer o mapeamento das entradas e saídas de todos os componentes neurônios, ou seja, interligar as entradas da rede com os neurônios da camada oculta e as saídas dos neurônios da camada oculta com as entradas dos neurônios de saída. O resultado das conexões dos componentes são observados na Figura 25, representando toda a RNA implementada em VHDL.

Para a etapa de síntese da RNA, é utilizada a ferramenta ISE Xilinx 12.4 (XILINX, 2012). Os resultados da síntese referentes aos recursos necessários no FPGA por componente (assim como o percentual que tais valores correspondem em relação ao total disponível na plataforma) é visto na Tabela 21.

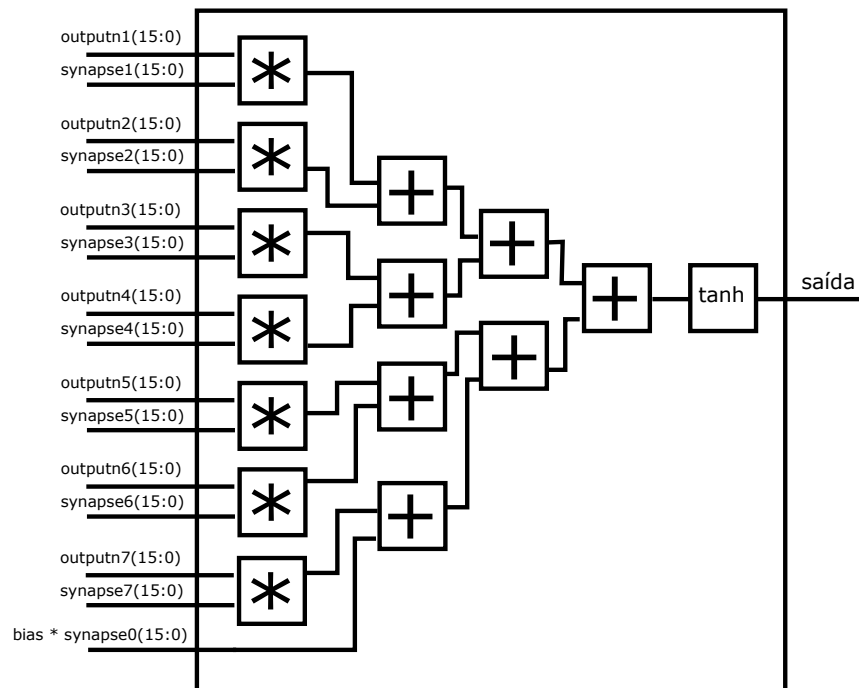
Como validação da implementação, as amostras do conjunto de teste são submetidos ao classificador na placa. Para tal, é implementado o processador Microblaze v8b na placa com o intuito de enviar as amostras e visualizar as saídas dos neurônios via porta serial,

Figura 23 – Esquemático de um neurônio - Camada oculta



Fonte: O autor

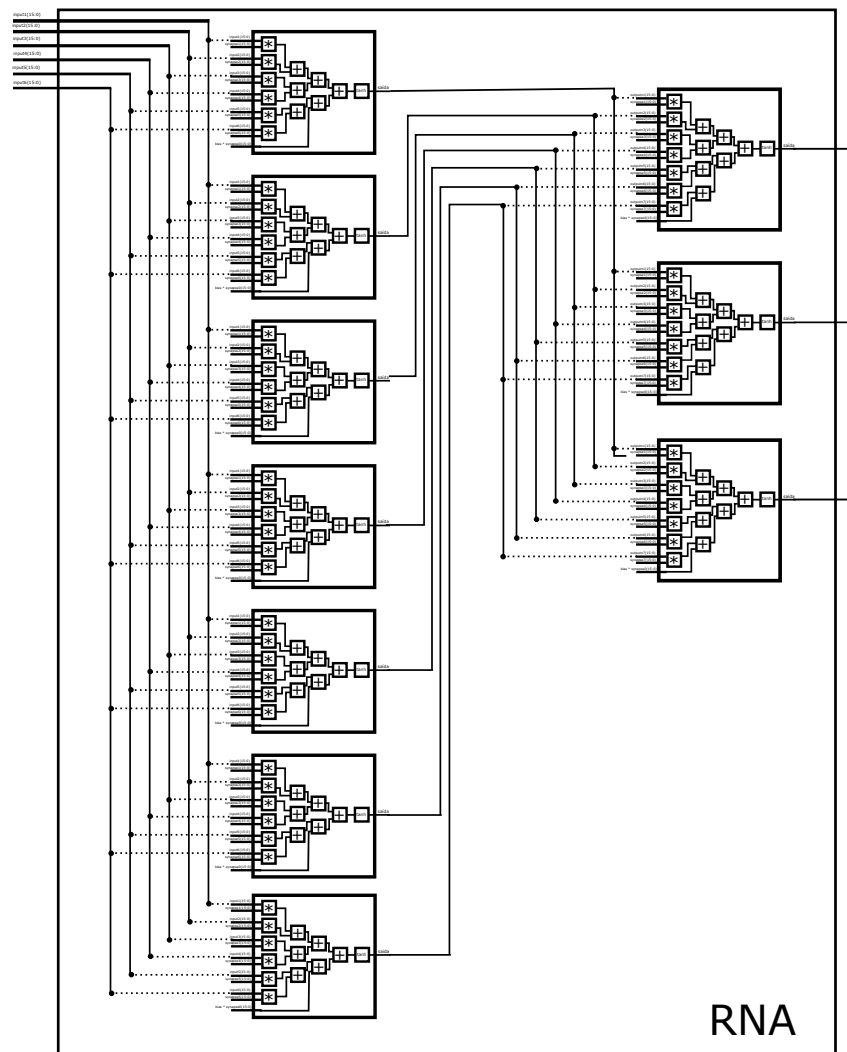
Figura 24 – Esquemático de um neurônio - Camada de saída



Fonte: O autor

através do Docklight 2.0 (HEGGELBACHER, 2016). O Microblaze (XILINX, 2009) é *soft microprocessor* RISC embarcado de 32 bits otimizado para implementação em FPGAs

Figura 25 – Esquemático da rede neural implementada em VHDL com os componentes mapeados



Fonte: O autor

da Xilinx (XILINX, 2009, pág. 14). É um processador *soft core* altamente configurável que permite o projetista escolher o conjunto de funcionalidades que mais se adequam ao projeto. Através dessa configuração, as amostras do conjunto de teste do classificador embarcado foi enviado e executado na placa.

Os resultados comparativos são vistos na Tabela 22, onde o resultado de três testes são mostrados: os resultados obtidos pelo classificador no Matlab com precisão de 15 casas decimais e utilizando a tangente hiperbólica original como função de ativação; os resultados obtidos no Matlab com as configurações de execução do FPGA (4 casas decimais, junto a função de ativação sendo a de aproximação com retas e RALUT); e os resultados obtidos pelo classificador sendo executado no FPGA. Tais teste são importantes para avaliar a diferença dos resultados obtidos via simulação no ambiente Matlab e no FPGA. Ao simular

Tabela 21 – Recursos necessários por componente da RNA no FPGA

Componente	Número de Slices	Slice Registers	Slice LUTs
Somador	30 (1%)	-	54 (1%)
Multiplicador	209 (1%)	5 (1%)	537 (1%)
Tangente Hiperbólica	87 (1%)	-	242 (1%)
Neurônio (Camada Escondida)	1,447 (8%)	30 (1%)	3,636 (5%)
Neurônio (Camada Saída)	1,476 (8%)	35 (1%)	4,202 (6%)
Rede Completa	6,388 (36%)	90 (1%)	16,253 (23%)

Fonte: O autor

as configurações de execução do FPGA no Matlab, é possível, além de validar ambas implementações, verificar onde possíveis diferenças de resultado ocorrem.

Tabela 22 – Comparativo dos resultados obtidos no FPGA e Matlab

	Acurácia (%)		EQM
	Ternário	Binário	
Matlab	61,51%	88,31%	0,3168
Matlab (Configuração do FPGA)	61,86%	89,18%	0,3144
FPGA	62,03%	89,17%	0,3144

Fonte: O autor

Os resultados validam a implementação em FPGA ao apresentar resultados similares aos obtidos via Matlab, seja com o classificador original quanto o utilizando as configurações do FPGA. A pequena diferença dos resultados do Matlab com a configuração do FPGA para os resultados obtidos na placa pode ser justificada pelos diferentes tipos de dados. Os valores no Matlab estão no formato *float*, enquanto que na placa são utilizados valores no formato Q15. Para visualização das diferenças das saídas geradas pelo Matlab (classificador original) e FPGA, as respostas de 6 amostras são exibidas na Tabela 23, onde N1, N2 e N3 significam, respectivamente, a saída dos neurônios 1, 2 e 3 da camada de saída.

Observa-se que há uma redução de precisão das saídas dos valores, mas ainda sim em termos de classificação não há nenhuma discrepância. Como análise mais completa, é calculado o erro quadrático médio do classificador do FPGA, em comparação ao original sendo executado no Matlab. Nesse teste, são submetidas 582 amostras, com os resultados obtidos nos classificadores dos dois ambientes (Matlab e FPGA) são utilizados para cálculo do EQM. Os EQM por amostra são plotados em ordem crescente no gráfico da Figura 26, onde é possível ver a distribuição dos erros, assim como outras informações. Os resultados

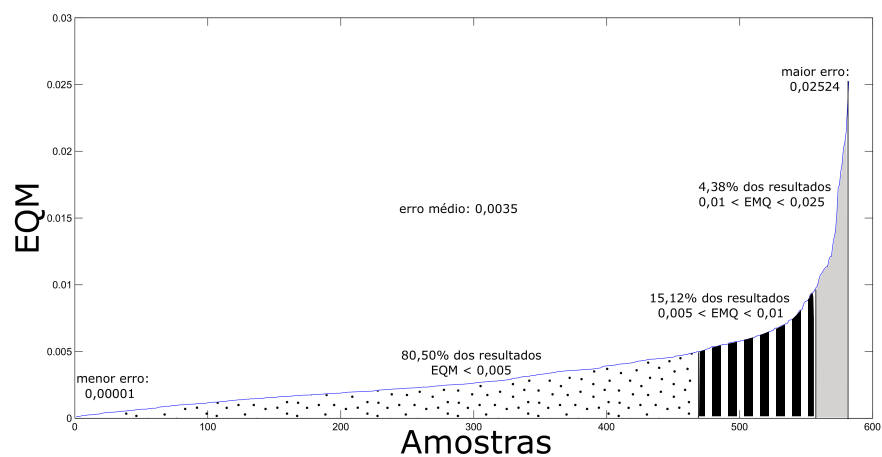
Tabela 23 – Comparativo das saídas em 5 amostras: Matlab × FPGA

Amostra	N1 (Matlab)	N1 (FPGA)	N2 (Matlab)	N2 (FPGA)	N3 (Matlab)	N3 (FPGA)
1 (Sem falha)	0,4326	0,3601	-0,6850	-0,6445	-0,8120	-0,8200
2 (Sem falha)	0,7906	0,7797	-0,8795	-0,8601	-0,9072	-0,9000
3 (Sem falha)	0,1706	0,2729	-0,7788	-0,8200	-0,8686	-0,8609
4 (Falha de AI)	-0,9964	-0,9600	0,5666	0,5642	-0,4440	-0,3874
5 (Falha de BI)	-0,9974	-0,9602	-0,9992	-0,9602	0,9997	0,9599

Fonte: O autor

indicam que o erro é na maioria das vezes pequeno, sugerindo que a implementação é válida.

Figura 26 – EQM dos resultados do FPGA em comparação aos obtidos no Matlab



Fonte: O autor

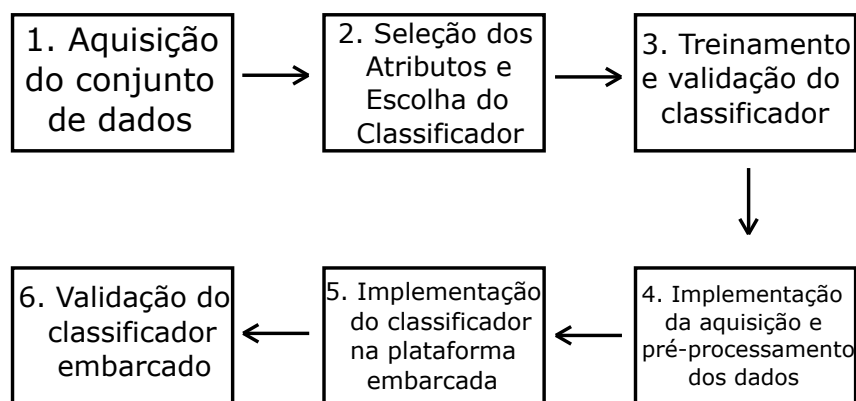
Outra verificação avaliada é a de tempo de execução. Devido a natureza paralela da arquitetura fazendo com que todos os valores das entradas sejam propagados pela rede de forma simultânea, para avaliar o tempo de execução é necessário observar o maior tempo (pior caso) de propagação de todas as entradas (bits) da rede até a saída. Para tal, é utilizada a ferramenta ISE Project Navigator, versão 12.4 (XILINX, 2012), durante a etapa de síntese da RNA na placa. Os resultados indicam que o maior intervalo para resposta é de 76,250 ns.

Conforme mostrado, a parte implementada em FPGA é somente da rede neural. Diversos outros processos ainda devem ser implementados em busca de um produto final, como todo o processo de aquisição e pré-processamento dos dados.

6.4 Métodos e Técnicas para prototipação de classificadores neurais em plataformas embarcadas

Após as descrições dos processos apresentados para embarcar o classificador neural em duas plataformas diferentes, é possível sintetizar os métodos utilizados no trabalho. De forma a sistematizar o processo para projetos que visam embarcar classificadores neurais, a Figura 27 mostra o fluxo geral de desenvolvimento do projeto embarcado, onde cada etapa é discutida a seguir..

Figura 27 – Processo para desenvolvimento de um classificador neural embarcado



Fonte: O autor

1. **Aquisição do conjunto de dados:** a aquisição dos dados deve ser feita de forma a representar as condições operacionais reais do que se busca classificar, se não for possíveis todas, pelo menos grande parte dessas condições. No exemplo deste trabalho, ao coletar as amostras de falha do motor em diferentes condições de operação (frequências, carga, diferentes composições de falha), é possível construir um classificador robusto em termos de generalização do problema. A qualidade dos dados, como resolução em imagens e vídeos ou taxas de amostragem em sinais em geral, por exemplo, devem ser representativas o suficiente para que seja possível caracterizar as classes que se procura identificar, e de forma que tais amostras possam ser reduzidas ou adaptadas, caso necessário, para prototipação embarcada. No caso deste trabalho, ao aquirir sinais em superamostragem, é possível modificá-los e efetuar testes em diferentes condições derivadas da aquisição original, como feito no Capítulo 5. Quanto maior for a precisão ou resolução dos dados, maior a quantidade de recursos computacionais, seja para armazená-los ou aquisitá-los. Em Medeiros e Barreto (2015, págs. 81-88) os autores apresentam diversas instruções de como aquirir e formar um conjunto de dados, alertando a importância e relevância desta etapa. Outro ponto é, descrever e documentar a etapa de aquisição para

replicar o processo em possíveis novas coletas para treinamento ou para validação do classificador *on-line*.

2. **Seleção dos atributos e escolha do classificador:** após aquisitados os dados, a seleção dos atributos deve ser efetuada. Essa pode ser uma etapa mais desafiadora, porém pode ser facilitada através de algumas técnicas. Histograma, projeções bidimensionais (ou gráficos de dispersão), aplicação de técnicas para seleção de atributos, clusterização, Análise de Componentes Principais (PCA), entre outras técnicas podem ser aliadas poderosas nessa etapa. Mais informações e detalhes sobre essas técnicas podem ser encontrados em Medeiros e Barreto (2015, págs. 89-98), Duda, Hart e Stork (2000, pág.11) e Guyon e Elisseeff (2003). Já a escolha do classificador está diretamente relacionada a seleção dos atributos e a forma como os mesmos estão distribuídos no espaço. O ideal, contudo, é buscar classificadores com menor custo computacional ou com uma complexidade reduzida, ainda mais quando se busca trabalhar com plataformas embarcadas. Conforme indicado em Medeiros e Barreto (2015, págs. 10), é uma técnica comum começar com algoritmos mais simples de inteligência computacional ao tentar solucionar um problema e partir para classificadores mais complexos, caso os resultados obtidos não sejam aplicáveis para uma solução satisfatória.
3. **Treinamento e validação do classificador:** o treinamento e validação devem ser feitos normalmente, mas tendo em mente as possíveis restrições da plataforma. É comum encontrar MLPs onde a função de ativação da camada de saída é linear. Logo, treinar classificadores dessa forma pode ser uma técnica interessante para reduzir recursos dos neurônios da camada de saída. Visto também na Tabela 17, a tangente hiperbólica é a responsável por cerca de 69% do tempo de execução da RNA no dsPIC. Existem outras funções de ativação com um menor custo ou até funções otimizadas que podem ser utilizadas já no treinamento, técnica essa que pode reduzir o tempo de execução, assim como a diferença de saída apresentadas entre o ambiente Matlab e as plataformas embarcadas. Já RNAs muito grandes, por exemplo, com várias camadas podem ser um problema em termos de recursos computacionais. Como visto no presente trabalho, uma rede pequena consumiu cerca de 10% da memória do dsPIC e não mais que 35% de espaço do FPGA. O desenvolvimento de modelos neurais pequenos é uma preocupação perceptível no trabalho realizado por Oliveira (2014), no qual este se baseia, uma vez que o mesmo já tinha em mente que tal classificador seria embarcado no futuro. Isso ocasionou a implementação da RNA sem desafios relacionados a necessidade de memória, visto os números supracitados.
4. **Implementação da aquisição e pré-processamento dos dados:** a etapa de desenvolvimento de aquisição e pré-processamento dos dados é responsável por grande

parte do tempo de desenvolvimento da pesquisa. Conforme visto no Capítulo 3, tais etapas são responsáveis por grande parte das dificuldades encontradas no processo de prototipação embarcada das diversas aplicações. Como observado na Seção 6.2, tais etapas são responsáveis pela maior necessidade de recursos computacionais, tanto de memória quanto de tempo de execução. São comuns adaptações dos valores dos pesos e entradas das redes para execução das funções das plataformas embarcadas. No caso do trabalho, o uso do DSC, por exemplo, necessita de adaptações dos valores do sinal para evita o overflow em quase todas as funções nativas da plataforma. Os processos de aquisição em FPGA, por sua vez, conforme sugerido em trabalhos com aplicações em campo visto no Capítulo 3, são comumente efetuadas através de *soft processors*. Logo, para MCUs, DSPs ou DSCs é mais comum a implementação dessas etapas através de funções nativas das plataformas. Já pra FPGAs, a implementação da RNAs em *hardware* e dos processos complementares (aquisição e pré-processamento dos dados) em *soft processors* é mais utilizada.

5. **Implementação do classificador na plataforma embarcada:** a implementação do algoritmo na linguagem da plataforma alvo, como no presente trabalho em C para o dsPIC e em VHDL para o FPGA pode ser feita pela própria equipe ou buscar padrões ou códigos já prontos como os já existentes apresentados anteriormente no estado da arte. É natural que o reuso reduz o tempo de projeto mas pode tornar o projeto menos flexível para atender alguns requisitos do projeto. Já em uma equipes menos experientes, toda a implementação do código pode aumentar muito o tempo de projeto e possíveis erros de implementação. Logo, um equilíbrio pode ser uma solução ideal para equipes pequenas e pouco experientes, como apresentado no trabalho: buscar alguns componentes já prontos e adaptá-los para o problema, no caso dos FPGAs, utilizar-se de funções e bibliotecas das plataformas, no caso do DSC, e buscar códigos e ideias na literatura para acelerar algumas etapas (como na função de ativação utilizado para o FPGA apresentado no trabalho). Após validação do algoritmo, os pesos (no caso das redes neurais, ou qualquer outro coeficiente ou valores que representem o conhecimento adquirido para classificação na fase de treinamento) deve ser carregado no código da plataforma. A forma mais simples é associar tais valores o próprio código. Uma dica é determinar tais variáveis de uma forma clara, para possíveis mudanças no futuro, caso novos classificadores sejam treinados. Na implementação do dsPIC, por exemplo, os pesos são carregados como constantes dentro do código. No FPGA, também são associadas ao código, onde cada peso correspondente ao neurônio é considerada como entrada do componente de mesmo nome.
6. **Validação do classificador embarcado:** a validação pode ocorrer em dois momentos distintos: uma ao verificar os resultados obtidos através de simulações do

ambiente de classificação por intermédio de um computador ou da validação do protótipo *on-line* já na aplicação real. Como apresentado no trabalho, no caso dos protótipos nas duas plataformas utilizadas, a simulação por computador ocorre ao enviar os sinais da corrente do motor via porta serial. Ao receber os resultados obtidos pelos cálculos nos protótipos pelo mesmo canal para o computador, é possível uma análise mais detalhada das saídas obtidas dos classificadores embarcados, como por exemplo o valor do EQM. Essa é uma análise interessante para avaliar a precisão dos resultados, não só de acurácia média mas sim de valores gerados pela saída das RNAs. É possível também utilizar o próprio compilador das plataformas para visualizar os valores obtidos do processamento (como o simulador do FPGA ou debbugador do dsPIC). Ao avaliar o protótipo na aplicação real, a documentação que descreve o ensaio para aquistar os dados originais é justificada para replicação precisa desta etapa. É comum observar em trabalhos similares uma perda de precisão dos resultados e os projetistas devem sempre levar em conta essa perda, por menor que seja, ao projetar os classificadores.

6.5 Conclusão

Este capítulo apresenta o processo de prototipação do classificador neural para detecção de falha em MIT em duas plataformas distintas: em um DSC, um híbrido de microcontrolador com um DSP e em um FPGA. Os métodos e ferramentas utilizadas para prototipação nas plataformas são descritas. Três métricas são avaliadas em ambas as plataformas: o tempo de computação, quantidade de recursos necessários e comparação da acurácia obtida originalmente via simulação em computador. Os resultados indicaram que ambas implementações são válidas, com resultados próximos aos obtidos via simulação em computador.

A aquisição dos dados e extração dos atributos é a principal responsável pelos recursos computacionais exigidos no protótipo do DSC. Por outro lado, o classificador neural exige poucos recursos computacionais, comparado ao total exigido do protótipo, o que valida as escolhas feitas durante o desenvolvimento do modelo neural por Oliveira (2014).

7 Conclusão

Nesta dissertação é descrito o processo para embarcar um classificador neural para detecção de curto-circuito entre espiras do bobinamento de motores de indução trifásico alimentados por conversor de frequência. A partir do conjunto de dados e do modelo neural definido em Oliveira (2014), composto por 441 amostras representando diferentes condições de falha (variando a extensão e intensidade da falha) e de operação do motor (diferentes frequências de operação, níveis de carga e de fase de coleta dos dados), uma nova forma de classificar os dados é avaliada, diferente da classificação binária que já vinha sendo trabalhada em trabalhos anteriores. A escolha da rede MLP se deu por sua simplicidade computacional em comparação a outros algoritmos de inteligência computacional, exigindo assim menos recursos, assim como trabalhos anteriores que justificam sua utilização na solução do problema.

A partir dos novos modelos, são exploradas diferentes configurações de aquisição de dados para redução de recursos computacionais do processo, visando a prototipação embarcada. Inicialmente, o conjunto original de 10 kHz a 10 segundos é reduzido em diversos outros conjuntos de menor taxa (5 kHz, 2,5 kHz, 2 kHz, 1,25 kHz e 1 kHz). Os novos dados são submetidos aos classificadores já treinados com o conjunto original, assim como são utilizados para o treinamento de novos classificadores. Os testes indicaram que a taxa de amostragem de 1 kHz é o suficiente para o desenvolvimento do protótipo.

De forma similar, o número de pontos é reduzido ao simular menores períodos de aquisição. Novos conjuntos são criados simulando menores intervalos de aquisição, onde os testes dos conjuntos ocorrem para verificar qual é o menor tempo de aquisição suficientemente representativo para obtenção dos valores necessários para treinamento da rede. Os resultados mostram que é possível uma redução no período de aquisição de 10 segundos para 4 segundos. No total, há uma redução de 96% dos pontos no processo de aquisição e 60% do tempo de diagnóstico em relação ao conjunto original, o que reduz significativamente o custo de embarcar o classificador ao se levar em conta a quantidade de memória necessária para armazenar os valores.

Um estudo sobre o impacto das precisões numéricas dos valores dos pesos e das redes também é desenvolvido. Após o treinamento dos classificadores, é verificado a diferença resultante da redução da precisão dos pesos sobre a acurácia original. Essa verificação também ocorre nos valores das componentes de frequência (as entradas da rede). Conclui-se que com até 4 casas decimais não há redução significativa no resultado final dos modelos neurais comparado aos resultados originais.

Duas propostas de implementação da tangente hiperbólica também são discutidas

e avaliadas para implementação em VHDL. A primeira é baseada pela aproximação da tangente hiperbólica com 10 retas e a segunda em uma proposta que utiliza a aproximação da função com 2 retas junto a uma memória (RALUT) que auxilia na redução do erro cometido pela aproximação de tais retas. Em testes no MATLAB, as duas se mostraram válidas como alternativa de função de ativação com uma pequena redução na acurácia que não afeta significativamente o resultado final. A segunda proposta é escolhida por não necessitar, ao contrário da primeira solução, de multiplicação, visto que esse cálculo é um componente custoso em termos computacionais em implementações em *hardware*.

Após definir o período e taxa de amostragem de aquisição dos dados, assim como um classificador para tais configurações, códigos em C e VHDL do algoritmo são implementados e embarcados, respectivamente, em um DSC e em um FPGA. Os processos são descritos e métricas como consumo de memória e tempo de execução são discutidos. Conclui-se pela implementação principalmente no dsPIC que o gargalo do projeto está na aquisição dos sinais. Essa etapa é responsável por consumir grande parte da memória disponível no DSC, assim como ser responsável pelo tempo de diagnóstico de 4 segundos. Já ao avaliar os recursos necessários para execução da rede neural, a tangente hiperbólica é responsável por 69% do tempo de execução. No FPGA, somente a rede neural foi implementada. Em termos comparativos, o dsPIC executa a rede em 1,24 ms com valores das operações do tipo *float* e o FPGA em 76,250 ns, com os valores das operações no formato Q15.

Por fim, baseada na experiência obtida e pesquisas similares, os métodos e processos utilizados para embarcar o classificador neural são condensados. Um fluxograma para implementação de classificadores direcionados a plataformas embarcadas é exposto descrevendo as etapas seguidas empiricamente no trabalho, assim como os principais desafios encontrados no decorrer do projeto.

Há nesse cenário um avanço para solução do problema para detectar falhas de motor de indução trifásico. A partir de trabalhos anteriores, o resultado final deste trabalho é um passo significativo no desenvolvimento de um protótipo funcional. Ao considerar o processo para embarcar algoritmos computacionais, observa-se que há de certa forma uma negligência dos projetistas a respeito dos processos de aquisição e pré-processamento dos dados. São poucos os trabalhos encontrados na literatura que descrevem tais etapas e o impacto das mesmas nos projetos. No presente trabalho, por exemplo, essas foram as etapas que mais demandam recursos, sejam computacionais e/ou de pesquisa.

7.1 Trabalhos futuros

Uma vez definido os algoritmos nas plataformas alvo e uma configuração de aquisição dados que elas possam trabalhar, novos modelos neurais devem ser encontrados para classificar melhor os dados adaptados a tais mudanças. Tais modelos podem melhorar

significativamente as taxas apresentadas no trabalho.

Novas amostras podem ser coletadas também para compor um conjunto de dados maior. Visto a quantidade de subclasses e a pouca representatividade dos dados normais, novas coletas podem auxiliar significativamente na composição de novos classificadores.

No FPGA, somente a rede neural foi implementada. Logo, há espaço para explorar a implementação das etapas de aquisição e pré-processamento na plataforma.

Visto que o modelo neural consumiu menos recursos do que esperado, é possível que novos algoritmos possam ser embarcados e comparados de forma a enriquecer o produto, podendo até mesmo possibilitar a criação de um comitê de classificadores.

Por fim, uma nova estrutura de bancada está sendo confeccionada para aquisição de dados, uma vez que o processo de aquisição original foi modernizado. Isso sugere que os novos dados a serem aquisitados são mais confiáveis para diagnóstico do motor, em comparação aos utilizados nesta dissertação. Quando pronto, além dos novos dados já citados que podem ser adquiridos, o sistema pode ser testado em tempo real com emulações das falhas para verificação do desempenho do protótipo.

Referências

- AKLAH, Z.; ANDREWS, D. *A Flexible Multilayer Perceptron Co-processor for FPGAs*. Bochum, Germany: Springer International Publishing, 2015. 427-434 p. ISBN 978-3-319-16214-0. Disponível em: <http://dx.doi.org/10.1007/978-3-319-16214-0_39>. Acesso em: 05 de de Junho de 2015. Citado 2 vezes nas páginas 35 e 62.
- ALBRECHT, P. F. et al. Assessment of the reliability of motors in utility applications - updated. *IEEE Transactions on Energy Conversion*, EC-1, n. 1, p. 39–46, March 1986. ISSN 0885-8969. Disponível em: <<http://dx.doi.org/10.1109/TEC.1986.4765668>>. Acesso em: 22 de Abril de 2016. Citado 2 vezes nas páginas 19 e 23.
- ALDEC. *Active-HDL 8.1 User Guide*. Henderson, Nevada, EUA, 2016. Disponível em: <<http://users.ugent.be/~wmeeus/software/Active-HDL8.1SP1-SoftwareManual.pdf>>. Citado na página 76.
- BASTERRETXEA, K.; ECHANOBÉ, J.; CAMPO, I. d. A wearable human activity recognition system on a chip. In: *Design and Architectures for Signal and Image Processing (DASIP), 2014 Conference on*. Madrid, Spain: IEEE, 2014. p. 1–8. Citado na página 39.
- BATISTA, G. E. d. A. P. A. *Pré-Processamento de Dados em Aprendizado de Máquina Supervisionado*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, São Paulo, 2003. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/55/55134/tde-06102003-160219/publico/TeseDoutorado.pdf>>. Acesso em: 16 de Junho de 2016. Citado na página 28.
- BERMAK, A.; MARINEZ, D. A compact 3-d vlsi classifier using bagging threshold network ensembles,. *IEEE Transactions on Neural Networks*, v. 14, n. 5, p. 1097–1109, 2003. Disponível em: <<http://ieeexplore.ieee.org/document/1243713/>>. Citado na página 34.
- BOSQUE, G.; CAMPO, I. d.; ECHANOBÉ, J. Fuzzy systems, neural networks and neuro-fuzzy systems: A vision on their hardware implementation and platforms over two decades. *Engineering Applications of Artificial Intelligence*, v. 32, p. 283 – 331, 2014. ISSN 0952-1976. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0952197614000384>>. Citado na página 34.
- BOUZID, M. B. K.; CHAMPENOIS, G. Neural network based method for the automatic detection of the stator faults of the induction motor. In: *Electrical Engineering and Software Applications (ICEESA), 2013 International Conference on*. Hammamet, Tunisia: IEEE, 2013. p. 1–7. Citado na página 36.
- BRAGA, A. L. S. et al. Performance, accuracy, power consumption and resource utilization analysis for hardware / software realized artificial neural networks. In: *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*. Changsha, China: IEEE, 2010. p. 1629–1636. Citado na página 62.
- BRAGA, N. C. *Conheça o dsPIC*. 2014. Disponível em: <<http://www.newtoncbraga.com.br/index.php/microcontroladores/141-microchip-pic/3472-mic029>>. Acesso em: 27 de Junho de 2016. Citado 2 vezes nas páginas 71 e 72.

- BROWN, B. E.; YU, X.; GARVERICK, S. L. Mixed-mode analog vlsi continuous-time recurrent neural network. In: RASHID, M. H. (Ed.). *Circuits, Signals, and Systems*. IASTED/ACTA Press, 2004. p. 398–403. Disponível em: <<http://dblp.uni-trier.de/db/conf/iastedCCS/iastedCCS2004.html#BrownYG04>>. Citado na página 34.
- COELHO, D.; MEDEIROS, C. M. S. Short circuit incipient fault detection and supervision in a three-phase induction motor with a som-based algorithm. In: *Advances in Self-Organizing Maps*. Santiago, Chile: Springer, 2013. p. 315–323. Citado 2 vezes nas páginas 37 e 47.
- COELHO, D. N. et al. Performance comparison of classifiers in the detection of short circuit incipient fault in a three-phase induction motor. In: *Computational Intelligence for Engineering Solutions (CIES), 2014 IEEE Symposium on*. [S.l.: s.n.], 2014. p. 42–48. Citado 5 vezes nas páginas 19, 37, 38, 47 e 106.
- CONTIN, M. C. *Motores Alimentados por Inversores de Frequência: O Isolamento Resiste*. s.d. Accessed: 2016-12-19. Disponível em: <http://www.centralmat.com.br/artigos/mais/isolamento_inversores.pdf>. Citado na página 24.
- CUNHA, R. G. C. Trabalho do Conclusão de Curso, *Aquisição e Tratamento de Dados para Aplicação de RNA em Microcontroladores: Estudo de Caso para Detecção de Falhas em Motor de Indução*. Fortaleza, Ceará: [s.n.], 2016. Citado na página 43.
- CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, v. 2, n. 4, p. 303–314, 1989. ISSN 1435-568X. Disponível em: <<http://dx.doi.org/10.1007/BF02551274>>. Citado na página 26.
- D'ANGELO, M. F. et al. Incipient fault detection in induction machine stator-winding using a fuzzy-bayesian change point detection approach. *Applied Soft Computing*, v. 11, n. 1, p. 179 – 192, 2011. ISSN 1568-4946. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S156849460900221X>>. Citado na página 24.
- DEOTALE, P. D.; DOLE, L. Design of fpga based general purpose neural network. In: *Information Communication and Embedded Systems (ICICES), 2014 International Conference on*. Chennai, India: IEEE, 2014. p. 1–5. Citado na página 62.
- DOUGLASS, B. P. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Primeira edição. Boston, USA: Addiston Wesley, 2008. v. 1. Citado na página 18.
- DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern Classification (2Nd Edition)*. New York, USA: Wiley-Interscience, 2000. ISBN 0471056693. Citado na página 83.
- EICKHOFF, R.; KAULMANN, T.; RÜCKERT, U. Sirens: A simple reconfigurable neural hardware structure for artificial neural network implementations. In: *Proceedings of the 19th International Joint-Conference on Neural Networks (IJCNN)*. Vancouver, Canadá: IEEE, 2006. Citado na página 35.
- ENGELBRECHT, A. P. *Computational Intelligence: An Introduction*. 2nd. ed. Hoboken, New Jersey, USA: Wiley Publishing, 2007. ISBN 0470035617. Citado 4 vezes nas páginas 25, 27, 28 e 96.

ESMAEILZADEH, H. et al. Nnep, design pattern for neural-network-based embedded systems. In: *2007 14th International Conference on Mixed Design of Integrated Circuits and Systems*. Ciechocinek, Poland: IEEE, 2007. p. 673–678. Citado na página 35.

FUNAHASHI, K. On the approximate realization of continuous mappings by neural networks. *Neural Netw.*, Elsevier Science Ltd., Oxford, UK, UK, v. 2, n. 3, p. 183–192, maio 1989. ISSN 0893-6080. Disponível em: <[http://dx.doi.org/10.1016/0893-6080\(89\)90003-8](http://dx.doi.org/10.1016/0893-6080(89)90003-8)>. Citado na página 26.

GHATE, V. N.; DUDUL, S. V. Optimal mlp neural network classifier for fault detection of three phase induction motor. *Expert Syst. Appl.*, Pergamon Press, Inc., Tarrytown, NY, USA, v. 37, n. 4, p. 3468–3481, abr. 2010. ISSN 0957-4174. Disponível em: <<http://dx.doi.org/10.1016/j.eswa.2009.10.041>>. Citado na página 23.

GRACIOLA, C. L. et al. Neural speed estimator for line-connected induction motor embedded in a digital processor. *Applied Soft Computing*, v. 40, p. 616 – 623, 2016. ISSN 1568-4946. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1568494615008121>>. Citado na página 39.

GUYON, I.; ELISSEEFF, A. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, JMLR.org, v. 3, p. 1157–1182, mar 2003. ISSN 1532-4435. Disponível em: <<http://dl.acm.org/citation.cfm?id=944919.944968>>. Citado na página 83.

HAYKIN, S. *Redes Neurais: Princípios e Práticas*. Segunda edição. Porto Alegre: Bookman, 2007. ISBN 978-85-7307-718-6. Citado 6 vezes nas páginas 18, 24, 29, 96, 100 e 101.

HEGGELBACHER, F. . *Docklight V2.2 User Manual 07-2016*. [S.l.], 2016. Disponível em: <www.xilinx.com/support/documentation/sw_manuals/xilinx11/mb_ref_guide.pdf>. Citado na página 77.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Netw.*, Elsevier Science Ltd., Oxford, UK, UK, v. 2, n. 5, p. 359–366, jul. 1989. ISSN 0893-6080. Disponível em: <[http://dx.doi.org/10.1016/0893-6080\(89\)90020-8](http://dx.doi.org/10.1016/0893-6080(89)90020-8)>. Citado na página 26.

HUSIN, Z. et al. Embedded portable device for herb leaves recognition using image processing techniques and neural network algorithm. *Computers and Electronics in Agriculture*, v. 89, p. 18 – 29, 2012. ISSN 0168-1699. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0168169912001949>>. Citado na página 40.

IEEE, T. o. I. A. Report of large motor reliability survey of industrial and commercial installations, part i. *IEEE Transactions on Industry Applications*, IA-21, n. 4, p. 853–864, July 1985. ISSN 0093-9994. Citado na página 23.

KAISTI, M. et al. Agile methods for embedded systems development - a literature review and a mapping study. *EURASIP Journal on Embedded Systems*, v. 2013, n. 1, p. 1–16, 2013. ISSN 1687-3963. Disponível em: <<http://dx.doi.org/10.1186/1687-3963-2013-15>>. Citado 2 vezes nas páginas 18 e 62.

- KAUFHOLD, M. et al. Interface phenomena in stator winding insulation - challenges in design, diagnosis, and service experience. *IEEE Electrical Insulation Magazine*, v. 18, n. 2, p. 27–36, March 2002. ISSN 0883-7554. Citado 2 vezes nas páginas 19 e 24.
- KOHONEN, T.; SCHROEDER, M. R.; HUANG, T. S. (Ed.). *Self-Organizing Maps*. 3rd. ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001. ISBN 3540679219. Citado na página 37.
- KOZIEL, S. et al. A neural network embedded system for real-time estimation of muscle forces. *Procedia Computer Science*, v. 51, p. 60 – 69, 2015. ISSN 1877-0509. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1877050915010042>>. Citado 2 vezes nas páginas 34 e 40.
- LASHKARI, N.; POSHTAN, J. Detection and discrimination of stator interturn fault and unbalanced supply voltage fault in induction motor using neural network. In: *Power Electronics, Drives Systems Technologies Conference (PEDSTC), 2015 6th*. Tehran, Iran: IEEE, 2015. p. 275–280. Citado 2 vezes nas páginas 23 e 37.
- LEHMANN, T.; BRUUN, E.; DIETRICH, C. Mixed analog/digital matrix-vector multiplier for neural network synapses. *Analog Integrated Circuits and Signal Processing*, v. 9, n. 1, p. 55–63, 1996. ISSN 1573-1979. Disponível em: <<http://dx.doi.org/10.1007/BF00158852>>. Citado na página 34.
- MAIOROV, V.; PINKUS, A. Lower bounds for approximation by mlp neural networks. *Neurocomputing*, v. 25, n. 1–3, p. 81 – 91, 1999. ISSN 0925-2312. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0925231298001118>>. Citado na página 26.
- MARWEDEL, P. *Embedded System Design*. Segunda edição. Dortmund, Germany: Springer, 2011. v. 1. Citado 4 vezes nas páginas 17, 30, 31 e 76.
- MEDEIROS, C. M. S.; BARRETO, G. A. *Notas de Aula: Inteligência Computacional Aplicada*. Fortaleza, 2015. Citado 5 vezes nas páginas 27, 29, 82, 83 e 100.
- MICROCHIP, S. O. d. *dsPIC33F/dsPIC33E: High Performance DSCs*. 2016. Disponível em: <<http://www.microchip.com/design-centers/16-bit/products/dspic33f-e>>. Acesso em: 27 de Junho de 2016. Citado na página 72.
- MISRA, J.; SAHA, I. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, v. 74, n. 1–3, p. 239 – 255, 2010. ISSN 0925-2312. Artificial Brains. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S092523121000216X>>. Citado na página 34.
- MOERLAND, P. D.; FIESLER, E.; SAXENA, I. Incorporation of liquid-crystal light valve nonlinearities in optical multilayer neural networks. *Appl. Opt.*, OSA, v. 35, n. 26, p. 5301–5307, Sep 1996. Disponível em: <<http://ao.osa.org/abstract.cfm?URI=ao-35-26-5301>>. Citado na página 34.
- MORÉ, J. J. *The Levenberg-Marquardt algorithm: Implementation and theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978. 105–116 p. ISBN 978-3-540-35972-2. Disponível em: <<http://dx.doi.org/10.1007/BFb0067700>>. Citado na página 29.

- MORSALIN, S. et al. Induction motor inter-turn fault detection using heuristic noninvasive approach by artificial neural network with levenberg marquardt algorithm. In: *Informatics, Electronics Vision (ICIEV), 2014 International Conference on*. Dhaka, Bangladesh: IEEE, 2014. p. 1–6. Citado 2 vezes nas páginas 24 e 36.
- NAMIN, A. H. et al. Efficient hardware implementation of the hyperbolic tangent sigmoid function. In: *2009 IEEE International Symposium on Circuits and Systems*. Taipei, Taiwan: IEEE, 2009. p. 2117–2120. ISSN 0271-4302. Citado 4 vezes nas páginas 62, 63, 65 e 66.
- NANDI, S.; TOLIYAT, H. A. Condition monitoring and fault diagnosis of electrical machines-a review. In: *Industry Applications Conference, 1999. Thirty-Fourth IAS Annual Meeting. Conference Record of the 1999 IEEE*. Phoenix, AZ, USA: IEEE, 1999. v. 1, p. 197–204 vol.1. ISSN 0197-2618. Citado 2 vezes nas páginas 23 e 24.
- NATARAJA, R. Failure identification of induction motors by sensing unbalanced stator currents. *IEEE Transactions on Energy Conversion*, v. 4, n. 4, p. 585–590, Dec 1989. ISSN 0885-8969. Citado na página 24.
- NEDJAH, N.; MOURELLE, L. de M. Reconfigurable hardware for neural networks: binary versus stochastic. *Neural Computing and Applications*, v. 16, n. 3, p. 249–255, 2007. ISSN 1433-3058. Disponível em: <<http://dx.doi.org/10.1007/s00521-007-0086-x>>. Citado na página 34.
- NEDJAH, N.; SILVA, R. M. d.; MOURELLE, L. d. M. Compact yet efficient hardware implementation of artificial neural networks with customized topology. *Expert Systems with Applications*, v. 39, n. 10, p. 9191 – 9206, 2012. ISSN 0957-4174. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0957417412003296>>. Citado na página 62.
- NYQUIST, H. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, v. 47, n. 2, p. 617–644, April 1928. ISSN 0096-3860. Citado na página 53.
- OLIVEIRA, A.; PONTES, R.; MEDEIROS, C. M. S. Neural network used to stator winding interturn short-circuit fault detection in an induction motor driven by frequency converter. In: *Computational Intelligence and 11th Brazilian Congress on Computational Intelligence (BRICS-CCI CBIC), 2013 BRICS Congress on*. Recife, Brasil: IEEE, 2013. p. 459–464. Citado 2 vezes nas páginas 37 e 47.
- OLIVEIRA, A. S. d.; ANDRADE, F. S. d. *Sistemas Embarcados: Hardware e Firmware na Prática*. Primeira edição. São Paulo: Erica, 2006. v. 1. Citado 2 vezes nas páginas 17 e 32.
- OLIVEIRA, t. G. *Classificadores Neurais Aplicados na Detecção de Curto-Circuito Entre Espiras Estatísticas em Motores de Indução Trifásicos Acionados por Conversores de Frequência*. Dissertação (Mestrado) — Universidade Federal do Ceará, Fortaleza, Maio 2014. Citado 12 vezes nas páginas 19, 42, 45, 48, 49, 50, 51, 53, 83, 85, 86 e 106.
- OPENCORE.ORG. *OpenCore.org*. 2009. Accessed: 2015-09-16. Disponível em: <http://opencores.org/project,hierarch_unit>. Citado na página 77.

PALÁCIOS, R. H. C. et al. Reconhecedor neural de defeitos no estator em motores de indução trifásicos apoiado por análise de componentes principais. In: *Anais do XII Simpósio Brasileiro de Automação Inteligente, SBAI 2015*. Natal, Rio Grande do Norte: Online, 2015. p. 1206–1211. Disponível em: <<http://www.sbai2015.dca.ufrn.br/download/artigo/314>>. Citado na página 36.

PANDEY, K. . K.; ZOPE, P. H.; R., S. S. Article: Review on fault diagnosis in three-phase induction motor. *IJCA Proceedings on National Conference "MEDHA 2012"*, MEDHA, n. 1, p. 53–58, September 2012. Full text available. Citado na página 24.

PAPA, J. P.; FALCÃO, A. X. F. Optimum-path forest: A novel and powerful framework for supervised graph-based pattern recognition techniques. In: *Concurso de Teses e Dissertações (CTD)*. Bento Gonçalves, RS, Brasil: Online, 2009. v. 1, p. 41–48. Citado na página 18.

PENMAN, J. et al. Detection and location of interturn short circuits in the stator windings of operating motors. *IEEE Transactions on Energy Conversion*, v. 9, n. 4, p. 652–658, Dec 1994. ISSN 0885-8969. Citado na página 45.

SALDANHA, L. B.; BOBDA, C. An embedded system for handwritten digit recognition. *J. Syst. Archit.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 61, n. 10, p. 693–699, nov. 2015. ISSN 1383-7621. Disponível em: <<http://dx.doi.org/10.1016/j.sysarc.2015.07.015>>. Citado na página 40.

SHARMA, P. *FPGA implementation of artificial neural networks*. Dissertação (Mestrado) — National Institute of Technology, Rourkela, Odisha, India, mar. 2007. Citado na página 32.

SILVA, E. T. d. *Middleware Adaptativo para Sistemas Embarcados e de Tempo-real*. Tese (Doutorado) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Rio Grande do Sul, 2008. Disponível em: <<https://www.lume.ufrgs.br/bitstream/handle/10183/13657/000652234.pdf?sequence=1>>. Acesso em: 5 de Maio de 2016. Citado na página 30.

SMITH, S. W. *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, CA, USA: California Technical Publishing, 1997. ISBN 0-9660176-3-3. Citado na página 33.

SOBRINHO, C. A. N. et al. Desenvolvimento de sistema embarcado para detectar falhas em motores de indução. In: *Anais do XII Simpósio Brasileiro de Automação Inteligente, SBAI 2015*. Online, 2015. Disponível em: <<http://www.sbai2015.dca.ufrn.br/download/artigo/64>>. Citado na página 37.

SOUZA, A. H. de et al. Minimal learning machine: A new distance-based method for supervised learning. In: ROJAS, I.; JOYA, G.; GABESTANY, J. (Ed.). *Advances in Computational Intelligence*. Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 7902). p. 408–416. ISBN 978-3-642-38678-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-38679-4_40>. Citado na página 38.

SUYKENS, J.; VANDEWALLE, J. Least squares support vector machine classifiers. *Neural Processing Letters*, v. 9, n. 3, p. 293–300, 1999. ISSN 1573-773X. Disponível em: <<http://dx.doi.org/10.1023/A:1018628609742>>. Citado na página 18.

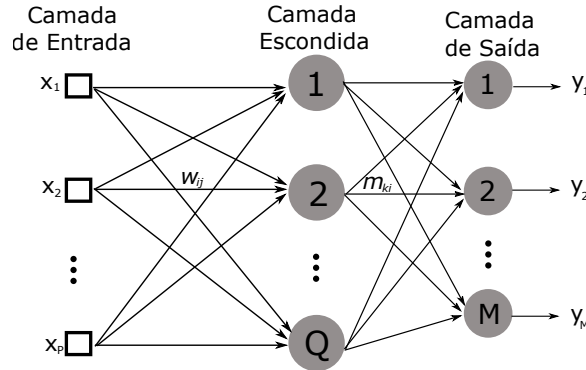
- TALLAM, R. M. et al. A survey of methods for detection of stator-related faults in induction machines. *IEEE Transactions on Industry Applications*, v. 43, n. 4, p. 920–933, July 2007. ISSN 0093-9994. Citado na página 24.
- THOMSON, W. T.; FENGER, M. Current signature analysis to detect induction motor faults. *IEEE Industry Applications Magazine*, v. 7, n. 4, p. 26–34, Jul 2001. ISSN 1077-2618. Citado na página 23.
- VIEIRA, R. G.; MEDEIROS, C. M. S.; SILVA, E. T. Classification and sensitivity analysis to detect fault in induction motors using an mlp network. In: *Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN)*. Vancouver, Canadá: IEEE, 2016. p. 796–802. ISBN 978-1-5090-0619-9. Citado 4 vezes nas páginas 47, 48, 51 e 105.
- WOLF, W. *Computers as Components: Principles of Embedded Computing System Design*. Segunda edição. Burlington, Massachusetts: Elsevier, 2008. v. 1. Citado 3 vezes nas páginas 17, 30 e 33.
- XILINX. *MicroBlaze Processor Reference Guide*. San Jose, California, USA, 2009. Disponível em: <www.xilinx.com/support/documentation/sw_manuals/xilinx11/mb_ref_guide.pdf>. Citado 2 vezes nas páginas 77 e 78.
- XILINX. *ISE In-Depth Tutorial*. San Jose, California, USA, 2012. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ise_tutorial_ug695.pdf>. Citado 2 vezes nas páginas 77 e 81.
- XILINX. *Virtex-5 Family Overview*. San Jose, California, USA, 2015. Disponível em: <www.xilinx.com/support/documentation/data_sheets/ds100.pdf>. Citado na página 76.
- ZAREI, J.; TAJEDDINI, M. A.; KARIMI, H. R. Vibration analysis for bearing fault detection and classification using an intelligent filter. *Mechatronics*, v. 24, n. 2, p. 151 – 157, 2014. ISSN 0957-4158. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S095741581400004X>>. Citado na página 23.

APÊNDICE A – O algoritmo de treinamento

Backpropagation

Um dos mais conhecidos algoritmos de treinamento de MLP é o *Backpropagation* (BP), traduzido comumente como algoritmo de retropropagação (HAYKIN, 2007, pág. 188). Esse algoritmo é baseado no Gradiente Descendente, um método numérico de primeira ordem comumente utilizado em otimização (ENGELBRECHT, 2007, pág. 38). Uma das principais vantagens do BP é sua facilidade de implementação (HAYKIN, 2007, pág. 252). Contudo, apresenta um tempo de computação superior a algoritmos de treinamento como ELM e LVM. A Figura 28 exibe, para auxiliar no entendimento dos conceitos a seguir, a estrutura básica de uma MLP com uma única camada oculta.

Figura 28 – Representação genérica de uma MLP com uma camada escondida



Fonte: Autor desta dissertação

Nesse algoritmo, a etapa de treinamento é dividida em dois momentos: uma fase direta e outra reversa. No sentido direto, há o cálculo das ativações e saídas de todos os neurônios, tanto da camada escondida quanto de saída. Para tal, as unidades de entrada recebem um vetor de entrada \mathbf{x} , na interação t , onde o primeiro passo é o cálculo das ativações dos neurônios da camada escondida, como segue

$$u_i^{(h)} = \sum_{j=1}^P w_{ij}(t)x_{j(t)} - \theta_i^{(h)} = \sum_{j=0}^P w_{ij}(t)x_{j(t)}, \quad i = 1, \dots, Q, \quad (\text{A.1})$$

onde

- $w_{ij}(t)$ é uma conexão sináptica, também conhecida como peso entre a j -ésima entrada e o i -ésimo neurônio da camada escondida;
- $\theta_i^{(h)}$ é o limiar do i -ésimo neurônio da camada escondida;

- $Q (2 \leq Q \leq \infty)$ é o número de neurônios da camada escondida;
- P é a dimensão do vetor de entrada, sem o limiar;

A definição de $x_0(t) = -1$ (valor do bias) e $w_{i0}(t) = \theta_i^{(h)}(t)$ simplifica a notação, conforme visto em A.1.

Em seguida, a saída de cada neurônio oculto é calculada como listado em A.2

$$y_i^{(h)}(t) = \varphi_i [u_i^{(h)}(t)] = \varphi_i \left[\sum_{j=0}^P w_{ij}(t) x_{j(t)} \right], \quad (\text{A.2})$$

onde $\varphi_i(\cdot)$ assume comumente uma das seguintes formas

$$\varphi[u_i^{(h)}(t)] = \frac{1}{1 + \exp[-u_i^{(h)}(t)]} \quad (\text{Logística}) \quad (\text{A.3})$$

e

$$\varphi[u_i^{(h)}(t)] = \frac{1 - \exp[-u_i^{(h)}(t)]}{1 + \exp[-u_i^{(h)}(t)]}. \quad (\text{Tangente Hiperbólica}). \quad (\text{A.4})$$

Após os cálculos das ativações de todos os neurônios das camadas ocultas, o passo seguinte é calcular as ativações dos neurônios da camada de saída,

$$y_k^{(o)}(t) = \varphi_k [u_k^{(o)}(t)] = \varphi_k \left[\sum_{i=0}^Q m_{ki}(t) y_i^{(h)}(t) \right], \quad (\text{A.5})$$

onde:

- $m_{ki}(t)$ é o peso da conexão sináptica entre o i -ésimo neurônio da camada escondida e o k -ésimo neurônio ($k = 1, \dots, M$) da camada de saída;
- $M \geq 1$ é o número de neurônios da camada de saída.

A notação é simplificada novamente ao definir $y_0^{(h)}(t) = -1$ e $m_{k0}(t) = \theta_k^{(o)}(t)$, onde $\theta_k^{(o)}(t)$ é o limiar do neurônio da saída k .

Todas as notações estão em função de t , definindo assim momentos diferentes, pois todos os pesos na fase de treinamento são ajustados a cada apresentação de um novo vetor de entrada, também conhecidas como as amostras do conjunto de treinamento (ou estimação).

No sentido reverso, há uma retropropagação dos sinais de erro de saída através das camadas de saída e escondidas, até atingir a entrada da rede. Para tal, é necessário calcular o valor de erro $e_k^{(o)}(t)$ gerado por cada neurônio de saída no momento t :

$$e_k^{(o)}(t) = d_k(t) - y_k^{(o)}(t), \quad k = 1, \dots, M \quad (\text{A.6})$$

em que $d_k(t)$ é o valor desejado para a saída do k -ésimo neurônio da camada de saída.

Após o cálculo do erro, os pesos devem ser adaptados com intuito de minimizar uma função custo. Essa função pode ser tanto o erro quadrático médio calculado a partir

dos M neurônios de saída em relação a apresentação da amostra $\mathbf{x}(t)$, como visto na equação A.7, quanto pela soma dos erros quadráticos médios (EQM) estimados nos M neurônios de saída

$$\varepsilon(t) = \frac{1}{2M} \sum_{k=1}^M \left[e_k^{(o)}(t) \right]^2 = \frac{1}{2M} \sum_{k=1}^M \left[d_k(t) - y_k^{(o)}(t) \right]^2. \quad (\text{A.7})$$

A função custo $J(t) = \varepsilon(t)$ que deve-se minimizar é dependente dos pesos da rede MLP. Logo, a adaptação de cada peso é feita através da derivada da função custo em relação ao peso a ser atualizado. Pela regra da cadeia, a derivada da função custo em relação ao peso m_{ki} é definida por

$$\frac{\partial J}{\partial m_{ki}(t)} = \frac{d\varepsilon(t)}{de_k^{(o)}(t)} \frac{de_k^{(o)}(t)}{dy_k^{(o)}(t)} \frac{dy_k^{(o)}(t)}{du_k^{(o)}(t)} \frac{du_k^{(o)}(t)}{dm_{ki}(t)}, \quad (\text{A.8})$$

onde

$$\frac{\partial J}{\partial m_{ki}(t)} = -\delta_k^{(o)}(t) y_i^{(h)}(t), \quad (\text{A.9})$$

$\delta_k^{(o)}(t)$ é o gradiente local do neurônio k da camada de saída dado por

$$\delta_k^{(o)}(t) = \varphi'_k \left[u_k^{(o)}(t) \right] e_k^{(o)}(t). \quad (\text{A.10})$$

A derivada $\varphi'_k \left[u_k^{(o)}(t) \right]$ assume diferentes expressões de acordo com a função de ativação. Conforme já listado acima, tem-se como as possibilidades mais comuns a função logística e a tangente hiperbólica:

$$\varphi'_k \left[u_k^{(o)}(t) \right] = y_k^{(o)}(t) [1 - y_k^{(o)}(t)] \quad (\text{para a função Logística}) \quad (\text{A.11})$$

$$\varphi'_k \left[u_k^{(o)}(t) \right] = \frac{1}{2} [1 - (y_k^{(o)}(t))^2] \quad (\text{para a função Tangente Hiperbólica}). \quad (\text{A.12})$$

O próximo passo é calcular os gradientes locais dos neurônios da camada escondida. De forma similar, a derivada da função custo em relação ao peso w_{ij} é calculada como visto da equação A.13

$$\frac{\partial J(t)}{\partial w_{ij}(t)} = \delta_i^{(h)}(t) x_j(t), \quad (\text{A.13})$$

onde $\delta_i^{(h)}(t)$ representa o gradiente local do neurônio i da camada oculta dado pela equação A.14

$$\delta_i^{(h)}(t) = \varphi'_i \left[u_i^{(h)}(t) \right] e_i^{(h)}(t) = \varphi'_i \left[u_i^{(h)}(t) \right] \sum_{k=1}^M m_{ki}(t) \delta_k^{(o)}(t), \quad i = 0, \dots, Q. \quad (\text{A.14})$$

O termo $e_i^{(h)}(t)$ é como o sinal de erro retropropagado ou projetado para o i -ésimo neurônio da camada escondida, desde que tais "sinais de erro" são combinações lineares dos reais sinais de erro cometidos nos neurônios da camada de saída. Isso se deve ao fato de a rede só conhecer as saídas corretas da camada de saída, que são os rótulos das amostras, no caso da MLP como classificador.

O terceiro e último passo da fase reversa corresponde ao processo de atualização dos parâmetros (nesse caso os pesos sinápticos e limiares) da rede MLP. Assim, para a camada de saída há a regra de atualização dos pesos m_{ki} , dada por

$$m_{ki}(t+1) = m_{ki}(t) + \eta \delta_k^{(o)}(t) y_i^{(h)}(t), i = 0, \dots, Q. \quad (\text{A.15})$$

O símbolo η é a taxa de aprendizado, onde o seu valor está no intervalo $0 < \eta < 1$. Já na camada escondida, a regra de atualização dos pesos w_{ij} ocorre por:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) = w_{ij}(t) + \eta \delta_i^{(h)}(t) x_j(t). \quad (\text{A.16})$$

O conceito de época nos treinamentos das redes neurais é definido quando todos os N padrões do conjunto de treinamento é apresentado a rede. A cada padrão (ou amostra) do conjunto apresentado a rede, os pesos são atualizados conforme a sequência apresentada nas equações acima. Muitas épocas podem ser necessárias até que haja convergência na aplicação do algoritmo de retropropagação. Para avaliar tal convergência da rede, é utilizado o erro quadrático médio que pode ser calculado conforme equação A.17

$$\varepsilon_{train} = \frac{1}{2NM} \sum_{t=1}^N \sum_{k=1}^M [e_k^{(o)}(t)]^2 = \frac{1}{2NM} \sum_{t=1}^N \sum_{k=1}^M [d_k(t) - y_k^{(o)}(t)]^2. \quad (\text{A.17})$$

Esse valor é calculado ao fim de cada época usando as amostras do conjunto de treinamento. A cada novo cálculo do erro quadrático médio, se a variação do erro (ou mesmo se o valor do erro) estiver abaixo de um valor predeterminado pelo projetista, é considerado que houve uma convergência satisfatória.

Após isso, os pesos obtidos são armazenados e utilizados para classificar outros padrões. Para verificar se o treinamento foi efetuado de forma a generalizar o problema que se busca resolver, a rede é submetida a novos padrões que não foram apresentados ao classificador na fase de treinamento. O conjunto dessas amostras é chamado de conjunto de teste. A ideia é que se o classificador for capaz de classificar padrões desconhecidos, logo é possível concluir que ele "aprendeu" o problema, validando sua funcionalidade. Para tal, a

saída do k -ésimo neurônio da camada de saída da rede treinada é dada por

$$y_k^{(o)} = \varphi_k \left[\sum_{i=0}^Q m_{ki} \varphi_i \left(\sum_{j=0}^P w_{ij} x_j(t) \right) \right]. \quad (\text{A.18})$$

A.1 Termo de momento

O termo de momento (ξ) é uma melhoria que se pode fazer nas expressões de ajuste dos pesos sinápticos, apresentados nas equações A.15 e A.16. Conforme descrito em Medeiros e Barreto (2015, pág. 61), o objetivo é tornar o processo de modificação mais estável. Usar termo de momento também torna o processo de treinamento menos oscilatório e menos sensível em relação a escolha da taxa de aprendizagem. As modificações das equações A.15 e A.16 podem ser vistas, respectivamente, a seguir:

$$m_{ki}(t+1) = m_{ki}(t) + \eta \delta_k(t) y_i(t) + \xi \Delta m_{ki}(t-1). \quad (\text{A.19})$$

e

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_i(t) x_j(t) + \xi \Delta w_{ij}(t-1) \quad (\text{A.20})$$

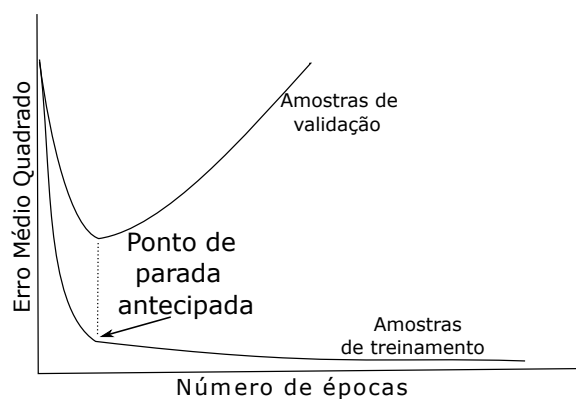
A.2 Validação Cruzada

A validação cruzada é uma ferramenta estatística comumente utilizada em treinamentos de algoritmos de inteligência computacional, em especial nas MLPs. O seu uso é justificado para evitar um problema recorrente ao tentar estimar modelos neurais conhecido como *overfitting* (HAYKIN, 2007, pág. 239). O *overfitting* é quando uma rede "memoriza" os dados de treinamento, perdendo assim sua capacidade de generalização. Uma rede neural artificial ao ser treinada com um número excessivo de épocas, pode acabar encontrando alguma característica presente nos dados de treinamento (como algum tipo de ruído), mas que não está presente na função subjacente que deve ser modelada. Em termos mais simples, a rede "decorou" os dados de treinamento, e não "aprendeu" o problema. Isso acarreta muitas vezes em um desempenho bom ao tentar classificar amostras do treinamento mas desempenhos fracos em padrões nunca antes apresentados a rede.

Para evitar isso, o uso da validação cruzada sugere, como já discutido anteriormente, em dividir o conjunto de dados disponíveis em um conjunto de treinamento e outro conjunto de teste. O conjunto de treinamento, por sua vez, é adicionalmente dividido em dois subconjuntos disjuntos: um subconjunto de estimação, usado para selecionar o modelo; e um subconjunto de validação. O subconjunto de estimação é utilizado de forma a atualizar os parâmetros da rede, com o intuito de reduzir o erro cometido pela rede, sendo

as amostras desse conjunto submetidas durante o treinamento da rede. Já o subconjunto de validação, é utilizado para definir o melhor momento para parar o treinamento. Isso é feito através da observação dos erros quadráticos médios dos subconjuntos de treinamento ao fim de cada época. Nessa abordagem, a partir do momento em que o erro do subconjunto de estimação se estabiliza e o de validação cresce, o treinamento é interrompido. A interpretação desse fenômeno é que a partir desse ponto, a rede parou de generalizar a função que busca modelar e está sobre-ajustando os dados do treinamento. Essa técnica é atrativa para tentar projetar uma rede neural com boa generalização. O processo é ilustrado na Figura 29.

Figura 29 – Ilustração da regra de parada antecipada baseada na validação cruzada



Fonte: Adaptado de (HAYKIN, 2007)

APÊNDICE B – Treinamento de novos modelos neurais para o novo conjunto de dados

Neste apêndice é exibido os resultados obtidos com os novos modelos neurais para os classificadores binários e ternários. As atividades apresentadas nesse apêndice dividem-se nas seguintes etapas:

- Alteração do número de neurônios da camada oculta da rede MLP de 5 a 15 neurônios;
- Criação de novos dados da classe normal gerados através de adição de ruído;
- Escolha dos melhores classificadores dentro dos novos modelos treinados.
- Avaliação do uso de taxa de rejeição nos classificadores escolhidos;

B.1 Alteração do Número de Neurônios na camada oculta

Todas as realizações aqui apresentadas utilizam os mesmos parâmetros de rede dos treinamentos descritos no Capítulo 4 (Tabela 2), alterando somente o número de neurônios da camada oculta (5 a 15) e o número de realizações, que foi reduzido de 50 para 30. Os resultados obtidos para os classificadores binários e ternários com diferentes números de neurônios na camada oculta são observados, respectivamente, nas tabelas 24 e 25. A coluna "Acm" representa a acurácia média para a classificação geral no conjunto de testes; "Amn" representa a acurácia somente para classificação dos dados normais; e 'EQM' exibe o erro quadrático médio. Essa notação é mantida em todas as tabelas doravante.

As números das tabelas indicam que mesmo com a adição de mais neurônios na camada oculta não há melhoria de forma significativa dos resultados. Há uma melhora na performance dos classificadores binários, mas pouco significativa. A melhora é ainda menor ao avaliar os classificadores ternários.

É possível também avaliar de outra forma os resultados dos classificadores ternários, conforme já mostrado no Capítulo 5. Nesta abordagem, os erros de classificação cometidos entre as classes de falhas são desconsiderados, o que transforma o classificador ternário em um binário. Os resultados desta abordagem, utilizando os mesmos classificadores da tabela 25, podem ser observados na tabela 26.

Comparando as Tabelas 24 e 26, é possível observar que a forma de interpretar os resultados "ternário como binário" é mais eficiente que os classificadores binários originais,

Tabela 24 – Resultados médios obtidos de 30 realizações para o classificador binário

Número de Neurônios	Acm	Amn	EQM
5	65,64 \pm 7,19	81,15 \pm 10,31	0,3746
6	66,47 \pm 6,32	81,15 \pm 9,65	0,3853
7	71,99 \pm 6,43	73,97 \pm 12,80	0,4432
8	68,16 \pm 8,27	79,49 \pm 12,05	0,3101
9	68,28 \pm 7,59	80,38 \pm 10,06	0,4377
10	66,98 \pm 8,39	77,05 \pm 12,05	0,4498
11	68,60 \pm 7,56	77,69 \pm 14,65	0,1840
12	70,78 \pm 8,19	76,79 \pm 13,49	0,4461
13	64,16 \pm 8,63	81,79 \pm 12,73	0,5319
14	68,70 \pm 7,64	77,56 \pm 12,20	0,3397
15	69,69 \pm 8,35	77,95 \pm 15,58	0,4307

Fonte: O autor

Tabela 25 – Resultados médios obtidos de 30 realizações para o classificador ternário

Número de Neurônios	Acm	Amn	EQM
5	53,02 \pm 4,91	70,13 \pm 13,38	0,3480
6	51,51 \pm 5,04	70,26 \pm 13,51	0,4310
7	52,20 \pm 3,73	68,46 \pm 12,38	0,4625
8	52,63 \pm 5,50	70,64 \pm 12,22	0,4333
9	52,18 \pm 4,39	73,38 \pm 14,63	0,4189
10	49,44 \pm 5,87	75,90 \pm 12,03	0,3664
11	49,72 \pm 6,75	75,13 \pm 14,86	0,3481
12	51,23 \pm 4,79	73,59 \pm 12,96	0,3824
13	53,48 \pm 3,90	64,62 \pm 17,56	0,4982
14	51,59 \pm 4,80	73,21 \pm 13,45	0,3130
15	52,29 \pm 5,72	71,92 \pm 14,18	0,4347

Fonte: O autor

isso quando a acurácia geral é avaliada. Vale observar, contudo, que há uma redução na classificação das amostras normal.

Analisando a Tabela 26, que contém os melhores resultados comparados aos exibidos nas Tabelas 24 e 25, o modelo escolhido para avaliação é o com 14 neurônios na camada oculta. Com resultados médios aproximados dos maiores valores obtidos com outros modelos tanto na classificação dos dados normais quanto na classificação geral, o modelo apresenta também o menor EQM.

Dentro dos classificadores do modelo, destacam-se três:

- Classificador da realização 1: Acurácia total de 71,99% para todos os dados e 88,46% para os padrões conjunto normal;

Tabela 26 – Resultados médios obtidos de 30 realizações para o classificador ternário com interpretação dos resultados como binário

Número de Neurônios	Acm	Amn	EQM
5	75,90 \pm 7,73	70,13 \pm 13,38	0,3480
6	73,61 \pm 7,22	70,26 \pm 13,51	0,4310
7	75,97 \pm 6,23	68,46 \pm 12,38	0,4625
8	76,40 \pm 9,06	70,64 \pm 12,22	0,4333
9	75,56 \pm 7,63	73,38 \pm 14,63	0,4189
10	69,79 \pm 8,74	75,90 \pm 12,03	0,3664
11	71,09 \pm 9,46	75,13 \pm 14,86	0,3481
12	73,72 \pm 7,55	73,59 \pm 12,96	0,3824
13	78,44 \pm 7,33	64,62 \pm 17,56	0,4982
14	73,87 \pm 7,24	73,21 \pm 13,45	0,3130
15	76,28 \pm 8,83	71,92 \pm 14,18	0,3853

Fonte: O autor

- Classificador da realização 4: Acurácia total de 60,31% para todos os dados e 100,00% para os padrões conjunto normal;
- Classificador da realização 25: Acurácia total de 80,77% para todos os dados e 82,13% para os padrões conjunto normal;

Outra alternativa para melhorar a confiabilidade dos diagnósticos, é a aplicação de uma faixa de rejeição. Essa técnica consiste em ignorar classificações onde o resultado do neurônio da camada de saída referente a classe que tal neurônio representa, estiver abaixo de um valor pré-determinado.

Como exemplo, consideramos o modelo apresentado neste trabalho, contém três neurônios na camada de saída. Logo, para cada classificação, o resultado da execução da rede neural são três valores, cada um no intervalo de -1 a +1, uma vez que os neurônios utilizam como função de ativação a tangente hiperbólica. Caso o valor de saída do primeiro neurônio for o maior, a amostra é classificada como normal; caso o valor de saída do segundo neurônio for o maior, a amostra é classificada com falha de alta impedância. Caso o maior valor seja o do terceiro neurônio, o padrão é classificado como falha de baixa impedância. Contudo, mesmo que o valor de saída de um neurônio seja maior, comparado aos valores dos outros neurônios, tal valor pode estar longe do rótulo atribuído originalmente (no caso, +1). Uma forma de interpretar tais resultados é que, quando mais longe a saída do neurônio estiver do seu rótulo original, menos confiável é a classificação. Exemplificando, a saída esperada para um padrão normal é, respectivamente para os três neurônios da camada oculta, "+1 -1 -1"; Se a saída obtida for "+0,2 -1 -1", o padrão ainda será classificado como normal. Mas é possível entender que tal classificação não é tão confiável comparado a saídas mais próximas de "+1". Neste exemplo, caso a taxa de

rejeição seja de "0,3", qualquer resultado cuja saída for menor que a taxa de rejeição, não será classificado, sendo "rejeitado" pelo classificador (como no caso do exemplo de saída "+0,2 -1 -1").

Os três classificadores listados anteriormente são submetidos a diferentes taxas de rejeição. O impacto na acurácia (total e amostras normais) e o número de padrões que são rejeitados (coluna Padrões Rejeitados) são exibidas na tabela 27.

Tabela 27 – Resultados dos classificadores com diferentes taxas de rejeição

Taxa de Rejeição	Classificador	Acurácia total	Acurácia N	Padrões Rejeitados
Sem taxa	Classificador 1	71,99%	88,46%	0%
	Classificador 4	60,31%	100%	0%
	Classificador 25	82,13%	80,77%	0%
0	Classificador 1	74,43%	90,91%	16%
	Classificador 4	66,01%	100%	30%
	Classificador 25	89,67%	93,75%	26%
0,1	Classificador 1	76,13%	94,74%	23%
	Classificador 4	73,54%	100%	44%
	Classificador 25	90,25%	93,7%	31%
0,2	Classificador 1	78,02%	94,47%	30%
	Classificador 4	78,68%	100%	53%
	Classificador 25	90,75%	93,33%	33%
0,3	Classificador 1	79,25%	100%	36%
	Classificador 4	83,84%	100%	60%
	Classificador 25	91,80%	92,86%	37%
0,4	Classificador 1	80,5%	100%	45%
	Classificador 4	88,14%	100%	69,5%
	Classificador 25	92,44%	100%	40,9%

Fonte: O autor

Analisando os resultados, é proposto o classificador 1 com a taxa de rejeição 0 para ser testado em campo. A taxa de rejeição é relativamente baixa (16%), com uma boa acurácia para os dados normais (90,91%) e com uma acurácia total de 74,43%. Adicionalmente, os resultados apresentados em Vieira, Medeiros e Silva (2016), indicam que a maior parte das falhas que não são detectadas (que são confundidas com o motor operando sem falha) são: as que apresentam falhas no menor número de espiras (curto-circuito em 5 espiras) e nas falhas onde foi emulado um curto de alta impedância (curto de menor intensidade). Já que se trata de uma falha que evolui (ou seja, o curto evolui de alta impedância para baixa impedância e o número de espiras sob curto tende aumentar), tais conclusões sugerem que o classificador seja capaz de detectar as falhas mais severas antes delas atingirem um estado degradativo do bobinamento. Contudo, naturalmente,

tais afirmações só poderão ser confirmadas com o teste do protótipo em campo, conectado a um motor real.

B.1.1 Criação de novas amostras para o conjunto normal

Conforme discutido em Coelho et al. (2014) e Oliveira (2014), o menor número de amostras do conjunto normal comparado ao conjunto de falha, dificulta o treinamento, assim como uma avaliação mais precisa dos resultados dos classificadores. Em ambos os trabalhos, a solução proposta pelos autores foi criar novas amostras para o conjunto normal adicionando ruído.

Para avaliar a técnica de criação de novas amostras artificiais com o novo conjunto de dados, o método utilizado foi a mesma do autor Oliveira (2014, pág. 77), onde o ruído foi gerado a partir de uma função randômica uniforme (rand), que retorna valores entre dois limites pré-estabelecidos. O nível de ruído aplicado são valores randômicos do intervalo de 0 a 0,00001 sobre os valores individuais de cada atributo de cada padrão. As dados artificiais criados são utilizados somente para treinamento dos classificadores. Os dados originais adquiridos da bancada de testes são alocados (ainda de forma aleatória) nos conjuntos de validação e teste para cada realização. A Tabela 28 exibe a quantidade de dados utilizados em cada conjunto para as realizações dos novos classificadores binários e ternários.

Tabela 28 – Quantidade de amostras nos conjuntos de Treinamento, Validação e Teste para treinamento das RNAS nos classificadores binários e ternários

Projeto	Conjunto de Treinamento	Conjunto de Validação	Conjunto de Teste
1 - N, F	529 N (ruído) 529 F	75 N (ruído) 75 F	152 N (126 originais, 26 ruído) 152 Falha
2 - N, AI, BI	264 N (252 ruído, 12 originais) 264 AI 264 BI	37 N (originais) 37 AI 37 BI	77 N (originais) 77 AI 77 BI

Fonte: O Autor

Com essa configuração de separação dos dados, novos treinamentos foram realizados com diferentes números na camada oculta. Os resultados para o classificador binário, ternário e ternário com saída binária são observados, respectivamente nas tabelas 29, 30 e 31

Os resultados indicaram que o melhor modelo dos treinamentos com adição de ruído é o com 15 neurônios para classificador binário. Dentro do modelo, destaca-se um classificador:

Tabela 29 – Resultados médios obtidos de 30 realizações para o classificador binário com adição de novos dados de conjunto normal

Número de Neurônios	Acm	Amn	EQM
5	80,91 \pm 2,11	88,18 \pm 4,82	0,2434
6	82,07 \pm 1,59	88,29 \pm 5,92	0,2319
7	84,22 \pm 2,38	89,30 \pm 4,04	0,2222
8	85,29 \pm 3,07	91,25 \pm 4,59	0,2116
9	86,68 \pm 3,15	93,33 \pm 3,86	0,2368
10	86,23 \pm 2,89	92,58 \pm 4,28	0,2450
11	86,97 \pm 2,53	93,99 \pm 3,46	0,2092
12	87,94 \pm 3,08	93,86 \pm 3,60	0,2147
13	87,52 \pm 2,38	94,89 \pm 3,22	0,2294
14	87,32 \pm 2,64	93,49 \pm 4,34	0,1690
15	88,40 \pm 1,95	96,03 \pm 2,77	0,1776

Fonte: O autor

Tabela 30 – Resultados médios obtidos de 30 realizações para o classificador ternário com adição de novos dados de conjunto normal

Número de Neurônios	Acm	Amn	EQM
5	62,34 \pm 2,29	77,06 \pm 8,19	0,3193
6	61,88 \pm 3,23	75,50 \pm 9,19	0,3019
7	62,41 \pm 2,34	77,40 \pm 8,25	0,3108
8	61,21 \pm 2,87	75,97 \pm 9,02	0,3044
9	63,03 \pm 3,34	76,84 \pm 9,92	0,3105
10	62,02 \pm 3,40	79,96 \pm 7,92	0,3032
11	64,34 \pm 2,87	78,53 \pm 8,31	0,3227
12	63,39 \pm 2,92	78,79 \pm 7,84	0,2942
13	62,87 \pm 2,79	77,62 \pm 7,56	0,2964
14	64,24 \pm 3,48	80,43 \pm 10,34	0,3134
15	61,90 \pm 3,35	78,14 \pm 11,13	0,3191

Fonte: O autor

- Classificador da realização 16: Acurácia total de 91,77% para todos os dados e 98,68% para os padrões conjunto normal com EQM de 0,1619;

A aplicação de taxa de rejeição de até 0,5 neste classificador não ocasionou nenhuma rejeição de padrões, indicando que o uso de taxa de rejeição não é necessário para este classificador. Logo, com tais resultados apresentados, o classificador é sugerido para ser embarcado para teste em campo e comparado com o classificador proposto treinado sem dados artificiais. Tal comparação pode permitir afirmações mais precisas sobre a capacidade de generalização de ambos os classificadores, verificando, também, se o processo de criação de novos padrões com ruído é válido ao testar a classificação com amostras reais do motor

Tabela 31 – Resultados médios obtidos de 30 realizações para o classificador ternário com interpretação dos resultados como binário

Número de Neurônios	Acm	Amn	EQM
5	$77,82 \pm 2,52$	$77,06 \pm 8,19$	0,3193
6	$77,76 \pm 4,02$	$75,50 \pm 9,19$	0,3019
7	$78,46 \pm 3,13$	$77,40 \pm 8,25$	0,3108
8	$77,69 \pm 3,58$	$75,97 \pm 9,02$	0,3044
9	$79,19 \pm 3,46$	$76,84 \pm 9,92$	0,3105
10	$78,76 \pm 2,89$	$79,96 \pm 7,92$	0,3032
11	$78,23 \pm 3,60$	$78,53 \pm 8,31$	0,3227
12	$80,14 \pm 3,33$	$78,79 \pm 7,84$	0,2942
13	$79,39 \pm 2,88$	$77,62 \pm 7,56$	0,2964
14	$81,57 \pm 3,67$	$80,43 \pm 10,34$	0,3134
15	$78,70 \pm 4,45$	$78,14 \pm 11,13$	0,3191

Fonte: O autor

operando sem falha.

APÊNDICE C – Código da rede neural na linguagem C para execução no DSC

Nesse Apêndice é exibido o código da rede neural utilizado no DSC. Há dois arquivos principais: um arquivo "RNA.h" que contém algumas declarações de funções e constantes e o arquivo "RNA.c" que contém a rede neural.

```

1 #file 'RNA.h'
2 #ifndef RNA_H
3 #define RNA_H
4
5 #ifdef __cplusplus
6 extern "C" {
7 #endif
8
9 #define NO_OF_LAYERS 2 //num de camadas;
10 #define NO_OF_OUTPUTS 3 //num de saidas;
11 #define NO_OF_INPUTS_HIDDEN_LAYER 7 //na primeira camada, sem bias;
12 #define NO_OF_INPUTS_OUTPUT_LAYER 8 //na primeira camada, sem bias;
13 #define BIAS -1
14
15 //Funcoes da rede
16 double neuronInput(__psv__ double *weights,int noOfWeights,int
    index,double *input); // funcao que calcula o a ativacao (somatorio) do
    neuronio;
17 double neuronOutput(char axonFamily,double activation); //funcao que
    calcula a saida (funcao de ativacao) do neuronio;
18 unsigned char mlp(__psv__ double *w,__psv__ double *m,double *input);
    //funcao principal que executa o algoritmo;
19 void imprime(double num);
20
21 /*Declaration of Weights
22 w = pesos da entrada para camada escondida.
23 m = pesos da camada escondida para de saida;
24
25 Os pesos devem ser colocado no formato de vetor; os limiares (pesos
    referentes ao BIAS) de cada neuronio ja devem ser inclusos, sendo o
    primeiro de cada "linha";
26
27 Como exemplo:
28
29 => Sendo uma matriz de pesos 5x7:
30     - As linhas representam os neuronios (5 neuronios);

```



```

31      - As colunas representam os pesos (1 limiar no primeiro indice da
          coluna + 6 pesos nos outros indices = 7);
32      => A matriz deve ser adicionada em w como um vetor de 35 (5x7) onde:
33      - 0 a 6, representam os pesos do primeiro neuronio, onde 0 eh o
          limiar do neuronio e 1 a 6 sao os pesos referentes as entradas
34      - 7 a 13, representam os pesos do segundo neuronio, onde 7 eh o
          limiar do neuronio e 8 a 13 sao os pesos referentes as entradas
35      - 14 a 20, representam os pesos do terceiro neuronio, onde 14 eh o
          limiar do neuronio e 15 a 20 sao os pesos referentes as entradas
36      - 21 a 27, representam os pesos do quarto neuronio, onde 21 eh o
          limiar do neuronio e 22 a 27 sao os pesos referentes as entradas
37      - 28 a 34, representam os pesos do quinto neuronio, onde 28 eh o
          limiar do neuronio e 29 a 34 sao os pesos referentes as entradas
38 */
39
40 #ifndef __cplusplus
41 }
42 #endif
43
44 #endif /* RNA_H */

```

```

1 #file "RNA.c"
2 #include "RNA.h"
3 #include <math.h>
4 #include <string.h>
5
6 //Estrutura da Rede
7 int noOfInputs[NO_OF_LAYERS] = {6,7}; //numero de entradas em cada camada;
    0 para a escondida e 1 para de saida;
8 int layers[NO_OF_LAYERS] = {7,3}; //numero de neuronios em cada camada; 0
    para a escondida e 1 para de saida;
9 char axonFamilies[] = {'t','t'}; //funções de ativação de cada camada;
    0 para a escondida e 1 para de saida;
10
11 double neuronInput(__psv__ double *weights,int noOfWeights,int index,
    double *input){// funcao que calcula a ativacao(somatorio) do neuronio;
12     /* VARIAVEIS
13         INT
14         i= inteiro de controle;
15
16         DOUBLE
17         summatory = variavel que armazena o somatorio (ativacao)
18     */
19
20     int i;
21

```

```

22  double summatory = 0;
23
24  double Var1, Var2;
25
26  for (i=0;i<noOfWeights;i++){           //percorre o vetor
    de pesos i vezes (o numero de pesos da camada)
27      if (i==0){                         //caso for o primeiro peso:
28          summatory += BIAS * (*(weights+i+index)); //multiplica o
    limiar (no caso o peso 0) pelo BIAS;
29      } else {                             //caso contrário:
30          Var1=*(input+i-1);
31          Var2=*(weights+i+index);
32          summatory += Var1 * Var2;        //multiplica o peso
    pelas entradas;
33      }
34  }
35  return summatory;
36 }
37
38 double neuronOutput(char axonFamily, double activation){
39
40     double output; //variavel que armazena a saída
41     switch(axonFamily){
42         case 'l':                         //funcao de ativacao: linear
43             output = activation;
44             break;
45         case 't':                         //funcao de ativacao: tangente
    hiperbolica
46             output = tanh(activation);
47             break;
48     }
49     return output;
50 }
51
52 void imprime(double num){
53     unsigned char buffer[60];
54     if (num < 0)
55         sprintf(buffer, "%d.%04u ", (int) num, (int) fabs(((num - (int)num) *
    10000)));
56     else
57         sprintf(buffer, " %d.%04u ", (int) num, (int) fabs(((num - (int)num) *
    10000)));
58     sendBufferUsart(buffer);
59 }
60
61 int P=0;
62 unsigned char mlp(__psv__ double *w,__psv__ double *m,double *input){

```

```

63
64  /* VARIÁVEIS
65      INT
66      i, n = inteiros de controle;
67      noOfNeurons = Armazena numero de Neuronios da camada atual;
68      noOfWeights = Armazena numero de Pesos da camada atual;
69      index = variavel de controle para indicar o indice correto do
          vetor de pesos;
70      DOUBLE
71      *weights = ponteiro que aponta para os pesos da camada atual;
72      uh = vetor com os somatorios (ativacoes) dos neuronios da camada
          escondida;
73      yh = vetor com as saidas (apos funcoes de ativacao) dos neuronios
          da camada escondida;
74      uo = vetor com somatorios (ativacoes) dos neuronios da camada de
          saida;
75      yo = vetor com as saidas (apos funcoes de ativacao) dos neuronios
          da camada de saida;
76  */
77
78  int i, n, noOfNeurons, noOfWeights, index;
79  double Max;
80  unsigned char MaxIndex;
81
82  __psv__ double *weights;
83  double uh[NO_OF_INPUTS_HIDDEN_LAYER], yh[NO_OF_INPUTS_OUTPUT_LAYER],
84      uo[NO_OF_OUTPUTS], yo[NO_OF_OUTPUTS];
85
86
87  //P=1;
88  for (i=0; i<NO_OF_LAYERS; i++){ //percorre camada a camada da rede
89      if (i==0){ //aponta a variavel para os pesos referentes a primeira
          camada (escondida);
90      weights = w;
91      }
92      else if (i==1){ //aponta a variavel para os pesos referentes a segunda
          camada (saida);
93      weights = m;
94      }
95      noOfNeurons = layers[i];          //recebe o numero de neuronios de
          cada camada;
96      noOfWeights = noOfInputs[i] + 1; //recebe o numero de pesos de cada
          camada + 1 (limiar do neuronio);
97      for (n=0; n<noOfNeurons; n++){    //percorre todos os neuronios da
          camada atual
98      index = n * noOfWeights;          //recebe o valor referente ao
          indice que aponta o primeiro peso da camada no vetor de

```

```

    pesos correspondente
99     if (i==0){
100         uh[n] = neuronInput(weights,noOfWeights,index,input);
           //calcula somatorio do neuronio e armazena no vetor;
101         yh[n] = neuronOutput(axonFamilies[i],uh[n]);
           //calcula ativação do neuronio e armazena no vetor;
102     } else if (i==1){
103         uo[n] = neuronInput(weights,noOfWeights,index,yh);
           //calcula somatorio do neuronio e armazena no vetor;
104         yo[n] = neuronOutput(axonFamilies[i],uo[n]);
           //calcula ativação do neuronio e armazena no vetor;
105         imprime(yo[n]);
106     }
107 }
108 }
109
110 P=1;
111
112 Max = yo[0];
113 MaxIndex = 0;
114 for (i=1;i< NO_OF_OUTPUTS; i++){
115     if (yo[i]>Max){
116         Max=yo[i];
117         MaxIndex=i;
118     }
119 }
120
121 return MaxIndex;
122 }
```