UNIVERSIDADE DE FORTALEZA - UNIFOR
CENTRO DE CIÊNCIAS TECNOLÓGICAS
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ARTUR GOMES BARRETO

UTILIZAÇÃO DE DEEP LEARNING PARA MODELAGEM DE UM CONTROLADOR NEURAL DE VELOCIDADE DE UM MOTOR CC

FORTALEZA

2019

ARTUR GOMES BARRETO

UTILIZAÇÃO DE DEEP LEARNING PARA MODELAGEM DE UM CONTROLADOR NEURAL DE VELOCIDADE DE UM MOTOR CC

Trabalho de conclusão de curso apresentado à banca examinadora como requisito parcial à obtenção do título de bacharel em Engenharia Elétrica.

Orientador: Prof. Me. Antônio Roberto Lins de Macêdo.

FORTALEZA

2019

Ficha catalográfica da obra elaborada pelo autor através do programa de geração automática da Biblioteca Central da Universidade de Fortaleza

---

Barreto, Artur Gomes.
Utilização de Deep Learning para Modelagem de um Controlador Neural de Velocidade de um Motor CC / Artur Gomes Barreto. - 2019
102 f.

Trabalho de Conclusão de Curso (Graduação) - Universidade de Fortaleza. Curso de Engenharia Elétrica, Fortaleza, 2019.
Orientação: Antônio Roberto Lins de Macêdo.

1. Deep Learning. 2. Deep Belief Network. 3. Controle de Velocidade. 4. Motor CC. 5. Controlador PID. I. de Macêdo, Antônio Roberto Lins. II. Título.

UTILIZAÇÃO DE *DEEP LEARNING* PARA MODELAGEM DE UM CONTROLADOR NEURAL DE VELOCIDADE DE UM MOTOR CC

ARTUR GOMES BARRETO

PARECER: APROVADO

Data: 23 / 11 / 2019

BANCA EXAMINADORA:

VICTOR HUGO COSTA DE ALBUQUERQUE, Dr.

AFONSO HENRIQUES FONTES NETO SEGUNDO, Me.

ANTÔNIO ROBERTO LINS DE MACÊDO, Me.
Orientador(a)

Prof. Bruno Ricardo de Almeida, Dr.
Coordenador do Curso Engenharia Elétrica

Aos meus pais, Adonias e Elda, que sempre me apoiaram incondicionalmente.

AGRADECIMENTOS

Aos meus pais, Adonias Freires Barreto e Elda Gomes Barreto, por terem me dado uma boa educação, não apenas a que adquiri nas boas escolas e na universidade que frequentei, mas aquela que se aprende em casa, norteado por princípios universais da ética.

Aos meus irmãos Sandra Barreto Dantas e Marcelo Gomes Barreto, por sempre estarem ao meu lado, me apoiando em meus desafios.

A minha namorada Grazielen Alencar Lima, pelo companheirismo, pelo seu amor concedido a mim, por sempre me escutar e pelo seu apoio em todos os projetos que me envolvo, não somente com palavras, mas com suas atitudes.

Ao Prof. Me. Antônio Roberto Lins de Macêdo, por me orientar e me abrir portas.

Aos professores participantes da banca examinadora, Dr. Victor Hugo Costa de Albuquerque e Me. Afonso Henriques Fontes Neto Segundo, pelo tempo concedido para a avaliar meu trabalho e pelas colaborações que tanto engradeceram esse projeto.

Ao professor Evilásio Lucena, por ter me orientado no meu primeiro projeto de pesquisa.

À professora Maria das Graças Santos Rufino Pontes, por ter me incentivado a entrar no programa de monitoria no início de meu curso.

Ao professor Ricardo Silva Thé Pontes, por ter me ajudado a desenvolver minhas habilidades com simulações computacionais de motores elétricos.

Ao professor Átila Girão de Oliveira, por ter me orientado no programa de monitoria e por ter me apresentado às redes neurais artificiais.

Ao professor Claudio Sá, por ter me ajudado a definir meu tema de TCC.

Aos professores Garrido, Felisberto, Bruno Lisboa, Dayane, Jéssica Guimarães, José Bento, Daniela Cavalcanti, Sandra Lima, Lia Bitar, Rafaela Cardoso, Joel Sotero e Rodrigo Patrício pela grandiosa contribuição na minha formação profissional.

Aos meus coordenadores Bruno de Almeida e Rodrigo Paulino, por sempre me auxiliarem nas questões acadêmicas e estarem de portas abertas para me receber.

Aos técnicos de laboratório Maike, Emerson, Edmilson e Daniel, por sempre estarem dispostos a me ajudar nas práticas, mesmo fora do horário da aula.

À Liliane Oliveira, por sempre me ajudar em várias situações que enfrentei.

Aos amigos de turma Caio, Raquel, Ricardo, José Pires, Rubem, André e Artur pelo apoio e estudos em grupo durante todos esses anos de universidade.

Sem sombra de dúvidas, é impossível se formar engenheiro eletricista sozinho!

"A inteligência artificial provavelmente irá causar o fim do mundo, mas enquanto isso não acontece, existirão grandes empresas."

(Sam Altman)

RESUMO

O objetivo desse trabalho é modelar uma Rede Neural Artificial (RNA), denominada de Controlador Neural (CN), que controle a velocidade de um motor de Corrente Contínua (CC). Para a modelagem da rede, utilizou-se a linguagem de programação Python e suas *frameworks* de *Machine Learning* TensorFlow e Keras. O CN foi treinado com as entradas e saídas de um controlador PID ligado a um motor CC simulados na *toolbox PID Controller Design for a DC Motor* do Simulink® do MATLAB® R2018a. Utilizou-se um método empírico para construir, avaliar e refinar a topologia da RNA, apresentando sua evolução desde seu modelo inicial até a forma final. A arquitetura utilizada foi a *Deep Belief Network com* retropropagação. A fim de validar o modelo, comparou-se o desempenho do controlador PID com o CN, analisando-se suas respostas transitórias e de regimes permanentes para uma entrada degrau e uma entrada complexa. Essa análise foi feita com base em alguns parâmetros de resposta clássicos da teoria de controle. Analisando os resultados, percebeu-se que o CN é mais lento na busca da velocidade desejada no período transitório. Além disso, possui um erro estacionário maior. Contudo, sua resposta alcança uma estabilidade mais rapidamente e possui um sobressinal menor, em comparação ao controlador PID. Os perfis de velocidade do motor CC gerados pelo CN e pelo PID são bastante próximos, existindo um erro quadrático médio de apenas 0,0097 rad^2/s^2 entre eles. Com os resultados encontrados, afirma-se que é possível utilizar *Deep Learning* e montar uma RNA profunda capaz de aprender a se comportar como um controlador PID e controlar a velocidade de um motor CC.

Palavras-chave: *Deep Learning*. *Deep Belief Network*. Controle de Velocidade. Motor CC. Controlador PID.

ABSTRACT

The objective of this study is to model an Artificial Neural Network (ANN), called by Neural Controller (NC), that controls the speed of a Direct Current (DC) motor. For the network modeling, Python programming language and its Machine Learning frameworks TensorFlow and Keras were used. The NC has been trained with the inputs and outputs of a PID controller connected to a DC motor, both simulated in the Motor PID Controller Design for a DC Motor toolbox of Simulink® from MATLAB® R2018a. An empirical method was used to construct, evaluate and refine the ANN topology, presenting its evolution from its initial model to its final form. The architecture used was the Deep Belief Network with backpropagation. In order to validate the model, the performances of the NC and PID controller were compared, analyzing its transient and steady state responses for a step input and a complex input. This analysis was based on some classical response parameters of control theory. Analyzing the results, it was noticed that the NC is slower in search of the desired speed in the transient period. In addition, it has a larger steady state error. However, its response achieves stability faster and has a lower overshoot compared to the PID controller. The DC motor speed profiles generated by the NC and PID controller are very close, with a mean square error of 0.0097 rad^2/s^2 between them. With the results found, it is stated that it is possible to use Deep Learning to model a deep ANN capable of learning to behave like a PID controller and control the speed of a DC motor.

Keywords: Deep Learning. Deep Belief Network. Speed Control. DC Motor. PID Controller.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CN	Controlador Neural
CC	Corrente Contínua
DBN	*Deep Belief Network* (Rede de Convicção Profunda)
DL	*Deep Learning* (Aprendizagem Profunda)
DNN	*Deep Neural Network* (Rede Neural Profunda)
IA	Inteligência Artificial
IoT	*Internet of Things* (Internet das Coisas)
MBR	Máquina de Boltzmann Restrita
MSE	*Mean Square Error* (Erro Quadrático Médio)
MGD	Método do Gradiente Descendente
NA	Neurônio Artificial
PID	Proporcional Integrativo Derivativo
RNA	Rede Neural Artificial

LISTA DE SÍMBOLOS

b	Coeficiente de Atrito Viscoso do Motor
Σ	Combinador Linear
K_e	Constante de Força Eletromotriz
K_t	Constante de Torque do Motor
k_1	Constante que Modela as Características Construtivas do Motor
I_a	Corrente na Armadura
I_f	Corrente no Campo
N	Divisor de Filtro Derivado de Primeira Ordem
$u(t)$	Entrada Degrau Unitário
$e(t)$	Erro do Sinal de Saída do Controlador PID
E_s	Erro Estacionário
ER	Erro ou Estimativo do Erro
PV	Estado Atual do Sistema (*Process Value*)
SP	Estado Desejado para o Sistema (*Set Point*)
Φ	Fluxo Magnético Estabelecido no Entreferro
E	Força Eletromotriz Induzida (Força Contraeletromotriz da Armadura)
s	Frequência Complexa
g	Função de Ativação
k_d	Ganho Derivativo
k_i	Ganho Integral
k_p	Ganho Proporcional
L_a	Indutância das Bobinas da Armadura
L_f	Indutância das Bobinas de Campo
L	Indutância Elétrica
τ	Intervalo de Integração
θ	Limiar de Ativação
J	Momento de Inércia do Rotor
∇	Operador *nabla*
w_i	Peso Sináptico
u	Potencial de Ativação
R_a	Resistência do Circuito da Armadura

R_f	Resistência do Circuito do Campo
R	Resistência Elétrica
$c(\infty)$	Resposta do Sistema em Regime Estacionário no Domínio do Tempo
$G(s)$	Resposta do Sistema no Domínio da Frequência
$c(t)$	Resposta do Sistema no Domínio do Tempo
x_i	Sinais de Entrada
y	Sinal de Saída
M_p	Sobressinal Máximo
n_{apr}	Taxa de Aprendizagem da RNA
t	Tempo
t_s	Tempo de Acomodação
t_d	Tempo de Atraso
t_p	Tempo de Pico
t_r	Tempo de Subida
V	Tensão
U_a	Tensão Aplicada nos Terminais da Armadura
U_f	Tensão Aplicada nos Terminais do Campo
ω	Velocidade Angular
n	Velocidade de Rotação do Rotor

SUMÁRIO

1 INTRODUÇÃO

Esse capítulo apresenta o tema do trabalho, sua delimitação e a problemática que será analisada e resolvida. É apresentado o contexto em que o tema está inserido, bem como a importância do tema para a área de estudo. Também são apresentados os objetivos da pesquisa e as metodologia adotadas.

1.1 Contextualização

Uma Rede Neural Artificial (RNA) realiza um mapeamento entre as entradas e as respectivas saídas de um problema, estabelecendo uma correlação entre elas. Além disso, a RNA é capaz de realizar uma aproximação para os valores desconhecidos, possibilitando uma capacidade de generalização. Essas e outras características fundamentais das RNAs foram estabelecidos por Warren S. Mcculloch e Walter Pitts em 1943 no artigo *A Logical Calculus of the Ideas Immanent it Nervous Activity* (MCCULLOCH; PITTS, 1943).

Apesar de ser um conceito antigo, a utilização da RNA somente se tornou expressiva nas duas últimas décadas, sendo aplicada na classificação de dados, agrupamento de dados semelhantes (*clustering*), reconhecimento de padrões e predição de valores futuros (ABIODUN *et al*., 2018).

Essa demora em sua popularização ocorreu porque o ambiente adequado que permitiu que as RNAs fossem utilizadas para resolver problemas complexos só surgiu em meados dos anos 2000 devido a evolução do *hardware* dos computadores, a disponibilização de grandes conjuntos de dados na Internet e avanços nos algoritmos de *machine learning* (CHOLLET, 2018).

Segundo Chollet (2018), a indústria dos jogos eletrônicos foi fundamental para a consolidação das RNAs, pois esse mercado proporcionou o surgimento de placas de vídeo de alto rendimento que, hoje em dia, são utilizadas para rodar essas redes. Além disso, as RNAs precisam de um considerável volume de dados para serem treinadas e o agrupamento de dados, das mais diversas naturezas, como números em tabelas, sons, imagens e vídeos foi impulsionado com a expansão da Internet. Por fim, Chollet afirma que o surgimento de várias pequenas melhorias algorítmicas permitiu que as RNAs tivessem mais camadas, constituindo redes mais profundas. Essas redes são conhecidas como *Deep Neural Networks* (DNN) e sua

área de estudo é chamada de *Deep Learning* (DL). Com as DNNs é possível resolver problemas mais complexos e relevantes em comparação as redes neurais com menos camadas.

Graças a essas condições favoráveis, as RNAs vem sendo aplicadas cada vez mais em diversas áreas do conhecimento com os mais variados objetivos. Como por exemplo, pode-se utilizar RNAs na detecção de falhas em sistemas de distribuição de energia elétrica (WANG *et al*., 2019), na análise de textos provenientes de redes sociais para que as empresas respondam as mensagens dos clientes de forma mais eficiente (VERMEER *et al*., 2019) e na predição da qualidade do ar em grandes cidades (MA *et al*., 2019).

Além dessas aplicações, é possível utilizar DNNs para criar uma visão computacional para que o computador entenda e processe imagens com precisão (CHO; TAI; KWEON, 2019).

Outra área na qual as DNNs vem sendo cada vez mais utilizadas é a da detecção de padrões e seus vários subdomínios como o reconhecimento de vozes, a regressão, a predição de valores e as classificações (LECUN; BENGIO; HINTON, 2015).

Entre outras aplicações, destacam-se o diagnóstico de câncer de mama (YUE *et al*., 2018), a detecção de intrusos, *malware* e *spam* em computadores (APRUZZESE *et al*., 2018) e a análise de dados provenientes da *Internet of Things* (IoT) para obter informações, prever comportamentos futuros e tomar decisões de controle (MOHAMMADI *et al*., 2018).

Não demorou para que os algoritmos de Inteligência Artificial (IA), em especial os de *Deep Learning*, começassem a ser aplicados no campo do controle e automação. Por exemplo, um controle inteligente da turbina eólica é proposto por Mahmoud (2017). Neste trabalho, um sistema de controle automático inteligente e integrado é desenvolvido para aumentar a energia extraída dos ventos e melhorar o desempenho do sistema de potência.

O controle Proporcional Integral Derivativo (PID) é um conceito bastante consolidado e utilizado em aplicações industriais. No entanto, seu desempenho para sistemas não lineares varia e depende do ajuste dos parâmetros do controlador, ou seja, o ganho proporcional ($\mathrm{k_p}$), o ganho integral (k_i) e o ganho derivativo (k_d) (ANG *et al.*, 2006). Além desses parâmetros, o desempenho do controlador PID depende dos próprios parâmetros do sistema o qual ele está controlando como o momento de inércia do rotor, no caso do controle de velocidade de um motor.

Por exemplo, no caso de uma válvula motorizada usada na aplicação de polímeros em uma indústria, a fricção da válvula tende a aumentar se ela não for operada por um período. Isso ocorre devido ao depósito de partícula (ABRAHAM; SHRIVASTAVA, 2018). Isso resulta

na mudança no momento de inércia do rotor. Caso seja utilizado um controlador PID clássico, ele não será capaz de fornecer uma saída adequada porque os parâmetros do sistema que foram utilizados em sua calibração consideram um sistema sem esse atrito adicional. Logo, os parâmetros do controlador PID não correspondem a atual situação do sistema. Nesse caso, o controlador PID pode produzir *overshoots*, pois o desempenho do controlador se deteriora devido a alterações nos parâmetros do sistema.

Segundo Abraham e Shrivastava (2018), as estratégias de controle com autoajuste de parâmetros são adequadas para sistemas com características que variam no tempo. Controladores que utilizam *deep learning* são capazes de realizar esses autoajustes e estão sendo aplicados no controle de processos não-lineares complexos. Por exemplo, Cheon *et al.* (2015) propôs um controlador de aprendizado profundo para o controle de velocidade de um motor de corrente contínua. Utilizando algoritmos de *deep learning*, os autores modelaram um controlador com uma tolerância maior para alterações nos parâmetros do sistema em relação ao tempo. O maior benefício proporcionado pelo controlador proposto foi a atenuação das perturbações no processo. Esse controlador inteligente foi modelado com base nas entradas e saídas de um controlador PID padrão.

Os motores elétricos são bastante utilizados em diversas aplicações na indústria, sendo os responsáveis pela maior parte do consumo de energia elétrica do ramo (TRIANNI; CAGNO; ACCORDINI, 2019). Desde braços de robôs usado na indústria automobilística até bombas em motores de foguetes, os motores constituem os principais elementos de controle de diversas aplicações (ABRAHAM; SHRIVASTAVA, 2018).

1.2 Problema Central

Considerando o contexto apresentado, esse trabalho se propõe em utilizar a *deep learning* no controle de velocidade de motores de forma robusta, priorizando tanto a velocidade da resposta como sua acurácia. Utilizando um algoritmo de *deep learning* chamado de *Deep Belief Network* (DBN), uma RNA foi modelada para imitar o comportamento de um controlador PID e assim controlar a velocidade do motor CC. Esta RNA foi denominada de Controlador Neural (CN).

1.3 Justificativa

O Controlador Neural se mostra mais tolerante as mudanças dos parâmetros do sistema, tal como a variação do momento de inércia. Ele possui boas velocidade de resposta e acurácia em regime permanente. Além disso, não é necessário realizar uma recalibração na ocorrência de mudanças dos parâmetros do sistema.

1.4 Hipótese

É possível realizar o controle de velocidade de um motor CC utilizando uma rede neural profunda (*Deep Neural Network*) treinada com base nas entradas e saídas de um controlador PID clássico.

1.5 Metodologia

Segundo Silva e Menezes (2005), uma pesquisa deve ser classificada quanto à natureza, à abordagem do problema, aos objetivos e aos procedimentos técnicos.

A natureza da nossa pesquisa é aplicada, pois objetiva gerar conhecimentos para aplicações práticas dirigidas à solução de um problema específico.

Quanto a abordagem do problema, o estudo tem tanto características quantitativas como qualitativas.

Em relação aos objetivos da pesquisa, pode-se dizer que ela é descritiva, pois visa descrever as características do modelo utilizado para resolver um problema de controle.

Quanto aos procedimentos técnicos adotados neste estudo, utilizou-se a pesquisa bibliográfica, elaborada através de livros e artigos publicados em revistas de relevância mundial como IEEE e *Science Direct*. Também se utilizou de um estudo de caso para confirmar a hipótese levantada.

1.6 Objetivos

O objetivo geral do trabalho é modelar e treinar uma rede neural profunda, utilizando dados provenientes de simulações, que seja capaz de realizar o controle de velocidade de um motor CC de maneira robusta.

Os objetivos específicos são:

- Realizar um estudo bibliográfico sobre *Machine Learning*, em especial o *Deep Learning*, e apresentar seus principais conceitos e algoritmos;
- Realizar um estudo bibliográfico sobre controladores PID e motores CC e apresentar seus principais conceitos;
- Apresentar as principais características de todas as ferramentas utilizadas;
- Modelar e simular um motor CC e um controlador PID utilizando o MATLAB® R2018a e o Simulink®;
- Aplicar um método de modelagem de problemas de DL;
- Realizar uma análise comparativa entre o CN modelado e o controlador PID.

1.7 Estrutura do Trabalho

No capítulo 2, apresentamos todo o referencial teórico básico necessário para o entendimento desse estudo. Iniciamos falando sobre inteligência artificial, *machine learning* e *deep learning*. Em seguida, abordamos sobre o neurônio artificial, as redes neurais artificias e a arquitetura *Deep Belief Network*. Fechamos esse capítulo dissertando sobre os motores elétricos de corrente contínua e sobre os controladores PID.

O capítulo 3 consiste no desenvolvimento do nosso estudo de caso. Iniciamos esse capítulo apresentando todas as ferramentas utilizadas, seguido da visão geral do projeto, a caracterização do problema e a descrição do método proposto. Prosseguindo no estudo de caso, apresentamos as modelagens do motor CC, do controlador PID e a simulação desse sistema como um todo. Ainda nesse capítulo, apresentamos e aplicamos um método para tratar problemas de *Deep Learning*. Esse método tem como objetivo orientar o processo de evolução topológico da RNA, do seu estado inicial até o modelo final. Finalmente, com o modelo determinado, realizamos as análises das respostas do CN proposto.

No capítulo 4, apresentamos nossas conclusões e recomendações para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Esse capítulo apresenta os principais fundamentos teóricos utilizados na realização da pesquisa. Os conceitos necessários para o entendimento do estudo de caso estão definidos nas subseções a seguir.

Segundo Chollet (2018), o primeiro passo no estudo dos sistemas inteligentes é saber a diferença entre Inteligência Artificial (IA), *Machine Learning* (ML) e *Deep Learning* (DL). Devemos saber qual é a proposta de cada área para escolheremos as ferramentas corretas para resolvermos um dado problema de nosso interesse.

2.1 Inteligência Artificial

A IA nasceu em meados de 1950 quando os primeiros estudiosos da então recente ciência da computação começaram a se perguntar se os computadores seriam capazes de pensar como um ser humano. Nesse contexto, a IA se propõe em automatizar tarefas intelectuais normalmente realizada por humanos (CHOLLET, 2018).

A IA é um campo bastante vasto que engloba tanto as abordagens com aprendizagem - o *machine learning* e o *deep learning* - como abordagens sem aprendizagem, as chamadas IAs simbólicas. Esses sistemas consistem em programas com muitas regras explícitas que tentam modelar um determinado conhecimento. As IAs simbólicas não realizam um aprendizado propriamente dito, mas apenas seguem um longo roteiro explícito escrito pelos programadores para resolver um problema bem definido. Como exemplo desses sistemas, pode-se citar os primeiros programas de xadrez (LAZZERI; HELLER, 1996).

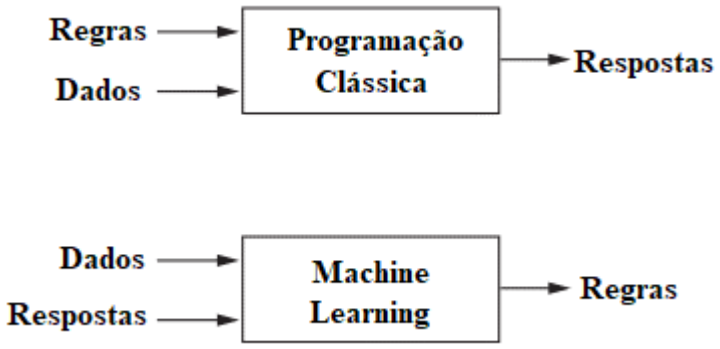
O escopo deste trabalho se delimita nos algoritmos que possuem aprendizagem. Portanto, não entraremos em detalhes da grande área da IA.

2.2 *Machine Learning*

Embora a IA simbólica tenham se mostrado adequada para resolver problemas lógicos bem definidos, ela não obteve bons resultados em resolver problemas mais complexos como a classificação de imagem e o reconhecimento de fala porque não existe regras explícitas diretas para modelar esses problemas. Segundo Silva *et al.* (2016), o *machine learning* surgiu para resolver estes e outros tipos de problemas complexos.

Chollet (2018) afirma que na programação clássica da IA simbólica, os programadores alimentam o sistema com as regras e os dados a serem processados para receberem as respostas como saída. Já no *machine learning*, os programadores alimentam o sistema com os dados e as respostas esperadas dessas entradas para receberem como saída as regras que correlacionam o que foi passado. Essas relações de entrada e saída da programação clássica e do *machine learning* podem ser visualizados na Figura 1 abaixo.

Figura 1 – Entradas e Saídas da Programação Clássica e do ML



Fonte: Adaptado de Chollet, 2018.

Analisando a Figura 1, fica evidente que no *machine learning* o sistema é treinado em vez de simplesmente alimentado com as regras prontas pelos programadores. O próprio sistema encontra as regras que modelam o problema e, de posse dessas regras, será capaz de definir saídas para entradas até então desconhecidas devido sua capacidade de generalização. Para isso, ele deve ser treinado, ou seja, apresentado algumas vezes a vários exemplos relevantes ao problema. Portanto, o programa aprende através da exposição aos dados e a repetição, tal como um ser humano faria (ABIODUN *et al.*, 2018).

No treinamento do sistema, além dos dados de entrada e das respectivas saídas esperadas, é necessário especificar uma forma de mensurar o quão bem o algoritmo está resolvendo o problema. Esta medição é feita através do cálculo da função do erro (*loss function*) que consiste em determinar, para uma dada entrada, a distância entre a saída calculada pelo algoritmo e a saída esperada. O cálculo da função do erro é utilizado como sinal de *feedback* para ajustar o funcionamento do algoritmo. Este processo de ajuste, chamado de aprendizagem, tem como objetivo minimizar este erro (SILVA, IVAN; SPATTI, DANILO; FLAUZINO, 2016).

Nesse processo de aprendizagem, os algoritmos de ML realizam constantes transformações nos dados, fazendo com que eles sejam representados de forma cada vez mais significativa para o sistema, tornando a saída calculada progressivamente mais próxima da saída

esperada. Essa busca sistemática e automática por melhores representações dos dados apresentados consiste no cerne do processo de aprendizagem e ocorre dentro de um conjunto pré-definido de operações chamado de espaço de hipótese (CHOLLET, 2018).

Resumindo, os algoritmos de ML realizam transformações nos dados para representá-los de maneira mais significativas objetivando resolver um determinado problema de maneira mais satisfatória. Essas transformações se dão dentro de um espaço pré-definido de possibilidades e utilizam um sinal de *feedback* como direcionamento.

É importante destacar que todos os conceitos dessa seção serão abordados mais detalhadamente nas próximas seções do trabalho.

2.3 *Deep Learning*

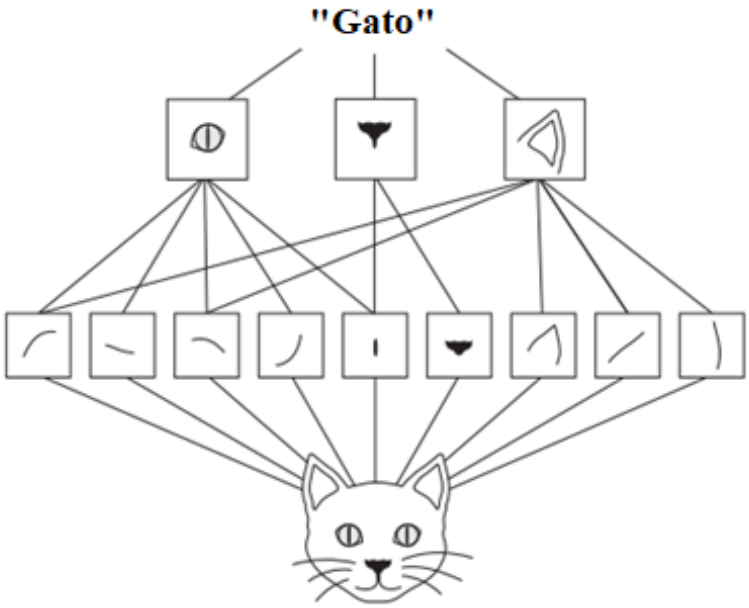
O *Deep Learning* (Aprendizagem Profunda) é uma subárea do ML que aplica sucessivas camadas de transformações nos dados de tal forma que eles são representados de maneira cada vez mais significativa para a solução do problema, tal como é feito no ML. Entretanto, a diferença está na quantidade de transformações realizadas (LECUN; BENGIO; HINTON, 2015). O termo profundo em sua designação diz respeito à quantidade de camadas de transformação presentes nesses sistemas, bem maior do que nos algoritmos de ML. Quanto mais camadas presentes, mais profundo o modelo é considerado.

Maier *et al.* (2019) apresenta um exemplo bastante didático deste conceito aplicado na detecção de padrões em imagens. Tomando um conjunto de imagens, após sucessivas entradas deste dados em um sistema DL, a primeira camada, aplicando uma determinada transformação, consegue identificar a presença ou ausência de bordas em orientações e localizações definidas. Já a segunda camada, aplicando uma outra transformação, detecta combinações das bordas encontradas pela primeira camada, identificando formas geométricas um pouco mais complexas. A terceira camada é capaz de detectar padrões ainda mais complexos e assim sucessivamente. Com um número suficientemente grande de camadas, é possível detectar padrões em imagens, desde simples objetos do nosso cotidiano até câncer de mama em estágios iniciais (MEHDY *et al.*, 2017).

Um outro exemplo de classificação de imagens é dado por Chollet (2018) e pode ser visto na Figura 2 abaixo. O algoritmo aplica uma transformação na imagem do gato na primeira camada e com isso consegue identificar bordas simples. Na segunda camada, essas bordas são combinadas e transformadas novamente e o algoritmo identifica a presença de olhos,

nariz e orelhas. Por fim, após outra transformação, o sistema combina essas formas mais complexas e consegue identificar que o objeto presente na imagem é um gato.

Figura 2 – Identificação de um Gato



Fonte: Adaptado de Chollet, 2018.

Vale ressaltar que essas transformações aplicadas em cada camada não são especificadas pelos programadores, mas sim aprendidas pelo sistema durante o processo de aprendizagem a partir dos dados de entrada (LECUN; BENGIO; HINTON, 2015). Além disso, os modelos construídos com DL costumam possuir entre dezenas e centenas de camadas enquanto que abordagens em ML possuem 1 ou 2 camadas (CHOLLET, 2018).

Tanto os modelos de ML como os de DL são implementados através de agrupamentos de Neurônios Artificiais (NAs) que constituem as chamadas Redes Neurais Artificiais (RNAs). Para que possamos nos aprofundar nos algoritmos de DL, devemos conhecer os principais conceitos das RNAs. Portanto, as próximas subseções servirão para este propósito.

2.4 Neurônio Artificial

O elemento básico das redes neurais artificiais é o neurônio artificial. Estes dois conceitos foram desenvolvidos a partir dos modelos matemáticas de suas contrapartes

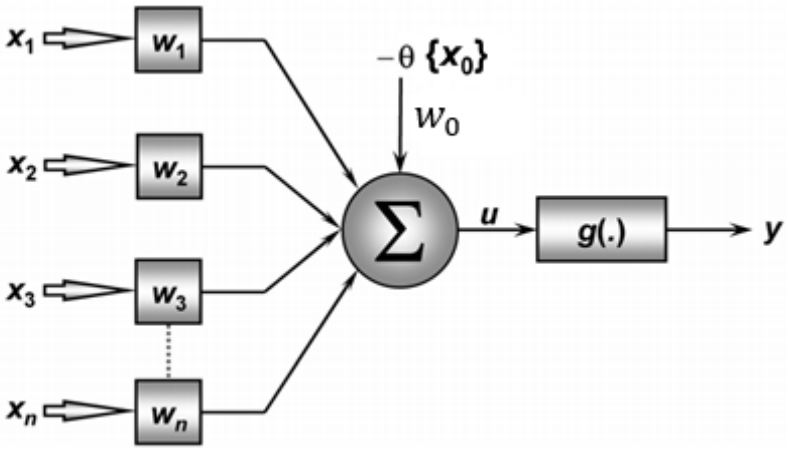
biológicas presentes no cérebro humano (SILVA, IVAN; SPATTI, DANILO; FLAUZINO, 2016).

Entretanto, os modelos atuais dos NAs simplificam consideravelmente os neurônios biológicos. Então, mesmo que o termo rede neural e que alguns outros conceitos centrais do *machine learning* tenham sido desenvolvidos, pelo menos em parte, por inspiração em nossa compreensão do cérebro humano, os modelos artificiais não tem a pretensão de serem modelos do cérebro (CHOLLET, 2018). Portanto, fica esclarecido que não há evidências científicas que afirmem que o cérebro realize algo semelhante aos mecanismos de aprendizado usados nos modelos de *machine learning* ou *deep learning*.

O marco zero da modelagem dos NAs é o estudo realizado por McCulloch e Pitts (1943) no qual eles propuseram um modelo para o neurônio biológico com grande paralelismo e alta conectividade, características também presentes no modelo da sua contraparte artificial. Outro trabalho que contribuiu consideravelmente para o modelo das NAs foi o do Hodgking e Huxley (1952) no qual eles analisaram a geração e propagação dos impulsos elétricos na membrana celular dos neurônios biológicos.

A Figura 3 abaixo ilustra o modelo de um NA típico. Segundo Braga *et al.* (2007), os NAs coletam sinais de entrada, agrega-os de acordo com sua função operacional e, por fim, aplicam uma função de ativação, gerando saídas tipicamente em valores contínuos. Vamos analisar cada uma dessas etapas a seguir.

Figura 3 – Modelo do Neurônio Artificial



Fonte: Silva *et al.,* 2016.

Verificamos na Figura 3 que o NA funciona a partir de sete elementos: Os sinais de entrada (x_1, x_2, ..., x_n), os pesos sinápticos (w_0, w_1, w_2, ..., w_n), o combinador linear (Σ), o

limiar de ativação (θ), o potencial de ativação (u), a função de ativação (g(.)) e o sinal de saída (y).

Os sinais de entrada (x_1, x_2, ..., x_n) são os dados externos que serão trabalhados na rede neural e representam os valores das variáveis de um dado problema. Costuma-se normalizados estes dados, ou seja, coloca-los em uma mesma escala a fim de aumentar a eficiência do treinamento (HAYKIN, 2003).

Os pesos sinápticos (w_0, w_1, w_2, ..., w_n) servem para ponderar cada uma das entradas e quantificam a relevância de cada uma delas à ativação ou não do respectivo neurônio. São eles que armazenam o aprendizado e realizam as transformações nos dados. Estes pesos são modificados durante o processo de treinamento (CHOLLET, 2018).

O combinador linear (Σ) tem como função somar todas as entradas ponderadas com seus respectivos pesos sinápticos, ou seja, $\Sigma w_{\mathrm{i}}.x_i = w_1.x_1 + w_2.x_2 + \cdots + w_n.x_n$.

O limiar de ativação (θ) tem como finalidade especificar o valor mínimo que o resultado gerado pelo combinador linear $\Sigma w_{\mathrm{i}}.x_i$ deve ter para que o neurônio se ative, ou seja, dispare.

O potencial de ativação (u) consiste na diferença entre o valor do combinador linear $\Sigma w_{\mathrm{i}}.x_i$ e o limiar de ativação θ ponderado com seu peso w_0, ou seja, $u = \Sigma w_{\mathrm{i}}.x_i - w_0.\theta$. Se seu valor for positivo ($\Sigma w_{\mathrm{i}}.x_i \geq w_0.\theta$) o neurônio terá um potencial excitatório, ou seja, se ativa (dispara). Caso contrário, o potencial será inibitório, ou seja, permanece desativado (sem disparo) (KOVÁCS, 2002).

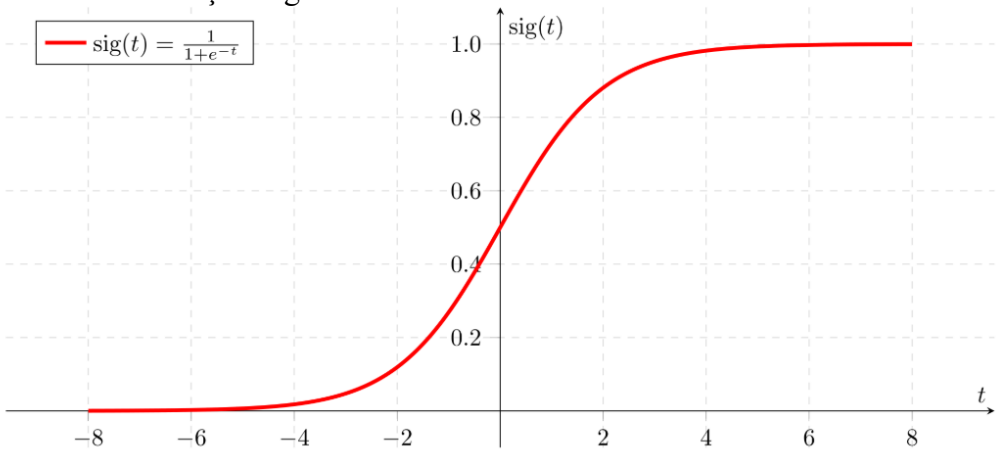
A função de ativação (g(.)) serve para limitar a saída do neurônio dentro de um intervalo de valores. Esse intervalo depende de qual tipo de função de ativação é utilizada. A função sigmóide e a tangente hiperbólica são as mais comuns (SILVA, IVAN; SPATTI, DANILO; FLAUZINO, 2016). Ressaltamos que utilizamos a função sigmóide no nosso estudo de caso.

Uma função sigmóide consiste em uma função matemática que possui uma curva em forma de "S" característica, chamada de curva sigmóide. A Equação (1) modela essa função (MITCHELL, 1997).

$$g(u) = \frac{1}{1 + e^{-u}} \tag{1}$$

Segundo Mitchell (1997), uma função sigmóide é uma função limitada e diferençável. O seu domínio é o conjunto dos números reais R e a sua imagem é um conjunto de números reais entre 0 e 1, conforme pode ser visto no Gráfico 1 abaixo.

Gráfico 1 – Função Sigmóide



Fonte: Arunava, 2018.

O sinal de saída (y) consiste no valor final produzido pelo neurônio. Este valor costuma ser um número real e é limitado pela função de ativação (HAYKIN, 2003). O sinal de saída é equacionado pela expressão $y = g(u)$. Logo, a saída do neurônio é o resultado da aplicação da função de ativação no potencial de ativação.

Vale salientar que, em uma rede neural, as saídas dos neurônios mais profundos são as entradas dos neurônios que estão sequencialmente interligados a eles, como veremos com mais detalhes na próxima seção.

Considerando todas essas informações, podemos resumir o funcionamento dos neurônios artificiais nas seguintes etapas:

1) Passagem de um conjunto de dados que representam as variáveis de entrada do neurônio;
2) Multiplicação de cada entrada x_i recebida pelo peso sináptico w_i correspondente, caracterizando uma transformação nos dados na tentativa de melhor representá-los;
3) Cálculo do potencial de ativação u através da equação $u = \sum_{i=1}^{n}(w_i . x_i) - w_0 . \theta$;
4) Determinação da saída y através da aplicação da função de ativação g(.) no potenciação de ativação u conforme a equação $y = g(u)$.

2.5 Redes Neurais Artificiais

Essa subseção se dedica ao estudo da combinação dos NAs na formação das Redes Neurais Artificiais (RNAs). Iniciamos apresentando a diferença entre os conceitos de arquitetura e topologia das RNAs. Em seguida, apresentamos as principais arquiteturas das RNAs. Finalizamos este tópico abordando os principais tipos de treinamento.

2.5.1 Principais Conceitos

Segundo Silva (2016), a arquitetura de uma RNA consiste na forma como os seus NA estão dispostos uns em relação aos outros, definindo a forma como são feitas as conexões sinápticas entre eles. Considerando uma determinada arquitetura, a topologia de uma RNA diz respeito as diferentes formas de composições estruturais que a rede pode assumir.

Para tornar esses conceitos mais claros, considere uma determinada arquitetura de RNA com 2 camadas. Para esse caso, podem existir diversas topologias. Uma delas pode conter 5 neurônios por camada e uma outra 15 neurônios. Ou então, uma topologia pode utilizar a função de ativação sigmóide e uma outra a função tangente hiperbólica. Em resumo, a arquitetura diz a forma como os neurônios estão ligados entre si e a topologia trata da quantidade e da natureza desses neurônios (BRAGA, ANTÔNIO; CARVALHO, ANDRÉ; LUDERMIR, 2007).

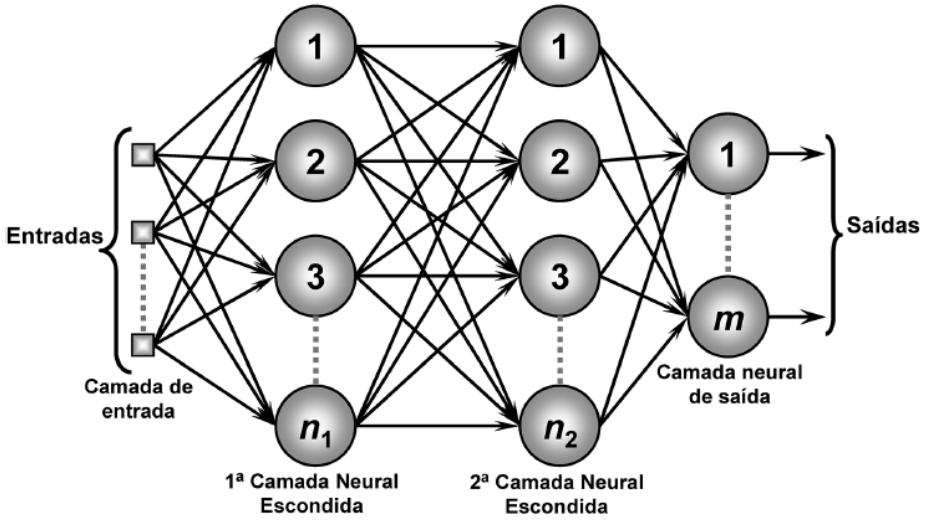
O treinamento é um outro conceito importante para o estudo das RNAs e consiste em um processo que se dá através da aplicação de uma série de passos ordenados com o objetivo de ajustar os pesos w_i e os limiares θ_i de todos os neurônios da rede. Esse processo, também conhecido como aprendizagem, faz com que a rede aprenda gradualmente a resolver o problema, tornando suas respostas cada vez mais próximas dos valores desejados (KOVÁCS, 2002).

2.5.2 Arquiteturas das RNAs

Uma RNA se divide em três parte: A camada de entrada, as camadas escondidas (também chamadas de ocultas, intermediárias ou invisíveis) e a camada de saída (SILVA, IVAN; SPATTI, DANILO; FLAUZINO, 2016). Cada camada consiste em um agrupamento de 1 ou mais neurônios em um mesmo nível, ou seja, não conectados entre si. A Figura 4 abaixo ilustra

uma RNA genérica com uma camada de n entradas, 2 camadas neurais escondidas com, respectivamente, n_1 e n_2 neurônios e uma camada neural com m saídas.

Figura 4 – Exemplo de RNA



Fonte: Silva *et al.,* 2016.

A camada de entrada é aquela que recebe os dados do meio externo que serão processados. Esses sinais costumam ser normalizados para uma mesma faixa de valores para que se obtenha uma melhor precisão numérica nas operações matemáticas realizadas na RNA (HAYKIN, 2003).

As camadas escondidas são aquelas responsáveis por realizar as transformações nos dados, tornando-os mais representativos para a solução do problema. Os NAs dessas camadas extraem as características fundamentais dos dados e armazenam esses conhecimentos em seus pesos sinápticos w_i e limiares θ_i (CHOLLET, 2018). A maior parte do processamento realizado em uma RNA ocorre nessas camadas.

A camada de saída é aquela que possui os neurônios que são responsáveis pelo cálculo e apresentação do resultado da RNA, com base no processamento realizado nas camadas escondidas. Segundo Braga *et al. (*2007), é nesta camada que a resposta calculada pela RNA é comparada com a resposta esperada e o sinal de *feedback* é gerado e repassado para o resto da rede com o intuito de direcionar o ajuste de todos os pesos sinápticos w_i, buscando minimizar o erro e assim, de maneira gradual, fazer a rede aprender e armazenar o conhecimento.

As principais arquiteturas das RNAs, levando-se em conta a disposição e as interligações entre os NAs e como as camadas são constituídas, são, de acordo com Silva *et al.* (2016):

1) Redes *feedforward* de camada simples;
2) Redes *feedforward* de camadas múltiplas;
3) Redes Recorrentes;
4) Redes Reticuladas.

Dado o escopo do nosso trabalho, estudaremos com maior profundidade apenas as redes *feedforward* de camadas múltiplas. Tanto essa arquitetura básica como suas variações são bastante comuns na literatura e se mostraram capazes de resolver problemas de diversas naturezas, tais como classificação de padrões, aproximação de funções, visão computacional e controle de processos (HAYKIN, 2003). Mais detalhes dessa arquitetura serão dados na subseção 2.6.

2.5.3 Treinamento da RNA

O último conceito básico que iremos definir nessa subseção é o treinamento. Uma das características mais interessantes das RNAs é a sua capacidade de aprender a partir da apresentação de amostras de dados (KOVÁCS, 2002). Se esse conjunto de dados for representativo e exprimir o comportamento daquilo que se deseja inferir, a rede será capaz, após ter passado pelo processo de treinamento, de generalizar soluções, ou seja, de calcular respostas para entradas que ela, até então, não teve contato (CHOLLET, 2018).

Resumindo o que Chollet (2018) e Kovács (2002) falam sobre o treinamento, podemos dizer que este processo consiste em um conjunto de passos aplicados em uma sequência determinada objetivando ajustar tanto os pesos w_i como os limiares θ_i de todos os neurônios da rede. Este processo tem como objetivo final a generalização das soluções a serem produzidas pelas saídas da RNA de forma que representem o sistema físico que está sendo modelado.

Para um determinado problema, costuma-se dividir, de maneira aleatória, o conjunto total de dados de amostras disponíveis em 3 subconjuntos, denominados de conjunto de treinamento, conjunto de validação e conjunto de testes (CHOLLET, 2018).

O conjunto de treinamento possui entre 50 e 80% das amostras do conjunto total e é utilizado durante o processo de treinamento, mais especificamente na extração das características do problema, ou seja, no mapeamento da relação entre as entradas e saídas e criação das regras que modelam o conhecimento da rede (BRAGA, ANTÔNIO; CARVALHO, ANDRÉ; LUDERMIR, 2007).

Já o conjunto de validação também é utilizado no processo de treinamento, mas não na extração das características do problema. Utiliza-se os dados desse conjunto apenas para se calcular a taxa de acerto da RNA e determinar se a rede está aprendendo generalizações do problema ou apenas se sobreajustando aos dados apresentados, caracterizando o *overfitting* (CHOLLET, 2018). Este conjunto possui entre 10 e 25% das amostras do conjunto total.

Por fim, o conjunto de testes possui entre 10 e 25% das amostras do conjunto total e é utilizado após o treinamento. Este conjunto serve para determinar o nível da generalização alcançado pela RNA (SILVA, IVAN; SPATTI, DANILO; FLAUZINO, 2016) e o resultado obtido em sua utilização serve para validar a topologia testada na modelagem do problema (KOVÁCS, 2002). Por exemplo, caso o resultado da aplicação desse conjunto de dados resulte em uma taxa de acerto baixa, significa que a topologia utilizada não é adequada para resolver o problema, devendo-se alterar alguns parâmetros da rede, tal como o número de neurônios por camada, para buscar um modelo que traga resultados melhores.

Segundo Silva (2016), há 5 tipos básicos de treinamento:

1) Treinamento supervisionado;
2) Treinamento não-supervisionado;
3) Treinamento com reforço;
4) Aprendizagem usando lote de padrões (*off-line*);
5) Aprendizagem usando padrão-por-padrão (*on-line*).

Levando em consideração o escopo de nosso trabalho, precisamos detalhar apenas o treinamento supervisionado e o não-supervisionado, pois os outros tipos não foram utilizados no estudo de caso.

O treinamento supervisionado é o tipo mais comum de treinamento e já foi apresentado de maneira intuitiva no decorrer desse trabalho. Nesse tipo de aprendizagem, tem-se de antemão a resposta esperada para cada entrada. A diferença entre a resposta calculada pela RNA e a resposta esperada é continuamente mensurada e usada nos ajustes dos pesos sinápticos w_i e dos limiares de ativação θ_i. O treinamento é finalizado quando essa diferença for menor que um determinado valor especificado pelo programador (HAYKIN, 2003).

Já no treinamento não-supervisionado, não se tem de antemão as saídas desejadas. Nesse caso, a RNA ajusta seus pesos w_i e limiares θ_i procurando identificar e agrupar os dados em subconjuntos que possuem similaridades - os chamados de *clusters* - buscando refletir e generalizar essas representações (SILVA, IVAN; SPATTI, DANILO; FLAUZINO, 2016).

2.6 RNAs de Camadas Múltiplas

Vimos que os algoritmos de ML servem para mapear as entradas e saídas de um problema, definindo a relação entre elas e que isso é feito por meio de uma sequência de transformações nos dados que são realizadas por neurônios dispostos em camadas. Por sua vez, estas transformações são aprendidas no processo de treinamento através da passagem de várias amostras de dados. Com essa visão descritiva em mente, chegou a hora de ver como tudo isso ocorre de maneira concreta.

2.6.1 Treinamento da RNA de Camadas Múltiplas

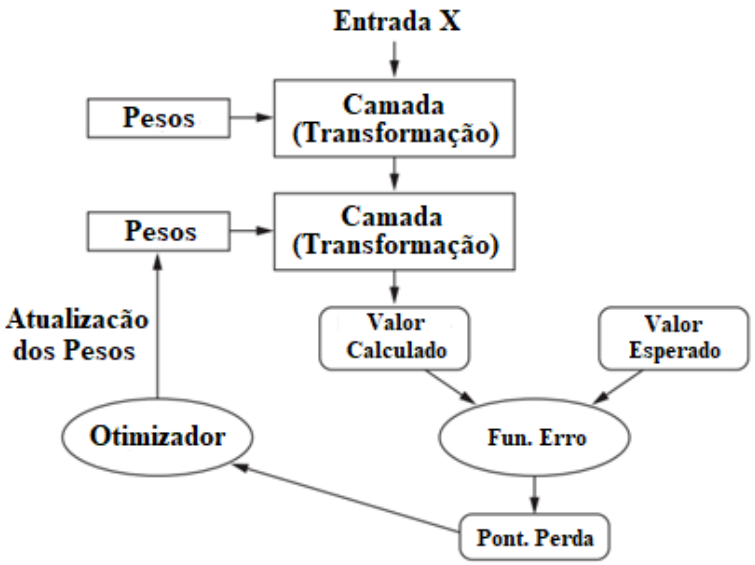
As transformações realizadas em cada camada são armazenadas nos pesos w_i e limiares θ_i de seus NAs. Portanto, podemos dizer que os pesos e limiares são os parâmetros que definem a transformação que é realizada em uma camada (CHOLLET, 2018). Assim, podemos chamar os pesos w_i e limiares θ_i dos NAs de uma camada de parâmetros da camada e o conjunto de todos os parâmetros de todas as camadas de parâmetros da rede.

Portanto, fazer uma RNA aprender significa encontrar um conjunto de parâmetros das camadas que aplique transformações nos dados que permitam a rede a inferir as relações entre as entradas e saídas, generalizando para todas os valores do domínio do problema (HAYKIN, 2003). Entretanto, os algoritmos de DL podem conter dezenas de milhões desses parâmetros e a alteração de um deles afeta o comportamento de todos os outros (CHOLLET, 2018). Percebe-se que definir os valores para todos eles demanda bastante processamento.

O primeiro passo na determinação dos parâmetros das camadas é calcular a diferença entre a saída encontrada pela RNA e a saída esperada através da função do erro (*loss function*) (BRAGA *et al.*, 2007). Essa diferença é chamada por Chollet (2018) de pontuação de perda (*loss score*).

Essa pontuação é utilizada para realizar um pequeno ajuste nos valores dos parâmetros com o intuito de reduzir o erro. Esse ajuste é realizado pelo otimizador, que é um subprograma que implementa uma lógica chamada de retropropagação, um dos algoritmos mais importantes da DL (KOVÁCS, 2002) que será discutido mais adiante.

Figura 5 – Fluxograma da DL com Retropropagação



Fonte: Adaptado de Chollet, 2018.

Chollet (2018) utiliza a Figura 5 acima para ilustrar todo este processo. Em um primeiro momento, os parâmetros das camadas possuem valores randômicos, fazendo com que a RNA realize apenas transformações aleatórias nos dados. Neste caso, a pontuação de erro que é calculada pela função do erro tem um valor alto, pois as saídas fornecidas pela RNA são bastante diferentes das saídas esperadas. Este resultado é intuitivo, pois não é esperado que uma série de transformações aleatórias seja capaz de modelar um problema. Durante o processo de treinamento, os pesos são ajustados pouco a pouco na direção correta e a pontuação da perda diminui gradativamente. Este processo de treinamento, que consiste na apresentação de dezenas a milhares de dados de exemplos, costuma ser repetido por dezenas de vezes e cada uma dessas repetições é chamada de época (ABIODUN *et al.*, 2018). A rede finaliza seu treinamento quando alcançam um erro mínimo e as saídas são tão próximas quanto possível dos valores esperados.

2.6.2 Processo de Minimização da Função do Erro

Com base na discursão anterior, percebe-se que a pontuação de perda, que é calculada pela função do erro, é uma função dos parâmetros das camadas ($f(w_i,\ \theta_i) = erro$) e que o processo de treinamento realiza pequenas mudanças nestes parâmetros buscando minimizar esta pontuação.

Utilizando conceitos básicos de cálculo diferencial, temos que quando se deseja minimizar uma função $f(x)$ alterando os valores de x por um fator *épsilon* suficientemente pequeno, devemos primeiramente obter a derivada da função f. A derivada descreve como $f(x)$ se comporta conforme o valor de x muda e, para minimizar o valor de $f(x)$, basta mover o valor de x na direção oposta de sua derivada (STEWART, 2009a).

Também do cálculo, sabe-se que se a derivada em torno de um ponto p qualquer do domínio da função é positiva, um pequena aumento de x resultará em um aumento de $f(x)$ e se a derivada em torno deste ponto é negativa, um pequeno aumento de x resultará em uma diminuição de $f(x)$ (FLEMMING; GONÇALVES, 2006).

Stewart (2009b) define gradiente como uma generalização do conceito de derivada para funções com múltiplas variáveis, como é o caso da função do erro das RNAs de camadas múltiplas. O símbolo deste operador é o *nabla* ∇.

Portanto, tendo em mente todos esses conceitos do cálculo diferencial, a questão fundamental do treinamento das RNAs se resume a calcular o gradiente da função do erro em relação aos parâmetros das camadas e alterar os valores destes parâmetros na direção oposta desse gradiente para que o erro seja minimizado.

Há diferentes formas de atualizar os parâmetros da rede através do cálculo do gradiente da função do erro, ou seja $\nabla f(w_i,\ \theta_i)$. Chollet (2018) lista os seguintes tipos:

1) Método do Gradiente Descendente (MGD) Verdadeiro;
2) Método do Gradiente Desentende (MGD) por Lotes;
3) Método do Gradiente Descendente (MGD) por Minilotes Estocásticos.

A diferença entre eles está na quantidade de dados que são considerados para ajustar os parâmetros da rede (SILVA, IVAN *et al.*, 2016). No MGD Verdadeiro, os ajustes nos parâmetros da rede são efetivados após a apresentação de cada amostra do conjunto de treinamento, sendo um método que não necessita de tantos recursos computacionais como os outros, porém é o menos preciso. Já no MGD por Lotes, os ajustes nos parâmetros da rede são efetivados após a apresentação de todo o do conjunto de treinamento, sendo o método mais dispendioso computacionalmente, porém é o mais preciso. Por fim, no MGD por minilotes estocásticos, os ajustes nos parâmetros da rede são efetivados após a apresentação de um subconjunto do conjunto de treinamento de tamanho definido pelo programador.

Segundo Abiodun *et al.* (2018), o MGD por minilotes estocásticos é o mais utilizado, pois consiste em um meio termo entre os três métodos e equilibra bem a necessidade de recursos computacionais e a acurácia dos resultados.

Portanto, devido a essas características, esse foi o método escolhido para ajustar os parâmetros da rede no estudo de caso realizado neste trabalho e apresentaremos suas características a seguir.

2.6.3 O Método do Gradiente Descendente de Minilotes Estocástico

Stewart (2009a) mostra que o mínimo de uma função é um ponto em que o valor de sua derivada é 0. Portanto, para encontrar seu mínimo global, basta determinar todos os pontos onde a derivada é 0 e verificar em qual deles a função assume o menor valor. Isso significa resolver a equação $\nabla f(w_i,\ \theta_i) = 0$ para todos os parâmetros da rede e encontrar o conjunto de pesos w_i e limiares θ_i que levam a menor pontuação de perda possível.

Entretanto, já sabemos que uma RNA possui milhares, as vezes milhões, de parâmetros. Logo, resolver $\nabla f(w_i,\ \theta_i) = 0$ é um processo computacionalmente dispendioso e não costuma ser feito na prática (SILVA, IVAN; SPATTI, DANILO; FLAUZINO, 2016).

Em vez disso, Chollet (2018) utiliza um método chamado de Método do Gradiente Descendente (MGD) de minilotes estocásticos (aleatórios).

Esse método se inicia com a divisão aleatória do conjunto de treino em partes menores chamadas de lotes. Em seguida, roda-se a rede para um lote e se calcula o erro da camada de saída. Se o ajuste estiver sendo feito na camada de saída, utiliza-se o erro para ajustar os parâmetros dessa camada. Caso o ajuste estiver sendo feito em alguma das camadas ocultas, calcula-se uma estimativa do erro daquela camada com base no erro da saída para realizar o ajuste. Independente do caso, o erro ou sua estimativa são representados pela variável ER. Mais detalhes sobre essa etapa do MGD serão dados na próxima subseção.

Finalmente, calcula-se o valor do gradiente com base no estado atual da RNA e, considerando a direção oposta do gradiente, utiliza-se seu valor na modificação dos parâmetros da rede, conforme as Equações (2) e (3):

$$w_i^{atual} = w_i^{anterior} - n_{apr} * \nabla f(w_i,\ \theta_i) * ER \tag{2}$$

$$\theta_i^{atual} = \theta_i^{anterior} - n_{apr} * \nabla f(w_i,\ \theta_i) * ER \tag{3}$$

A variável n_{apr} é chamada de taxa de aprendizagem e determina a velocidade do processo de treinamento da rede (HAYKIN, 2003). Seu valor pode ser constante ou variar no processo de treinamento, iniciando com um valor máximo e diminuindo gradativamente

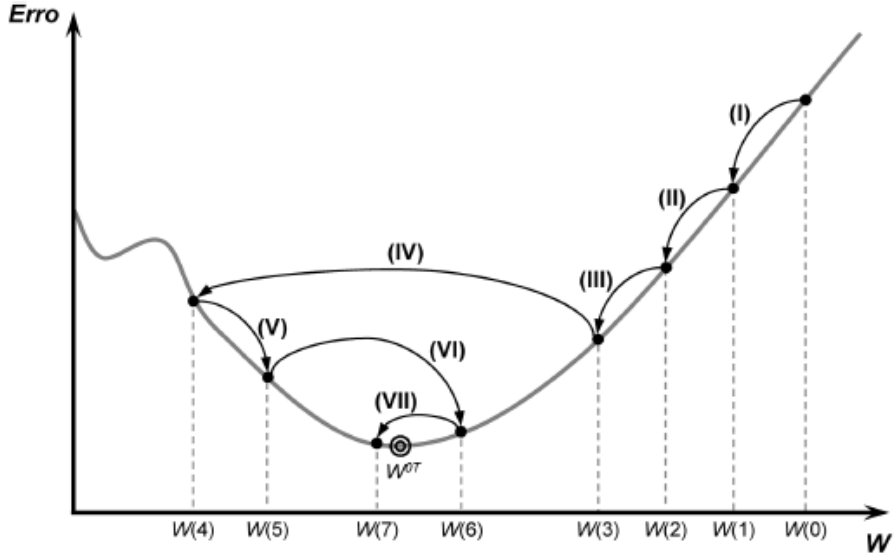
(KOVÁCS, 2002). Segundo Silva *et al.* (2016), seu valor deve estar compreendido entre $0 < n < 1$ para evitar instabilidades no treinamento.

Este processo é repetido para todas as camadas da RNA e utiliza todos os outros lotes de dados, de tal sorte que o erro diminui gradativamente. Quando todos os lotes de um conjunto de dados de treinamento são utilizados, é dito que se passou uma época. Então, os dados são divididos aleatoriamente em novos lotes e o treinamento continua enquanto o erro for maior que um determinado valor (KOVÁCS, 2002). O tamanho dos lotes e os critérios de parada – o nível de erro tolerável e o número máximo de épocas, por exemplo – são determinados pelo programador (BRAGA *et al.*, 2007).

O Gráfico 2 abaixo ilustra a evolução do erro em um típico processo de treinamento. Tomamos como exemplo, mas sem perda de generalização, uma função do erro qualquer definida pela curva cinza. A RNA é inicializada com um conjunto de parâmetros de rede aleatórios em $W(0)$, possuindo um erro elevado. Ao passo que o treinamento se sucede e o MGD é utilizado, percebemos que o erro tende a reduzir com o passar das épocas, convergindo para um valor próximo do seu mínimo global em W^{OT}.

Além disso, analisando o Gráfico 2 abaixo, percebemos que adotando um valor pequeno demais para a taxa de aprendizagem n_{apr}, serão necessários muitas épocas para que o erro convirja para o mínimo global, correndo o risco do processo ficar preso em um mínimo local. Já adotando uma taxa muito grande, os saltos entre as épocas serão exagerados, correndo o risco de o sistema vagar pela curva do erro aleatoriamente e parar em um local longe do mínimo global.

Gráfico 2 – Evolução do Erro Durante o Processo de Treinamento



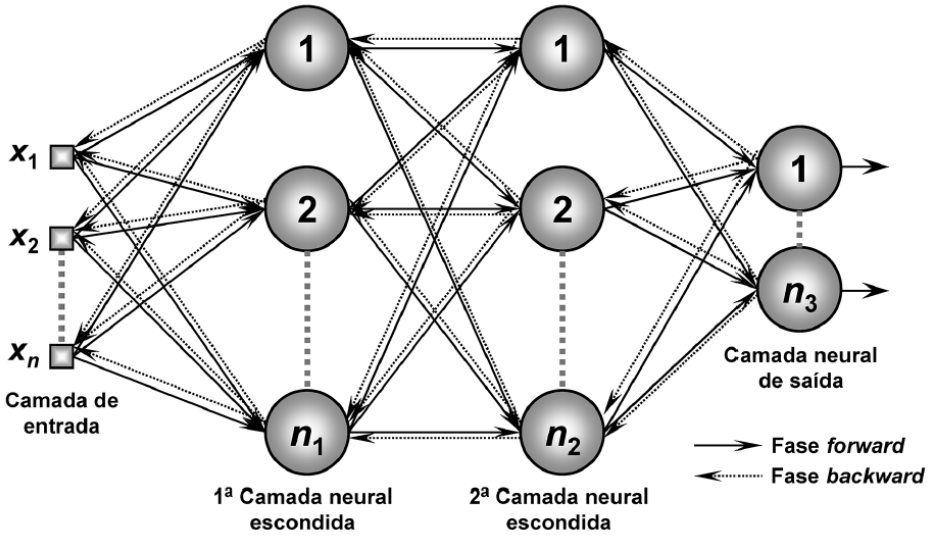
Fonte: Silva *et al.,* 2016.

Vimos anteriormente que a camada de saída é a responsável pelo cálculo do erro, gerando um sinal de *feedback* que é utilizado como direção no ajuste dos parâmetros da rede. Vamos abordar na próxima subseção como esse sinal se retropropaga entre as camadas, partindo da mais externa (a de saída) e chegando até a camada mais profunda (a de entrada). Esse processo é possível graças ao algoritmo da retropropagação.

2.6.4 O Princípio da Retropropagação

O treinamento das RNAs que utilizam do princípio da retropropagação é dividido em dois grandes momentos: a fase *forward* e a fase *backward* (HAYKIN, 2003). A Figura 6 abaixo apresenta o fluxo dessas fases, considerando uma RNA genérica com uma camada de n entradas, 2 camadas neurais escondidas e uma camada neural de saída com, respectivamente, n_1, n_2 e n_3 NAs.

Figura 6 – Fases *Foward* e *Backward* do Algoritmo de *Retropropagação*



Fonte: Silva *et al.,* 2016.

A primeira etapa do treinamento da RNA é chamada de fase *forward*. Ela se inicia com a passagem das amostras do conjunto de treinamento para as entradas da rede. Essas entradas são transformadas e propagadas conforme já descrito anteriormente, camada a camada, até chegar no final da rede, onde as saídas são produzidas. Esta etapa tem como objetivo simplesmente obter as respostas da RNA, considerando os valores atuais dos parâmetros da rede, os quais permanecerão inalterados nesta primeira etapa (SILVA, IVAN; SPATTI, DANILO; FLAUZINO, 2016).

Após a obtenção das respostas da RNA, é realizada uma comparação entre essas saídas geradas na última camada e as respostas desejadas utilizando a função do erro, conforme já visto anteriormente. Então, os desvios calculados são utilizados para alterar os parâmetros da camada neural de saída utilizando as Equações (2) e (3).

Segundo Haykin (2003), para as camadas escondidas, não há valores de saídas desejadas conhecidas de maneira direta, diferentemente da camada de saída onde estes valores são bem definidos e conhecidos. Então, o ajuste dos parâmetros das camadas escondidas é feito utilizando estimativas dos erros de suas saídas, calculadas na segunda etapa da retropropagação. Para uma dada camada oculta, suas estimativas são calculadas, segundo Silva (2016), com base nos parâmetros da camada imediatamente mais externa a ela que, logicamente, já foi ajustada, pois somente após o ajuste de uma camada mais externa é que se inicia o ajuste de uma camadas mais profunda.

A segunda etapa do treinamento da RNA é chamada de *backward*. Nessa fase, os erros calculados na camada de saída são retropropagados para a camada escondida imediatamente mais profunda a ela para que seja calculado uma estimativa do erro dessa camada que, por sua vez, é utilizada para atualizar seus parâmetros, conforme as Equações (2) e (3). Logo após esse ajuste, o sinal de *feedback* continua sua retropropagação, sendo enviado para uma camada ainda mais profunda a aquela que acabou de ser ajustada, dando prosseguimento aos ajustes de maneira análoga ao que já foi exposto. O processo de *backward* continua até que todas as camadas tenham realizado o ajuste dos parâmetros (KOVÁCS, 2002).

Com essa explicação, é possível identificar a essência primordial da retropropagação. No início, tem-se os parâmetros da camada de saída ajustados com base nos desvios calculados entre as respostas produzidas pela RNA e suas respectivas saídas esperadas. Em um segundo momento, esse erro é retropropagado para as camadas mais profundas, sendo utilizado para estimar os erros de suas saídas. Portanto, a saída desejada de uma camada escondida é determinada como uma função da camada imediatamente mais externa.

Portanto, podemos resumir que o treinamento de uma RNA que utiliza o algoritmo de retropropagação consiste no ajuste gradual e automático de seus parâmetros de rede através da aplicação sucessiva das fases *forward* e *backward.* Estes processos são repetidos por algumas épocas e, utilizando um conjunto de dados de treinamento e aplicando o método do gradiente descendente, levam a minimização do erro. No final de tudo isso, é esperado um sistema inteligente que generaliza as relações entre as entradas e saídas de um problema.

2.7 Arquitetura *Deep Belief Network*

O Método do Gradiente Descendente (MGD) possui uma grande limitação. Segundo Hua *et al.* (2015), no processo de treinamento de uma RNA que utiliza o MGD, quanto mais distante uma camada estiver da saída, menor será o ajuste de seus parâmetros, pois o gradiente do erro que serve como sinal de *feedback* se dissipa em cada retropropagação. Isso leva a problemas como o *overfitting* – a rede não é capaz de generalizar soluções como deveria, mas apenas se ajusta exageradamente aos dados apresentados – e o aprisionamento em mínimos locais – o processo de treinamento fica preso em um mínimo local longe da solução ótima.

Buscando resolver a dissipação do sinal de *feedback* e suas efeitos, Hinton *et al.* (2006) propôs uma nova arquitetura de *Deep Learning* denominada de *Deep Belief Network* (DBN).

Esta arquitetura se mostrou bastante eficiente na modelagem de uma RNA, sendo capaz de atuar como um controlador de velocidade de motores. Por exemplo, Cheon *et al.*, (2015) modelou uma RNA com arquitetura DBN treinada com entradas e saídas de um controlador PID capaz de se comportar como um controlador neural. Como o escopo de nosso trabalho aborda essa mesma problemática, apresentaremos, nesta subseção, os principais conceitos da DBN e adiantamos que esta será a arquitetura utilizada na RNA modelada no estudo de caso.

Veremos que uma DBN consiste em Máquinas de Boltzmann Restritas (MBR) posicionadas em série. Portanto, iniciaremos com o seu estudo.

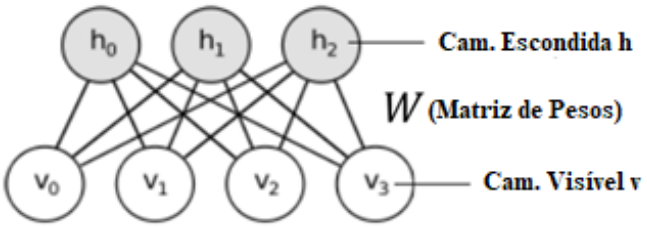
2.7.1 Máquina de Boltzmann Restrita

A MBR consiste em um método estocástico e generativo proposto por Smolensky que é utilizado para determinar a distribuição de probabilidades das entradas que são passadas a ela (SMOLENSKY, 1986). No contexto das RNAs, isso significa que a MBR determina padrões nos dados de maneira automática através da reconstrução das entradas, processo realizado durante o treinamento (AREL; ROSE; KARNOWSKI, 2010).

Na prática, a MBR consiste em uma RNA de 2 camadas, onde a primeira camada é a entrada da rede e a segunda consiste em uma camada neural escondida. Juntas, elas conseguem detectar características relevantes (*features)* nos dados apresentados que são armazenadas em seus parâmetros de rede W (HUA; GUO; ZHAO, 2015).

A camada de entrada é denominada de camada visível v e a segunda é chamada de camada escondida h. O termo restrito vem do fato dos NAs de uma mesma camada não terem ligações entre si. A Figura 7 abaixo ilustra uma MBR com 4 entradas na camada visível (v_0, v_1, v_2 e v_3) e 3 neurônios na camada escondida (h_0, h_1 e h_2).

Figura 7 – Máquina de Boltzmann Restrita (MBR)



Fonte: Adaptado de Hua *et al.,* 2015.

Segundo Arel *et al.* (2010), o treinamento da MBR é do tipo não-supervisionado e ocorre em duas etapas denominadas de *forward* e *backward*. No início do processo, os parâmetros da MBR, ilustrados na Figura 7 por W, são inicializados aleatoriamente com valores entre 0 e 1.

Na fase *forward*, os dados são apresentados para a camada visível para que sejam transformados com a utilização de W, buscando uma representação mais significativa das entradas para a rede neural. O resultado dessa transformação é enviado para a camada escondida e essa camada, por sua vez, realiza outra transformação nos dados com o intuito de reconstruir os dados de entrada, buscando desfazer a transformação realizada pela camada visível e reproduzir a entrada original em sua saída (AREL; ROSE; KARNOWSKI, 2010).

Em seguida, na fase *backward*, os dados transformados pela camada escondida são reenviados a camada visível que compara a tentativa de reconstituição da entrada feita pela camada escondida com a entrada original. A diferença entre a entrada reconstituída e a original é calculada através de um método chamado de divergência de Kullback-Leibler e seu valor é utilizado tanto para alterar os parâmetros da rede como para mensurar sua acurácia (HINTON; OSINDERO; TEH, 2006).

O treinamento modifica continuamente os parâmetros da rede até que a reconstituição da entrada esteja tão próxima da entrada original quanto for possível. Ao final do treinamento, a MBR terá capturado características relevantes que foram observadas nos dados amostrais, armazenando-as nos parâmetros da rede. Graças a essas sucessivas transformações realizadas por suas camadas, as MBRs acabam por decifrar as interrelações

entre os dados de entrada, encontrando padrões inerentes desses dados. Essa capacidade explica porque eles são frequentemente utilizados como extratores de características relevantes, também chamadas de *features extractors* (LIU *et al.*, 2017).

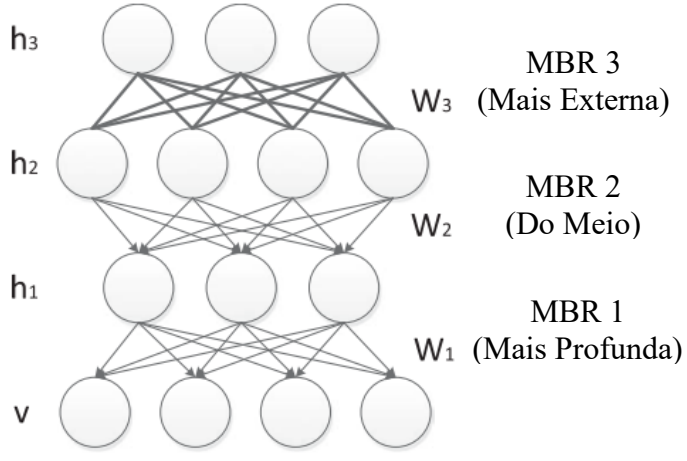
Veremos agora como as MBRs são combinadas e dão origem as *Deep Belief Networks* (DBNs).

2.7.2 Construção das DBNs

Hinton *et al.* (2006) propôs uma arquitetura de RNA que consiste em diversas MBRs em série, onde a saída da camada escondida de uma MBR mais profunda é a entrada da camada visível de uma MBR mais externa. Esta arquitetura possibilita que a RNA extraia características relevantes gradualmente mais complexas ao passo que as camadas se sucedem. Isso é feito através da combinação de características mais simples identificadas em camadas mais profundas, tornando a rede mais apta a entender e resolver o problema modelado do que outras arquiteturas. Hinton *et al.* (2006) deu a este arranjo de MBRs em série o nome de *Deep Belief Network* (DBN).

A Figura 8 abaixo ilustra uma DBN composta por 3 MBRs em série. Vemos que essa RNA possui 4 camadas, ou seja, uma camada visível de entrada e 3 camadas neurais escondidas. Além disso, como definido por Hilton, vemos que a camada escondida h_1 da MBR 1 (mais profunda) funciona como uma camada visível para a MBR 2 (do meio). Por sua vez, a camada escondida h_2 da MBR 2 (do meio) funciona como uma camada visível para a MBR 3 (mais externa). Com essa concatenação, temos uma DBN com uma camada visível v e uma camada escondida h_3.

Figura 8 – *Deep Belief Network* (DBN)



Fonte: Adaptado de Liu *et al.*, 2017.

Resta analisarmos como essas DBNs são treinadas. Abordaremos este tópico logo a seguir.

2.7.3 Treinamento das DBNs

As diferentes MBR de uma DBN são treinadas sequencialmente da mais profunda a mais externa. Assim, no nosso exemplo, a MBR mais profunda, representada pelas camadas v e h_1 na Figura 8, é treinada primeiro com a execução das etapas *forward* e *backward* descritas na subseção 2.5.1. Em seguida, as MBRs mais externas são sucessivamente treinadas, uma a uma, de maneira análoga (LIU *et al.*, 2017).

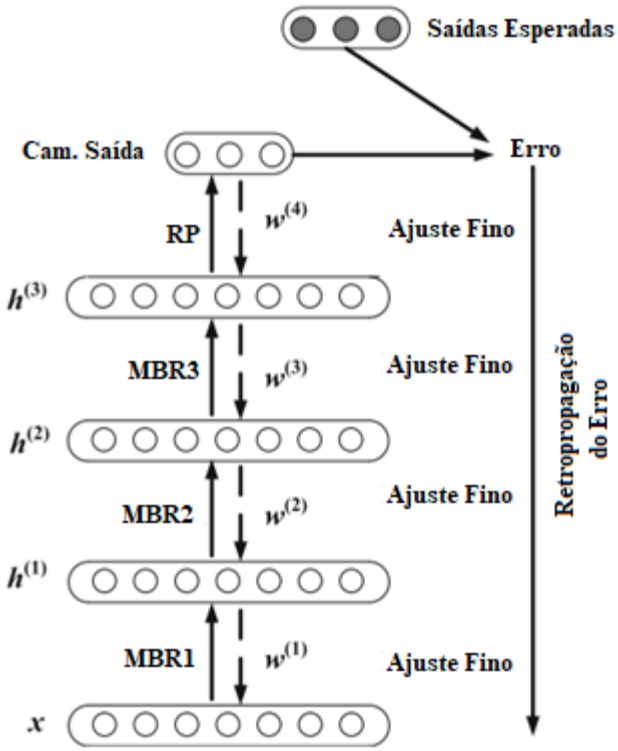
A DBN possui uma capacidade de aprendizado maior em comparação com o modelo que utiliza somente uma MBR, pois ela consegue extrair características relevantes mais complexas através da combinação de características mais simples identificadas em camadas mais profundas (ARNOLD *et al.*, 2010).

Já sabemos que o processo de treinamento de uma MBR é do tipo não-supervisionado. Logicamente, a arquitetura DBN também possui um treinamento não-supervisionado. Por isso, costuma-se adicionar uma camada de NAs após a saída da DBN, logo após todas as camadas de MBRs, para que seja possível realizar um treinamento supervisionado (HUA; GUO; ZHAO, 2015).

Com isso, é possível utilizar a retropropagação, já analisada anteriormente, para alimentar as MBRs das camadas mais profundas com as características relevantes extraídas das MBRs das camadas mais externas, melhorando os resultados obtidos pela DBN em comparação a sua versão sem essa camada adicional (DENG, 2012). Isso ocorre porque, com a adição da retropropagação, o ajuste dos parâmetros da RNA leva em conta todas as características relevantes encontradas por todas as camadas. Esta arquitetura está ilustrada na Figura 9 abaixo e é denominada de *Deep Belief Network* com retropropagação.

O treinamento da DBN com retropropagação possui 2 etapas: o pré-treino e o ajuste fino, também chamado de *fine-turning* (LIU *et al.*, 2017). O fluxo destas duas etapas pode ser visto na Figura 9 abaixo.

Figura 9 – *Deep Belief Network* com Retropropagação



Fonte: Adaptado de Zeng *et al.*, 2016.

A primeira etapa é o pré-treino que consiste na realização do treinamento de todas as MBRs de maneira sequencial, da mais profunda a mais externa, conforme já explicado anteriormente. Nessa etapa, ocorre um treinamento não-supervisionado em cada MBR para que elas extraiam características relevantes dos dados. Graças a esta etapa, os parâmetros da rede são levados para uma configuração próxima ao ótimo global, melhorando consideravelmente o desempenho da DBN (HINTON; OSINDERO; TEH, 2006).

Em seguida, inicia-se o ajuste fino que consiste na execução de um algoritmo de retropropagação. Nessa etapa, conforme já visto anteriormente, é realizado um treinamento supervisionado, onde um sinal de *feedback* é calculado utilizando o MGD e retropropagado para todas as outras camadas, no sentido da camada mais externa para a mais profunda. Cada camada utiliza esse sinal para ajustar seus parâmetros.

O pré-treino e o ajuste fino são repetidos até o erro da rede convergir para um valor mínimo, o que leva a conclusão do treinamento.

Em resumo, o pré-treino leva a RNA a um estado em que ela detecta, de maneira satisfatória, padrões entre as entradas e saídas dos dados, armazenando as regras que modelam o problema em seus parâmetros de rede, posicionando-os em uma situação próxima do ponto onde o erro é mínimo. Em seguida, no ajuste fino, ocorrem pequenas alterações nestes

parâmetros, levando a rede a uma configuração ainda mais próxima do ótimo global, onde o erro é o mínimo possível. Essas etapas e suas subetapas fazem que a DBN com retropropagação seja um detector de características relevantes bastante eficiente (CHEON *et al.*, 2015).

Finalizamos esta subseção reiterando que a DBN com retropropagação é a arquitetura adotada para a rede neural modelada em nosso estudo de caso, sendo utilizada com o propósito de controlar a velocidade de um motor.

Os assuntos apresentados até então abordaram sobre IA e suas subdivisões. Em relação aos conceitos básicos necessários para o entendimento do estudo de caso, resta abordarmos sobre os motores de corrente contínua e sobre os controladores PID, objetos de estudo das próximas subseções.

2.8 Motores Elétricos de Corrente Contínua

Nessa subseção, vamos discutir os principais conceitos sobre os motores de corrente contínua. Vamos iniciar esse estudo apresentando suas principais aplicações, vantagens e tipos. Em seguida, iremos analisar seus aspectos construtivos, seu princípio de funcionamento e seu circuito equivalente para, enfim, chegarmos no ponto que nos interessa que é o controle de velocidade.

2.8.1 Conceitos Básicos

Devido suas características, os motores de Corrente Contínua (CC) são utilizados em inúmeras aplicações, estando presentes, por exemplo, em máquinas de papel, bobinadeiras, laminadores, máquinas de impressão, extrusoras, prensas, elevadores, esteiras e moinhos de rolo (HONDA, 2006).

Os motores CC possuem diversas vantagens quanto a forma de operação, o nível de confiabilidade, a facilidade de utilização e a dinâmica de controle (HONDA, 2006). Dentre essas vantagens, destacamos:

1) Operação em 4 quadrantes com custo relativamente mais baixo;
2) Ciclo contínuo mesmo em baixas rotações;
3) Alto torque na partida e em baixas rotações;
4) Ampla variação de velocidade;
5) Facilidade em controlar a velocidade;

6) Alto nível de confiabilidade;
7) Alta flexibilidade.

Dentre essas características, Toro (1999) destaca a facilidade de se controlar a velocidade de um motor CC em relação a sua contraparte de corrente alternada. Essa vantagem foi um dos principais motivos que nos levou a escolher o motor CC para ser controlado pela RNA desenvolvida no estudo de caso.

Segundo Chapman (2013), podemos classificar os motores CC de uso geral em cinto tipos, a depender do tipo de excitação adotado. São eles:

1) O motor CC de excitação independente;
2) O motor CC em derivação;
3) O motor CC de ímã permanente;
4) O motor CC série;
5) O motor CC composto.

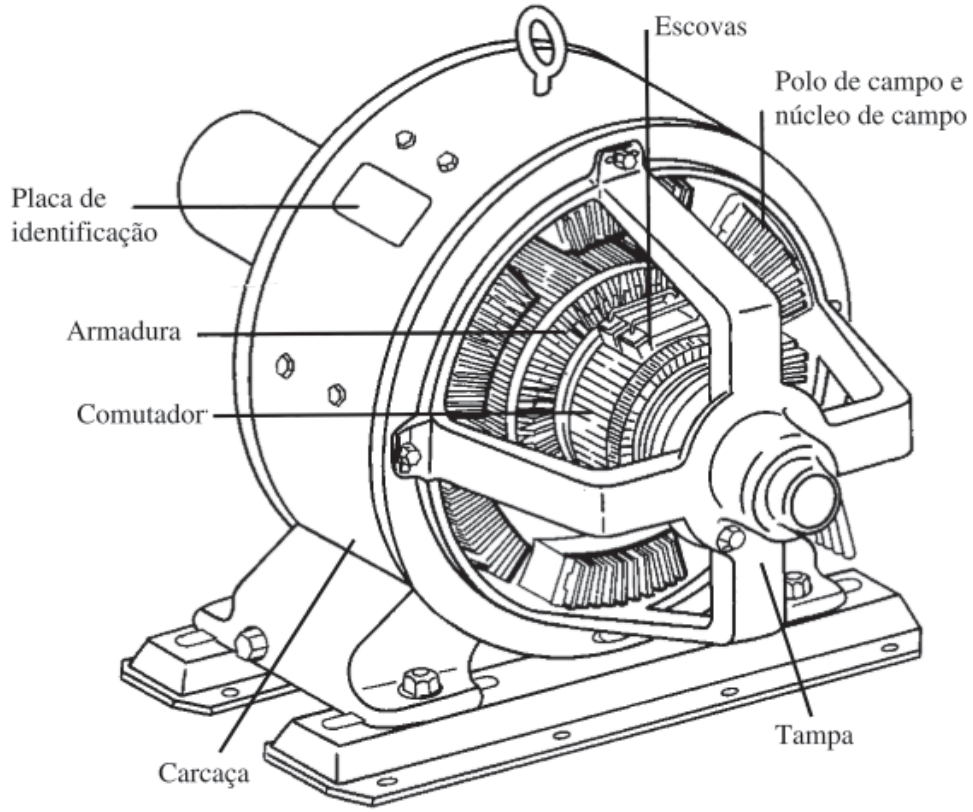
No motor CC de excitação independente, o controle de velocidade é feito primariamente através da variação da tensão aplicada à armadura do motor, mantendo a tensão do circuito de campo constante (CHAPMAN, 2013). Esse método de excitação é o mais desejável do ponto de vista de flexibilidade e possui alto rendimento operacional (TORO, 1999). Nesse método, a velocidade também pode ser controlada pelo variação do campo.

Conforme veremos no capítulo referente ao estudo de caso, a RNA modelada é capaz de controlar a velocidade de um motor CC através da variação da tensão nos terminais de armadura desse motor. Portanto, levando em consideração essa definição do projeto, percebemos que o motor CC de excitação independente é adequado para a resolução do nosso problema, sendo o tipo de motor CC adotado em nosso experimento. Logo, devemos entender melhor o funcionamento desse tipo de motor CC, o que será feito nas próximas subseções.

2.8.2 Aspectos Construtivos

Segundo Fitzgerald *et al.* (2014), o motor CC é uma máquina elétrica constituída por duas estruturas magnéticas chamadas de estator e rotor. O estator também é chamado de enrolamento de campo e o rotor de enrolamento de armadura. A Figura 10 consiste em um diagrama esquemático de um motor CC.

Figura 10 – Estrutura Física de um Motor CC



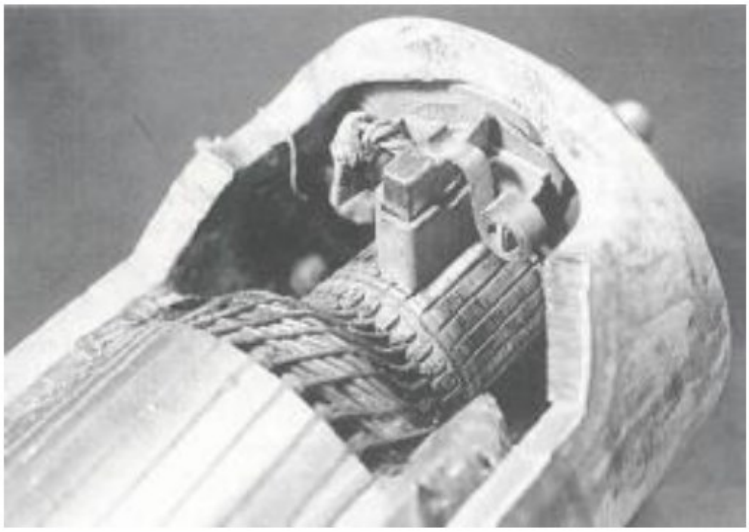
Fonte: Chapman, 2013.

O estator é constituído por uma estrutura ferromagnética de pólos salientes nos quais são enroladas bobinas que, na passagem de corrente elétrica, foram um campo magnético.

Já o rotor consiste em um eletroímã girante de núcleo ferromagnético cilíndrico com enrolamentos na sua superfície. Esses enrolamentos são alimentados por um sistema mecânico de comutação formado por um comutador e por escovas, conforme ilustrado pela Figura 11. O comutador é solidário ao eixo do rotor e possui uma superfície cilíndrica com diversas lâminas conectados aos enrolamentos do rotor. As escovas são fixas e estão ligadas aos terminais de alimentação. Essas escovas exercem pressão sobre o comutador, estabelecendo uma ligação elétrica entre a parte fixa (terminal de armadura) e a parte móvel do motor (as bobinas do rotor).

Conforme veremos mais adiante, o sistema de comutação serve para inverter a corrente nas bobinas do rotor de forma que o conjugado desenvolvido sempre esteja na mesma direção.

Figura 11 – Sistema de Comutação



Fonte: Honda, 2006.

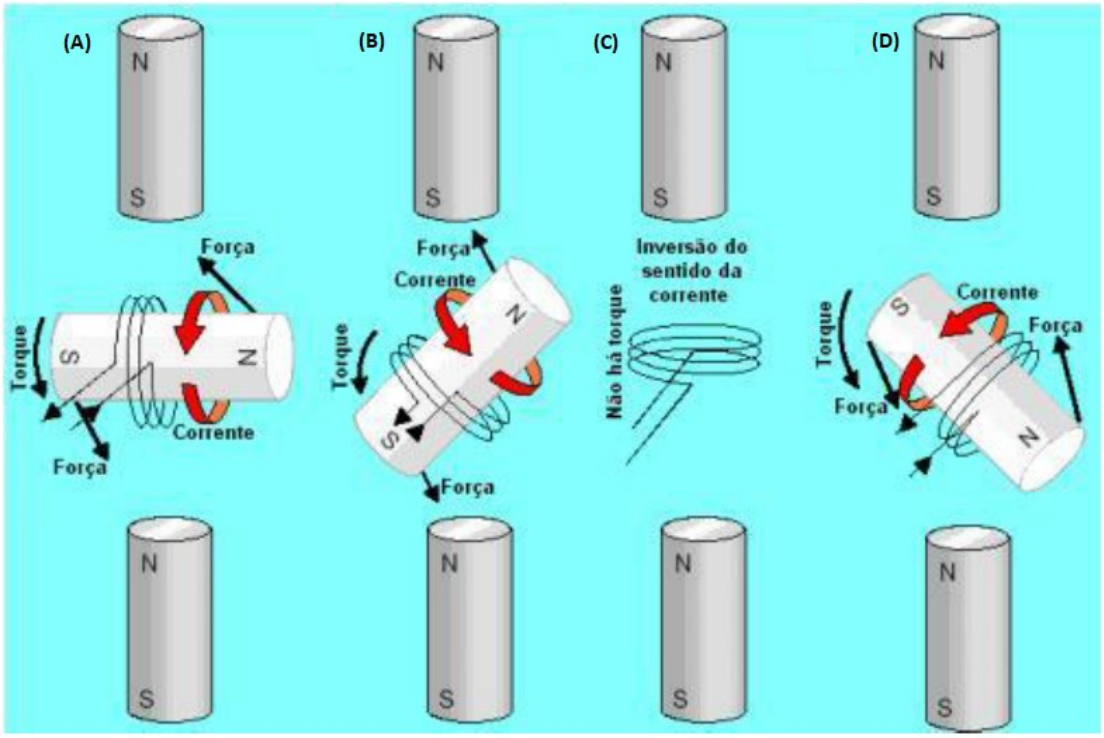
2.8.3 Princípios de Funcionamento

Pela teoria do eletromagnetismo, quando uma corrente elétrica passa por um condutor, é produzido um campo magnético (FITZGERALD; KINGSLEY; UMANS, 2014). Portanto, podemos imaginar que as bobina do estator e do rotor, quando energizadas, se comportam como ímãs permanente. A Figura 12 abaixo ilustra essa representação para um motor CC que possui apenas uma bobina no rotor e duas boninas no estator.

Fitzgerald *et al.* (2014) explica que os condutores do rotor são enrolados em um núcleo ferromagnético cilíndrico, formando bobinas de n espiras conectadas em série pelas lâminas do comutador. Os dois lados de cada enrolamento são inseridos em sulcos no núcleo ferromagnético do rotor espaçados igualmente e com a mesma distância entre os pólos do estator. Desse modo, analisando a Figura 12, quando os condutores do rotor referentes ao seu polo sul estão sob a influência do polo norte do estator, os condutores do rotor referentes ao seu polo norte estão sob a influência do polo sul do estator.

Honda (2006) utiliza a Figura 12 abaixo para explicar o funcionamento do motor CC. Imaginemos que o motor esteja na situação ilustrada em (A) onde a bobina do rotor está na horizontal. Como os pólos opostos dos ímãs se atraem, surge um torque na bobina do rotor que gera uma aceleração angular. Assim, o rotor passa a girar no sentido anti-horário, como vemos em (B).

Figura 12 – Funcionamento do Motor CC

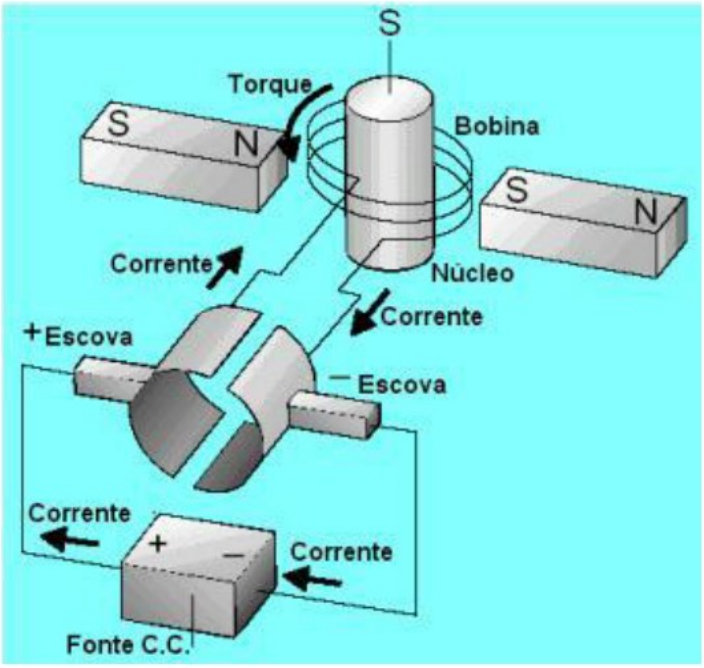


Fonte: Honda, 2006.

Esse torque existe até o momento em que os pólos da bobina do rotor alcancem os pólos opostos das bobinas do estator. Nesse momento, ilustrado em (C), a bobina do rotor girou 90º e não há mais torque atuando. Num momento imediatamente posterior a esse instante de torque nulo ocorre uma inversão no sentido da corrente na bobina do rotor. Logo, inverte-se os pólos do campo magnético do rotor, surgindo uma força de repulsão muito grande. Devido à inércia do rotor e ao fato de o rotor apresentar um momento angular anti-horário, ela continua girando nesse sentido e o torque gerado pelas forças de repulsão gera uma aceleração angular que mantém o rotor girando no sentido anti-horário, como mostrado em (D). A rotação continua e, no próximo instante de torque nulo, o ciclo se repete de maneira semelhante.

Essas atrações e repulsões coordenadas e originadas da inversão do sentido da corrente nas bobinas do rotor garantem o surgimento de torques no sentido favorável a continuidade do giro do motor. Esse processo é realizado pelo sistema de comutação (CHAPMAN, 2013). A Figura 13 abaixo representa um esquema simplificado de um motor CC e seu sistema de comutação formado pelo comutador e as escovas.

Figura 13 – Esquema Simplificado de um Motor CC



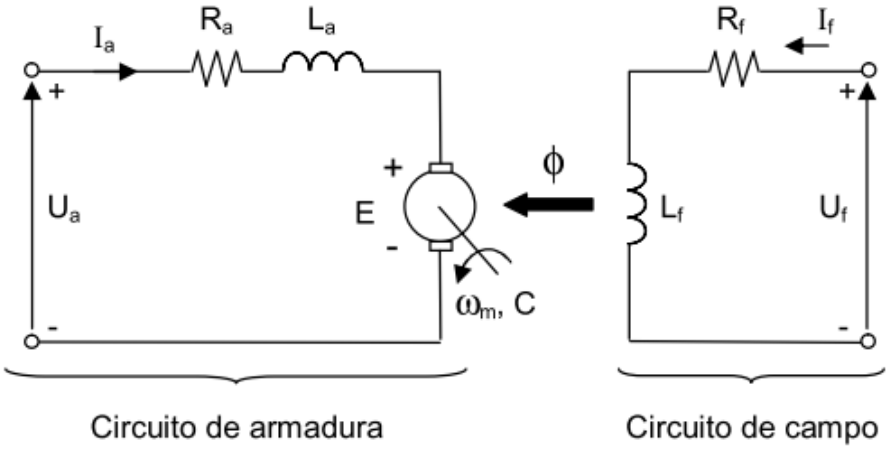
Fonte: Honda, 2006.

Analisando a Figura 13, percebemos que esse comutador simplificado é constituído de duas placas de cobre encurvadas e fixadas no eixo do rotor (HONDA, 2006). Tem-se que os terminais do enrolamento da bobina do rotor estão ligados a essas placas. O polo positivo da excitatriz do rotor se conecta através da escova (+) a uma das placas do comutador e o polo negativo da excitatriz se conecta através da escova (−) a outra placa do comutador. Essa ligação gera uma corrente que flui pela bobina do rotor no sentido indicado. Essa situação permanece enquanto o rotor realiza sua primeira meia-volta. Logo após, as placas do comutador trocam seus contatos entre as escovas, ou seja, a placa antes ligada a escova (+) agora está ligada a escova (−) e vice-versa. Essa mudança faz com que a corrente na bobina do rotor inverta seu sentido, mantendo o motor CC girando sempre com o mesmo sentido de rotação, conforme o princípio de funcionamento analisado anteriormente.

2.8.4 Circuito Equivalente do Motor CC de Excitação Independente

A Figura 14 abaixo mostra o circuito equivalente de um motor CC de excitação independente.

Figura 14 – Circuito Equivalente do Motor CC



Fonte: Honda, 2006.

O circuito de armadura consiste em uma fonte de tensão ideal E que representa a força eletromotriz induzida, também chamada de força contraeletromotriz da armadura (TORO, 1999), uma resistência R_a e uma indutância L_a. Essas grandezas, em conjunto, modelam a estrutura completa do rotor, incluindo as bobinas do rotor, os interpolos, as resistências de contato das escovas e os enrolamentos de compensação, caso existam (CHAPMAN, 2013). Já as bobinas de campo, que são as responsáveis por gerar o fluxo magnético do motor, são representadas pelo indutor L_f e pelo resistor R_f. Por fim, U_a e U_f são, respectivamente, as tensões aplicadas na armadura e no campo, I_a e I_f são suas correntes correspondentes, Φ é o fluxo magnético estabelecido no entreferro e ω_m é a velocidade angular do rotor.

Segundo Alexander e Sadiku (2013), utilizando a Lei de Kirchhoff das tensões no circuito de armadura, temos a Equação (4):

$$U_a = R_a * I_a + E \tag{4}$$

Já pela Lei da Indução de Faraday, podemos estabelecer a Equação (5) que diz que a força eletromotriz induzida é proporcional ao fluxo magnético estabelecido no entreferro Φ, à velocidade de rotação n e por uma constante k_1 que representa as características construtivas do motor tal como o tamanho do rotor, o número de polos e como esses polos são interconectados (FITZGERALD; KINGSLEY; UMANS, 2014).

$$E = k_1 * \Phi * n \tag{5}$$

Com esse circuito equivalente apresentado e equacionado, podemos encontrar uma expressão para a velocidade do rotor em função da tensão na armadura.

2.8.5 Controle de Velocidade

Combinando as Equações (4) e (5), obtemos uma expressão que determina a velocidade do motor CC em (6):

$$n = \frac{U_a - R_a * I_a}{k_1 * \Phi} \tag{6}$$

Fitzgerald *et al.* (2014) afirmam que podemos admitir que a queda de tensão na armadura é pequena, ou seja, que $R_a * I_a \cong 0$. Logo, a Equação (6) se reduz a Equação (7):

$$n = \frac{U_a}{k_1 * \Phi} \tag{7}$$

Portanto, a Equação (7) nos diz que a velocidade de rotação do motor é diretamente proporcional à tensão de armadura U_a e inversamente proporcional ao fluxo no entreferro Φ.

Concluímos, portanto, que o controle da velocidade de um motor CC de excitação independente é feito através da variação da tensão de armadura do motor U_a, mantendo-se o fluxo no entreferro Φ constante, até a velocidade nominal. Velocidades superiores à nominal são alcançadas diminuindo o fluxo no entreferro Φ e mantendo a tensão de armadura constante em seu valor máximo (HONDA, 2006).

Finalizamos a apresentação de todos os conceitos necessários sobre motores CC para o entendimento do estudo de caso. Na próxima subseção, vamos analisar os fundamentos básico da teoria de controladores PID, solução bastante utilizada para controlar a velocidade de motores em geral. Já comentamos anteriormente que as entradas e saídas de um controlador PID foram utilizadas para treinar a RNA que modelamos para a tarefa de controlar a velocidade de um motor CC. Portanto, devemos conhecer o básico de como um controlador PID funciona para termos um entendimento completo do experimento realizado.

2.9 Controlador PID

Iremos discutir nessa subseção os principais conceitos sobre os controladores PID. Vamos iniciar com uma breve apresentação do histórico sobre sua origem, suas vantagens, desvantagens e possibilidades de aplicação. Logo após, continuamos nosso estudo analisando

os parâmetros que definem o comportamento do controlador para, em seguida, estudar os conceitos referentes as análises de suas respostas. Por fim, comentamos sobre os objetivos e métodos de ajuste dos parâmetros de controle.

2.9.1 Contextualização

O controlador Proporcional Integral Derivativo (PID) consiste em um método utilizado para controlar o comportamento de sistemas e processos em geral (OGATA, 2010). Sua fundamentação matemática foi originada em 1910 com a publicação do trabalho de Elmer Sperry que consistiu em um sistema de piloto automático de um navio. Entretanto, somente em 1942 a popularidade do controlador PID passou a crescer devido a publicação do trabalho de Ziegler-Nichols no qual ele criou um método de ajuste direto para os parâmetros do controlador PID (ANG *et al.*, 2006).

Atualmente, a maioria dos algoritmos de controle PID modernos são implementados em microprocessadores digitais. Esses equipamentos processam sinais gerados por sensores e outros dispositivos eletrônicos para conceber uma saída que é utilizada para controlar um sistema ou processo (JOHNSON; MORADI, 2005).

Ang *et al.* (2006) afirmam que a utilização do controle PID é consagrado na indústria e que atualmente mais de 90% dos controladores são desse tipo.

Como exemplo de aplicação do controle PID na indústria, podemos citar o trabalho de Sariyildiz *et al.* (2015) no qual é proposto um método de ajuste dos parâmetros do controle PID com *feedback* de velocidade. O autor aplicou esse método em um sistema de controle de movimento de motores lineares.

Ang *et al.* (2006) afirmam que a popularidade do PID se dá devido suas características, tais como a simplicidade, bom funcionamento, alta gama de aplicação e facilidade de uso.

Entretanto, mesmo tendo um desempenho robusto sob condições complexas, em algumas situações o desempenho do controlador PID é prejudicado devido as não-linearidades dos sistemas que são controlados. Como exemplo, mudanças inesperadas no momento de inércia no rotor de um motor podem causar grandes perturbações em seu comportamento, levando a uma instabilidades no processo produtivo (CHEN *et al.*, 2018). Esse problema nos motivou a buscarmos uma forma de controle mais tolerante as mudanças dos parâmetros do sistema.

2.9.2 Parâmetros do Controlador PID

Segundo Graf (2016), a resposta de um controlador PID padrão é estabelecido por três parâmetros: o ganho proporcional (k_p), o ganho integral (k_i) e o ganho derivativo (k_d). A Equação (8) abaixo consiste na função de transferência no domínio do tempo que relaciona os parâmetros do controlador PID e a resposta do sistema $c(t)$, onde $e(\tau)$ é o erro, t é o tempo e τ é o intervalo de integração. Já a Equação (9) abaixo é a mesma função de transferência, porém no domínio da frequência, onde $G(s)$ é a resposta do sistema e s é a frequência complexa.

$$c(t) = k_p * e(t) + k_i * \int_0^t e(\tau)d\tau + k_d * \frac{de(t)}{dt} \tag{8}$$

$$G(s) = k_p + k_i * \frac{1}{s} + k_d * s \tag{9}$$

O erro $e(t)$ é definido como a diferença entre o estado desejado para o sistema, chamado de *Setpoint* (SP), e o estado atual do sistema, chamado de *Process Value* (PV), como vemos na Equação (10) abaixo (NISE, 2012).

$$e(t) = SP(t) - PV(t) \tag{10}$$

Segundo Ogata (2010), a resposta de um controlador PID se divide em duas partes – a resposta transitória e a resposta estacionária. A resposta transitória se refere ao comportamento do controlador entre um estado inicial e um estado final quando se realiza uma alteração de SP. Já a resposta estacionária é o comportamento final do sistema quando o tempo tende ao infinito.

Nise (2012) afirma que alterando os parâmetros k_p, k_i e k_d é possível determinar o comportamento das respostas transitória e estacionária do controlador PID. O autor mostra que esse conjunto de valores define a velocidade de resposta transitória e sua acurácia em regime permanente e que mesmo sendo apenas 3 parâmetros, não é fácil encontrar os valores exatos que determinam os requisitos desejados de velocidade e acuraria para o processo porque quando se privilegia um desses fatores o outro acaba por se deteriorar.

Ang *et al.* (2006) explica que k_p fornecer uma ação de controle geral proporcional ao sinal de erro. Já k_i está relacionado ao erro em regime permanente. Por fim, k_d diz respeito a velocidade de resposta transitória do controlador. O efeito no comportamento do controlador devido a alteração no valor de cada parâmetro será detalhado mais adiante.

Já sabemos o que é um controlador PID e quais são seus parâmetros de ajuste. Na próxima subseção, será apresentado como analisar o comportamento de um controlador PID para que se possa determinar se os valores escolhidos dos parâmetros geram uma saída que satisfaça os requisitos desejados de velocidade e acuraria.

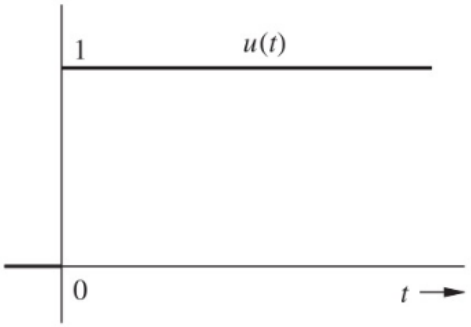
2.9.3 Análise da Resposta do Controlador

O desempenho de um controlador PID é analisado com base na sua resposta transitória e de regime permanente a uma entrada degrau unitário, pois quando ela é conhecida, é possível calcular a resposta para qualquer outro tipo de entrada (OGATA, 2010).

Segundo Haykin e Veen (2001), a entrada degrau unitário $u(t)$ é definida pela Equação (11) abaixo e pode ser vista no Gráfico 3, onde t é o tempo. Intuitivamente, vemos que ela é uma entrada brusca e que pode ser gerada facilmente devido sua simplicidade.

$$u(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases} \tag{11}$$

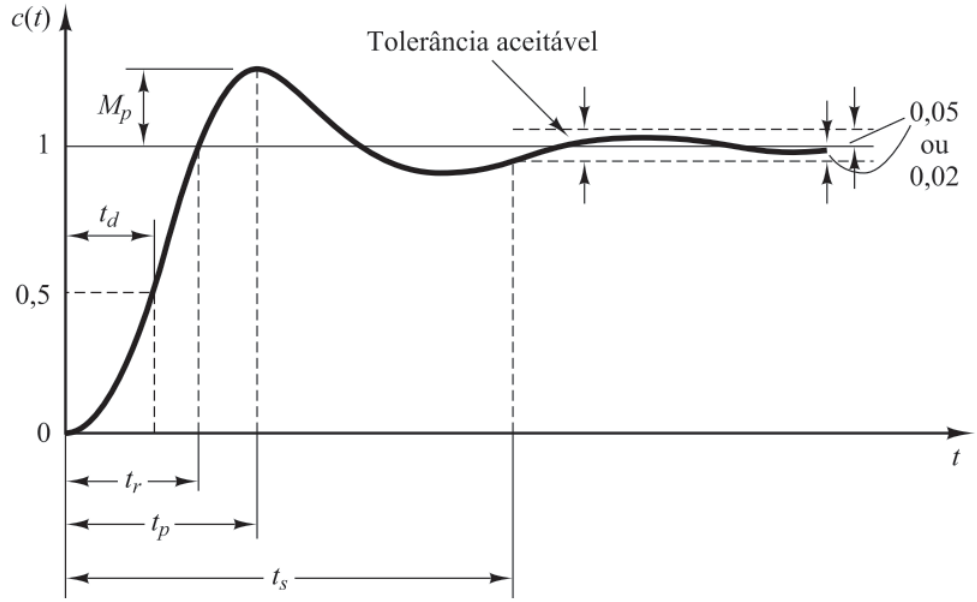
Gráfico 3 – Entrada Degrau Unitário



Fonte: Lathi, 2007.

Para analisar se o controlador atende aos requisitos de velocidade de resposta transitória e de acurácia em regime permanente estabelecidos pelo projetista do sistema, Ogata (2010) afirma que são utilizados os parâmetros visualizados no Gráfico 4 abaixo, onde $c(t)$ é a saída do sistema em um determinado tempo t.

Gráfico 4 – Parâmetros de Avaliação da Resposta do Controlador



Fonte: Ogata, 2010.

Vamos definir cada um deles logo abaixo:

1) Tempo de Atraso t_d: É o tempo necessário para que a resposta alcance 50% de seu valor final pela primeira vez;
2) Tempo de Subida t_r: É o tempo necessário para que a resposta alcance 100% de seu valor final pela primeira vez;
3) Tempo de Pico t_p: É o tempo necessário para que a resposta atinja o primeiro pico de sobressinal;
4) Sobressinal Máximo M_p: É o valor máximo da curva da resposta, medido a partir da unidade e definida pela Equação (12), onde $c(\infty)$ é a resposta em regime estacionário;

$$M_p = \frac{c(t_p) - c(\infty)}{c(\infty)} * 100\% \qquad (12)$$

5) Tempo de Acomodação t_s: É o tempo necessário para que a curva de resposta alcance valores em uma faixa de 2% ou 5% em torno do valor final. A largura dessa faixa é definida pelo projetista. No nosso estudo é 2%;
6) Erro Estacionário E_s: É a diferença percentual entre o estado desejado para o sistema SP e o estado atual efetivo do sistema em regime permanente $c(\infty)$. É definido matematicamente conforme a Equação (13).

$$E_s = \frac{SP - c(\infty)}{c(\infty)} * 100\% \quad (13)$$

Nos resta analisarmos quais são os objetivos e métodos de ajuste dos parâmetros de um controlador PID. Esses assuntos serão abordados na próxima subseção.

2.9.4 Objetivos e Métodos de Ajuste dos Parâmetros de Controle

Segundo Ang *et al.* (2006), os parâmetros de controle do controlador PID são ajustados de modo que o sistema seja estável e atenda, em diferentes níveis, um ou mais dos seguintes objetivos, a depender das necessidades específicas da aplicação:

1) Robustez de estabilidade;
2) Resposta mais rápido no transitório, o que inclui o t_d, t_r, t_p, M_p e t_s;
3) Melhor desempenho no regime permanente, incluindo menor E_s e tratativa de distúrbios;
4) Maior robustez contra incertezas de modelagem;
5) Atenuação de ruídos.

Ressaltamos que muitos desses objetivos são disjuntos, como já comentamos anteriormente. Isso significa que, em muitos casos, a priorização de um objetivo leva a degeneração dos outros. A Tabela 1 ilustra algumas das relação existentes entre os parâmetros da resposta e o aumento do valor dos parâmetros de controle.

Tabela 1 – Relação Entre os Parâmetros da Resposta e os Parâmetros de Controle

	t_d	t_r	M_p	t_s	E_s	Estabilidade
$\uparrow k_p$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$
$\uparrow k_i$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$
$\uparrow k_d$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$ ou $\uparrow$	$\uparrow$

Fonte: Ang *et al.*, 2006.

Com os objetivos de controle estabelecidos pelo projetista do sistema, resta ajustar os parâmetros do controlador para alcançá-los. Isso é feito utilizando algum dos métodos existentes de ajuste de parâmetro de controle. Esses métodos são agrupados por Ang *et al.* (2006) de acordo com sua natureza e uso:

1) Método Analítico;
2) Método Heurístico;

3) Método de Resposta a Frequência;
4) Método de Otimização;
5) Método de Ajuste Adaptativo.

O método que nos interessa e que foi utilizado no estudo de caso é o Método Analítico. Nele, os parâmetros do controlador PID são calculados a partir das funções de transferência do sistema a ser controlado e dos objetivos desejados (ANG *et al.*, 2006).

Utilizamos esse método, pois tínhamos em mãos um modelo bem definido do sistema, ou seja, as equações dinâmicas do motor CC. Essas equações, que serão apresentadas mais adiante, são amplamente disponibilizadas nas literaturas sobre o assunto e graças a elas, juntamente com as definições de projeto acerca da velocidade e acurácia da resposta, pudemos encontrar facilmente os parâmetros do controlador PID utilizado no estudo de caso. Essa etapa será mais detalhada adiante.

Concluímos a apresentação de todos os conceitos necessários sobre controladores PID para o entendimento do estudo de caso. Portanto, considerando todas as subseções anteriores, afirmamos que toda a teoria utilizada no desenvolvimento do projeto foi apresentada. Com isso, finalizamos aqui o referencial teórico.

3 ESTUDO DE CASO

Iremos apresentar nesse capítulo o estudo de caso realizado nesse trabalho. Vamos iniciar com uma breve apresentação de todas as ferramentas utilizadas. Em seguida, daremos uma visão geral de todo o escopo do projeto. Posteriormente, nas subseções seguintes, detalharemos mais profundamente cada etapa do desenvolvimento da solução.

Todos os arquivos gerados nesse estudo de caso estão disponíveis para *download* no repositório do GitHub <https://github.com/ArturBarreto/ControladorNeuralMotorCC/>. Implementações comentadas, conjuntos de dados, resultados, instruções de uso, instruções de licença, como citar esse trabalho e demais conteúdos e informações pertinentes estão disponíveis nesse endereço. Caso haja alguma dúvida, sinta-se à vontade para entrar em contato.

3.1 Ferramentas Utilizadas

Já comentamos algumas vezes que a proposta desse trabalho é modelar uma RNA que controle a velocidade de um motor de Corrente Contínua (CC). Para isso, utilizamos algumas ferramentas computacionais, que serão apresentadas a seguir:

1) Linguagem de Programação Python;
2) *TensorFlow*;
3) Keras;
4) Implementação da DBN;
5) MATLAB®;
6) Simulink®;
7) *Toolbox PID Controller Design for a DC Motor*.

3.1.1 Linguagem de Programação Python

O Python é uma linguagem de programação criada por Guido van Rossum em 1991 com o objetivo de ser uma linguagem de fácil codificação e manutenção. Segundo a página oficial do Python, podemos destacar as seguintes características da linguagem:

1) É uma linguagem de programação orientada a objetos de alto nível e de fácil utilização que permite que o programador desenvolva seus códigos de forma procedural ou funcional;

2) Sua sintaxe é simples;
3) Possui um alto nível de abstração, estando bastante próximo da linguagem humana;
4) É executado em ambientes Windows, Linux, MacOS, Android e outros sistemas operacionais;
5) É gratuito;
6) Possui código aberto;
7) Possui uma imensa comunidade ao redor do globo;

Destacamos que ela foi escolhida, dentre todas as linguagens de programação disponíveis no mercado, por ser bastante utilizada pelos programadores de IA e possuir ferramentas poderosas que auxiliam no desenvolvimento de sistemas de ML e DL, como o *TensorFlow* e Keras.

3.1.2 TensorFlow e Keras

O *TensorFlow* é uma plataforma de código aberto, disponível para Python e outras linguagens de programação, utilizada para construir modelos de ML e DL. Essa ferramenta foi desenvolvida por engenheiros e pesquisadores do Google *Brain Team* no departamento de pesquisas de *machine learning* do Google.

Segundo a página oficial do *TensorFlow*, essa plataforma possui um ecossistema flexível e abrangente de ferramentas, bibliotecas e recursos desenvolvidos em código aberto pela comunidade que permitem que os pesquisadores desenvolvam suas soluções de ML ou DL com o que há de mais modernas em se tratando de conceitos e técnicas.

O *TensorFlow* oferece vários níveis de abstração que podemos escolher conforme nossas necessidades de projeto.

Para a maioria das soluções, é possível utilizar uma API chamada Keras na qual os modelos são construídos basicamente conectando blocos de código de fácil utilização. Essa API foi desenvolvida por François Chollet e hoje está integrada ao TensorFlow.

Se é preciso uma maior flexibilidade nas customizações dos modelos, utiliza-se a API de subclasse de modelo onde é possível criar camadas, funções de ativações e algoritmos de treinamento personalizados.

Destacamos que o Keras foi a API escolhida para criar os modelos das RNAs de nosso estudo de caso por ser o caminho mais simples e por atender todas nossas necessidades.

3.1.3 Implementação DBN

Afirmamos no referencial teórico que a arquitetura utilizada na RNA desenvolvida no estudo de caso consiste em uma *Deep Belief Network* (DBN) com retropropagação. Já apresentamos suas principais características na subseção 2.5 e mais detalhes serão dados adiante, quando apresentarmos a evolução topológica da RNA.

Albert Bup (2017) codificou uma implementação simples e rápida em Python das *Deep Belief Networks* baseadas nas Máquinas de Boltzmann Restritas. Ele utilizou as bibliotecas NumPy e *TensorFlow* para criar seu código.

Com essa implementação, foi possível utilizar o conceito das DBNs no *TensorFlow* tendo a disposição toda a lógica do seu funcionamento já implementada e testada.

Assim, nossos esforços se transferiram para o que é mais importante e interessante na modelagem de RNAs – os testes topológicos da rede para a determinação de seus hiperparâmetros e a análise das respostas para validação do modelo.

Veremos esse processo detalhadamente mais adiante. Porém, podemos adiantar que os hiperparâmetros da rede dizem respeito a quantidade de camadas, quantidade de neurônios por camada, tamanho de lote de treinamento, quantidade de épocas, a função do erro utilizada, a taxa de aprendizagem utilizada, dentro outros (CHOLLET, 2018). A determinação dos hiperparâmetros de uma RNA determina a sua capacidade de solução do problema.

3.1.4 MATLAB®, Simulink® e PID Controller Design for a DC Motor

O MATLAB® é uma outra ferramenta utilizada em nosso estudo de caso. Ele consiste em um ambiente de computação numérica de linguagem de programação proprietária desenvolvida pela MathWorks. O MATLAB® é utilizado primordialmente para realizar manipulações matriciais e de funções, além de plotagem de gráficos. Porém, com ele é possível realizar tarefas das mais variadas naturezas. No seu site oficial, a MathWorks anuncia que é possível fazer ajuste de curvas, otimizações em geral, estudos estatísticos e muito mais.

O Simulink® é um pacote adicional do MATLAB® que adiciona a possibilidade de realizar modelagens, simulações e análises de sistemas dinâmicos de múltiplos domínios. Segundo a MathWorks, com ele é possível utilizar uma ferramenta gráfica de diagramação de

blocos e um conjunto personalizável de bibliotecas de blocos para montar os sistemas a serem simulados.

A última ferramenta que utilizamos em nosso estudo de caso é a *toolbox* do Simulink® chamada de *PID Controller Design for a DC Motor* desenvolvida por Turevskiy (2016). Nessa *toolbox*, o usuário passa os parâmetros de um motor CC e de um controlador PID que são representados por blocos interligados. Com isso, o sistema é simulado e seu comportamento determinado, permitindo que o projetista analise o sistema para diversas situações e parametrizações possíveis e, dessa forma, estabeleça os impactos de cada mudança de parâmetro no funcionamento do sistema. Essas informações são utilizadas para parametrizar o controlador PID com o intuito de que o comportamento do motor CC satisfaça as condições de funcionamento determinadas no projeto do sistema.

Veremos mais detalhadamente nas próximas subseções como a *toolbox PID Controller Design for a DC Motor* funciona e como ela foi utilizada em nosso estudo de caso. Porém, podemos adiantar que ela foi utilizada para gerar conjuntos de dados de treinamento e de teste para a RNA desenvolvida.

3.2 Visão Geral do Projeto

O estudo de caso realizado neste trabalho tem como objetivo modelar uma RNA que controle a velocidade de um motor de Corrente Contínua (CC). Para modelagem da rede, utilizou-se a linguagem de programação Python e suas frameworks de *Machine Learning TensorFlow* e Keras. Essa RNA foi denominada de Controlador Neural (CN).

O CN foi treinado com as entradas e saídas de um controlador PID ligado a um motor CC simulados na *toolbox PID Controller Design for a DC Motor* do Simulink® do MATLAB® R2018a.

Após obter uma rede treinada com um resultado satisfatório, conforme método descrito mais adiante, partimos para a análise de sua eficiência (velocidade de resposta transitória) e eficácia (acurácia em regime permanente). Para isso, realizamos uma comparação entre o CN e o controlador PID, analisando os seguintes parâmetros de resposta para uma entrada degrau:

1) Tempo de Atraso t_d;
2) Tempo de Subida t_r;
3) Tempo de Pico t_p;

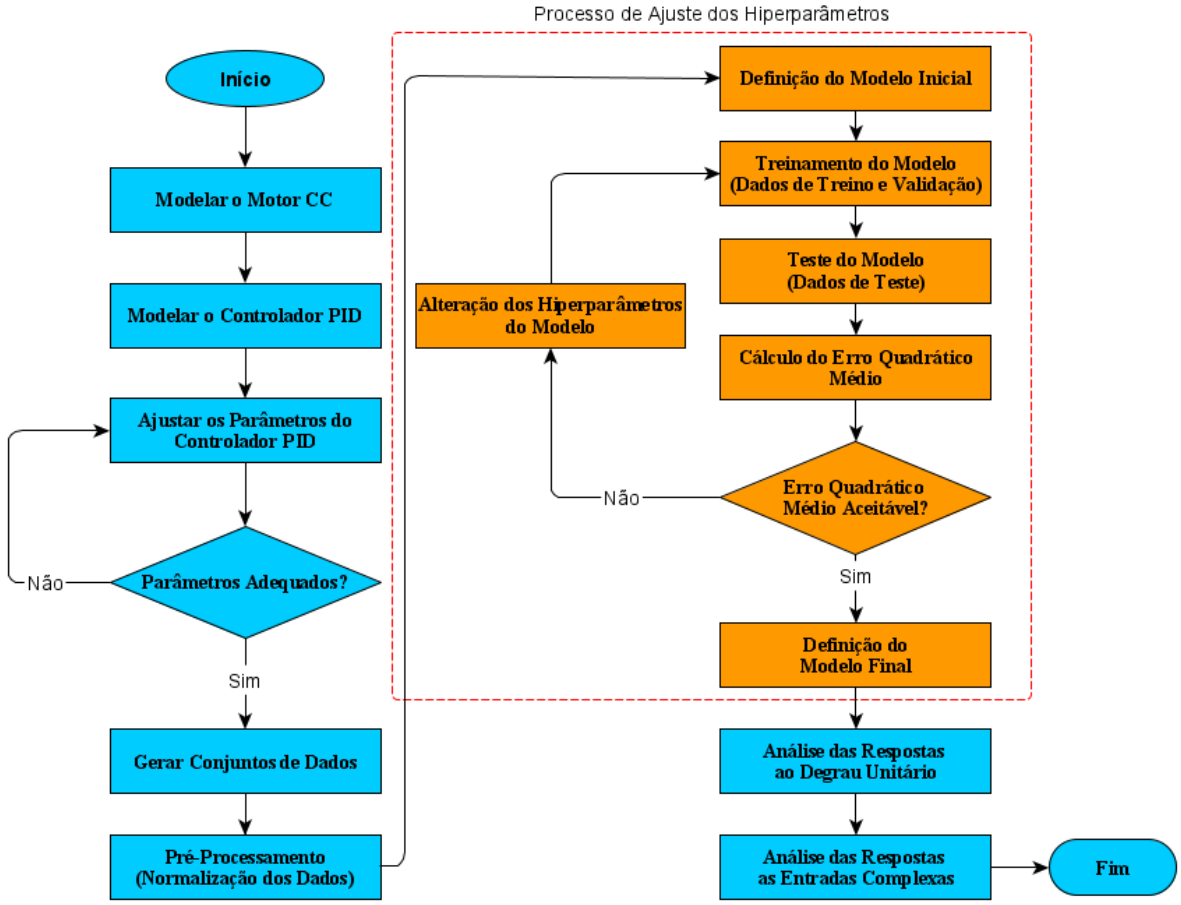
4) Sobressinal Máximo M_p;
5) Tempo de Acomodação t_s;
6) Erro Estacionário E_s.

Além disso, também analisamos algumas características do comportamento da resposta da RNA para uma entrada complexa, ou seja, o conjunto de teste.

Ressaltamos que os parâmetros do controlador PID foram mantidos constantes em todos esses testes.

Uma visão geral do projeto pode ser vista na Figura 15 abaixo. As etapas em azul foram implementadas no MATLAB® R2018a ou no Simulink®. Já as etapas em amarelo foram implementadas em Python, utilizando as bibliotecas TensorFlow e Keras. O processo de ajuste dos hiperparâmetros do modelo será mais detalhado na seção 3.9.

Figura 15 – Fluxograma do Projeto



Fonte: Elaborado pelo autor.

Nas próximas subseções, vamos detalhar a caracterização do problema estudado e o método para sua solução. Logo após, discorremos sobre a modelagem do motor e do

controlador PID que geraram os dados de entradas e saídas utilizadas para treinar e testar a RNA. Em seguida, descrevemos a modelagem da RNA propriamente dita, mostrando a evolução de sua topologia e justificando sua arquitetura final. Por fim, são realizadas as análises de desempenho do CN em comparação ao controlador PID.

3.3 Caracterização do Problema

Segundo Chapman (2013), os motores elétricos estão presentes em todos as áreas da sociedade. Nas nossas casas, eles são os responsáveis por acionar, por exemplo, as geladeiras, aspiradores de pó, aparelhos de ar condicionado e ventiladores. Nas indústrias, são os responsáveis por gerar a força motriz necessária nos mais variados processos.

Para controlar os motores nas mais diversas aplicações e ajustar seu funcionamento as necessidades do processo, costuma-se monitorar sua velocidade de rotação e ajustá-la a um nível que satisfaça os critérios almejados. Esse controle é normalmente realizado por um controlador Proporcional Integrativo Derivativo, ou simplesmente controlador PID, que está presente na maioria das aplicações industriais por ser uma solução já consolidada e conhecida por sua eficiência (ANG *et al.*, 2006).

Entretanto, segundo Abraham e Shrivastava (2018), o desempenho do controle PID aplicado a sistemas não-lineares varia bastante e depende tanto do ajuste dos parâmetros do controlador - o ganho proporcional (k_p), o ganho integral (k_i) e o ganho derivativo (k_d) – como dos próprios parâmetros do sistema em si – o momento de inércia do rotor, por exemplo.

Abraham e Shrivastava (2018) citam como ilustração dessa desvantagem o caso de uma válvula motorizada usada na aplicação de polímeros em uma indústria. Eles afirmam que se ela permanecer sem operar por um determinado período, ocorrerá um aumento no momento de inércia do rotor devido ao depósito de partículas no sistema, levando a uma maior fricção da válvula.

Nesse caso, se for utilizado um controlador PID padrão, ele não será capaz de fornecer um ajuste adequado de velocidade porque os parâmetros do sistema que foram utilizados para calibrar os parâmetros do controlador k_p, k_i e k_d não correspondem a atual situação do sistema, pois a fricção adicional existente não está sendo considerada pelo controlador.

Portanto, graças aos fatores externos e suas não-linearidades, o ajuste dos parâmetros de um controlador PID se torna bastante complicado, sendo comum a ocorrência de perturbações nos processos que utilizam esse tipo de solução (CHEN *et al.*, 2018).

Abordamos na seção anterior a grande capacidade do *Deep Learning* na solução de problemas não-lineares e já comentamos que essas abordagens inteligentes estão em alta, sendo aplicadas nos mais diversos campos do conhecimento. Especificamente na área de controle é fácil encontrar vários artigos com resultados satisfatórios que demonstraram que o DL é aplicável nessa área. Portanto, afirmamos que esses são os motivos que nos levaram a utilizar o DL para realizar o controle de velocidade de um motor CC. A proposta do modelo será detalhada na próxima subseção.

3.4 Método Proposto

Há muitos estudos que analisam a aplicabilidade do DL em soluções de controle. Moe *et al.* (2018) apresentam, além de um resumo do estado da arte da utilização do ML em sistemas de controle, um estudo de caso em que uma rede neural é treinada para imitar um controlador de velocidade de um navio autônomo. Já Tai *et al.* (2015) apresentam uma pesquisa que se concentra em soluções de DL direcionadas ao controle de sistemas robóticos.

Em especial, Chen *et al.* (2018) desenvolveram uma estratégia de controle de velocidade de servomotores utilizando algoritmos de DL. Eles foram motivados pela dificuldade na parametrização do controle PID padrão devido a existência de distúrbios no momento de inércia que ocorrem naturalmente nas aplicações práticas. A solução proposta por eles é mais tolerante as variações do sistema, resultando em um controle de processo mais robusto.

Para Abraham e Shrivastava (2018), um sistema de controle que é capaz de realizar um autoajuste de seus parâmetros é o ideal para o caso de processos com características que variam no tempo. Já citamos várias fontes que nos mostram que sistemas de controle baseados em DL são capazes de se ajustar as condições do processo graças sua capacidade de modelar problemas não-lineares complexos.

Neste contexto, vamos modelar uma RNA utilizando um algoritmo de DL para atuar como um controlador de velocidade de um motor de corrente contínua.

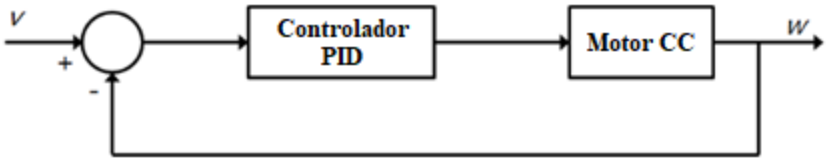
O primeiro passo para a criação desse CN é modelar o motor CC que será controlado. Todos seus parâmetros devem ser passados para um simulador que deve exprimir seu comportamento o mais próximo possível da realidade.

O segundo passo é fazer o mesmo para um controlador PID ligado a este motor CC. Os parâmetros k_p, k_i e k_d definem o comportamento do controlador quanto a velocidade de resposta e acurácia em regime permanente e esses parâmetros devem ser calibrados e inseridos no simulador conforme as necessidades do processo. Essa etapa é bastante importante, pois o CN aprenderá o comportamento do controlador PID. Portanto, o PID deve ser capaz de satisfazer os requisitos do processo em condições ideais.

As entradas do controlador PID são a velocidade desejada, também chamada de *set point* (SP), e a velocidade atual do motor CC. Já a saída consiste no nível de tensão que deve ser colocado nos terminais da armadura do motor para que a velocidade desejada seja alcançada.

Ao final desses dois passos, tem-se um sistema composto por um controlador PID que controla um motor CC, conforme a Figura 16 abaixo. Para essa parte do experimento, utilizamos uma *toolbox* do Simulink® do MATLAB® R2018a. Essas etapas serão mais detalhadas nas próximas subseções.

Figura 16 – Simulação do Controlador PID e do Motor CC

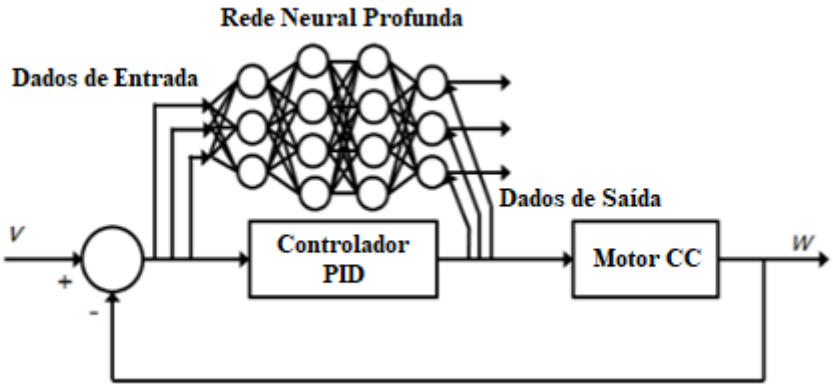


Fonte: Adaptado de Cheon *et al.,* 2015.

Em seguida, modela-se uma RNA em Python utilizando as frameworks de *Machine Learning* Keras e *TensorFlow* e se utiliza as entradas e saídas do controlador PID em seu treinamento e teste, conforme ilustrado pela Figura 17 abaixo, na esperança de que a RNA consiga aprender a se comportar como um controlador PID.

Como veremos mais adiante, esse processo é bastante demorado, pois é necessário testar várias topologias diferentes para se determinar um modelo de RNA que seja adequado para solucionar o problema.

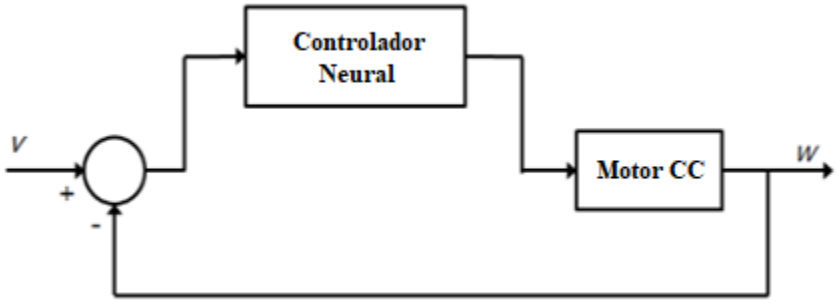
Figura 17 – Treinamento do CN



Fonte: Adaptado de Cheon *et al.,* 2015.

Logo após a determinação da topologia mais adequada e seu respectivo treinamento, tem-se um uma RNA treinada capaz de substituir o controlador PID. O controlador PID é então desconectado e o CN assume o controle do motor CC, conforme ilustrado pela Figura 18 abaixo.

Figura 18 – Substituição do Controlador PID pelo CN



Fonte: Adaptado de Cheon *et al.,* 2015.

Para validar o CN modelado, realizou-se uma série de testes comparativos com o controlador PID. Esses testes, bem como seus resultados, serão apresentados mais adiante. Entretanto, antes disso, vamos detalhar a simulação do motor CC e do controlador PID nas próximas subseções.

3.5 Modelagem do Motor CC

A modelagem e a simulação do motor CC foram realizadas utilizando a *toolbox PID Controller Design for a DC Motor* desenvolvida por Turevskiy (2016). Essa *toolbox* para o Simulink® é compatível para versões R2009b em diante do MATLAB®.

Nessa *toolbox*, o motor CC é modelado como um bloco onde um nível de tensão V (volts) e a velocidade angular ω (rad/s) são, respectivamente, a entrada e a saída do sistema. As entradas são processadas em saídas utilizando as funções de transferência (14) e (15), denominadas de equações dinâmicas de um motor CC, onde J é o momento de inércia do rotor ($\mathrm{kg.m^2}$), K_t é a constante de torque do motor ($\mathrm{N.m.rad/A}$), i_a é a corrente de armadura (A), b é o coeficiente de atrito viscoso do motor ($\mathrm{kg.m^2/s}$), L é a indutância elétrica (H), R é a resistência elétrica (Ω) e K_e é a constante de força eletromotriz ($\mathrm{V.s/rad}$) (CHEON *et al.*, 2015).

$$\frac{d\omega}{dt} = \frac{1}{J}(K_t * i_a - b * \omega) \tag{14}$$

$$\frac{di_a}{dt} = \frac{1}{L}(V - R * i_a - K_e * \omega) \tag{15}$$

O valor dos parâmetros do motor CC utilizados na simulação podem ser consultados na Tabela 2 abaixo.

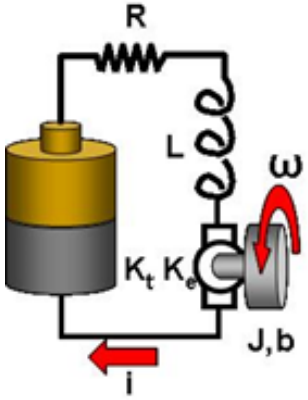
Tabela 2 – Parâmetros do Motor CC

J $[\mathrm{kg.m^2}]$	K_t $[\mathrm{N.m.rad/A}]$	b $[\mathrm{kg.m^2/s}]$	L $[\mathrm{H = V.s/A}]$	R $[\Omega = \mathrm{V/A}]$	K_e $[\mathrm{V.s/rad}]$
0,01	0,01	0,1	0,5	1	0,01

Fonte: Elaborado pelo autor.

Podemos visualizar na Figura 19 abaixo uma representação gráfica do bloco da *toolbox* que modela, utilizando as Equações (14) e (15) e os parâmetros Tabela 2, o comportamento de um motor CC.

Figura 19 – Bloco do Motor CC



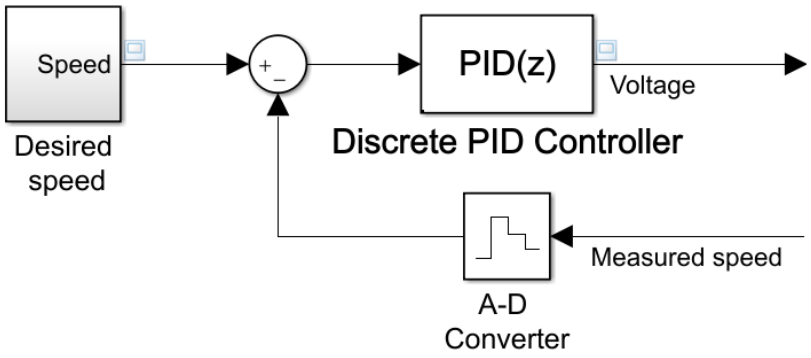
Fonte: Turevskiy, 2016.

Com a modelagem do motor pronta, o próximo passo é montar e parametrizar um controlador PID que satisfaça as necessidades de tempo de resposta e acurácia determinadas para o processo. Esse é o assunto da próxima subseção.

3.6 Modelagem do Controlador PID

Na *toolbox* criada por Turevskiy (2016), o controlador PID é modelado como um bloco de duas entradas digitais e uma saída analógica. As entradas consistem na velocidade desejada (*set point)* e na velocidade atual medida do motor CC. Já a saída consiste no nível de tensão que deve ser colocado nos terminais de armadura do motor para que a velocidade desejada seja alcançada. Podemos visualizar na Figura 20 abaixo uma representação gráfica do bloco da *toolbox* que modela o controlador PID.

Figura 20 – Bloco do Controlador PID



Fonte: Elaborado pelo autor.

A velocidade atual medida é proveniente do motor CC e deve passar por um conversor analógico-digital antes de alimentar o controlador PID.

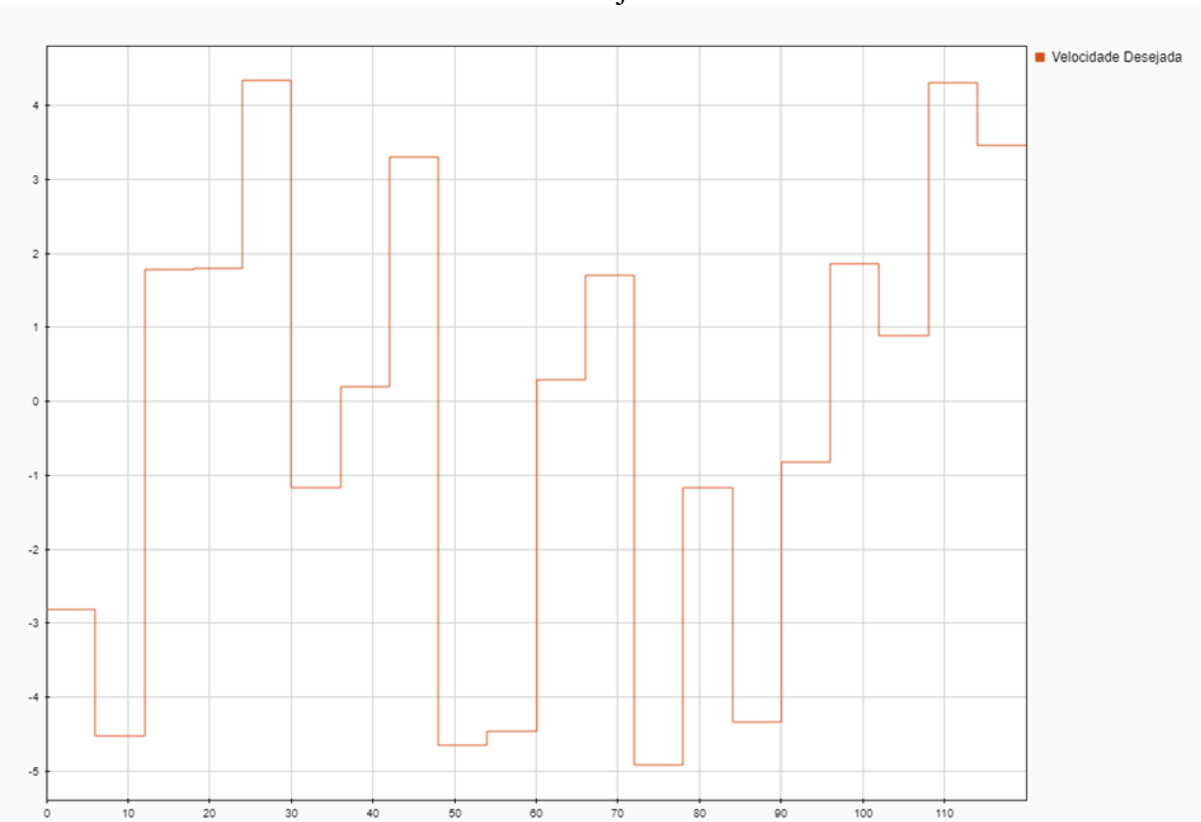
Já a velocidade desejada é estabelecida por um bloco *Uniform Random Number* do Simulink®. Esse bloco gera um sinal quadrado aleatório que representa as velocidades demandadas para o motor em cada instante t da simulação. Definimos arbitrariamente que essas velocidades estariam entre -5 e 5 rad/s, permanecendo em cada nível de velocidade por 6 segundos. Com isso, temos um perfil de velocidade desejado para o funcionamento do motor CC.

Ressaltamos que esse tempo de permanência de 6 segundos em cada nível de velocidade se deu por conta do tempo de acomodação t_s do controlador PID ser de 2,74 segundos. Portanto, estabelecemos uma margem de segurança de mais de 2 vezes para garantir

que a velocidade do motor se estabilizasse em cada nível e assim tivéssemos dados de regime permanente para cada um desses níveis.

O Gráfico 5 ilustra o perfil de velocidade desejado gerado por esse bloco para uma simulação de 120 segundos.

Gráfico 5 – Perfil Arbitrário de Velocidade Desejada



Fonte: Elaborado pelo autor.

Antes de simular o sistema e obter os dados de entrada e saída que serão utilizados para treinar a RNA, é necessário parametrizar o controlador PID. Como já vimos no referencial teórico, ele possui apenas três parâmetros principais que definem a velocidade de resposta transitória e a acurácia em regime permanente. Entretanto, não é fácil encontrar os valores exatos para alcançar os requisitos desejados de velocidade e acuraria, pois quando se privilegia um desses fatores o outro acaba sendo prejudicado.

A Equação (16) consiste na função de transferência no domínio da frequência do bloco que modela o controlador PID, onde $G(s)$ é a resposta, k_p é o ganho proporcional, k_i é o ganho integral, k_d é o ganho derivativo, s é a frequência complexa e N é o divisor de filtro derivado de primeira ordem.

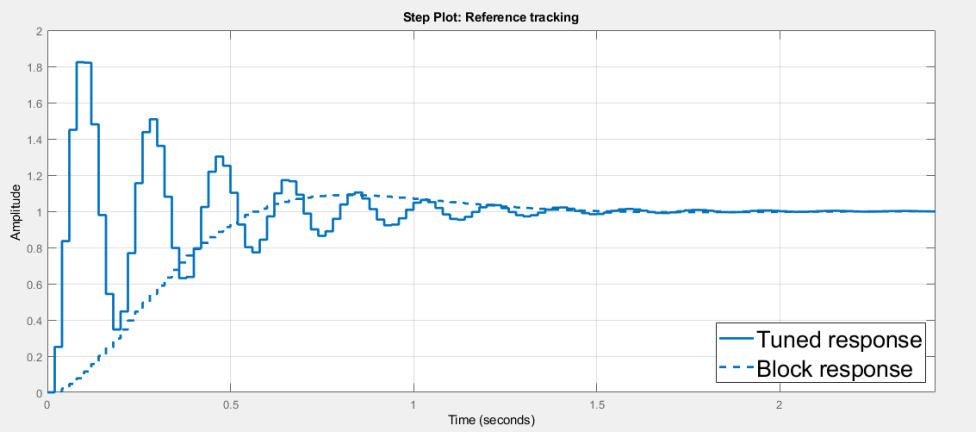
$$G(s) = k_p + k_i * \frac{1}{s} + k_d * \frac{N}{1 + N * \frac{1}{s}} \quad (16)$$

Segundo Ang *et al.* (2006), um controlador PID não limita os ganhos de alta frequência, o que permite que o sinal de controle de nível alto tenha um valor teoricamente infinito na mudança do nível do sinal de referência (*set point*) ou na ocorrência de perturbações aleatórias. Para evitar esse problema, a maioria dos controladores PID possuem alguma forma de filtro no diferenciador, que nesta *toolbox* é representada pela variável N.

Utilizando uma ferramenta nativa do Simulink® chamada de PID *Tuner App*, foi possível ajustar os parâmetros do controlador PID para controlar o motor CC através das funções de transferência representadas pelas Equações (14), (15) e (16) e da passagem de requisitos arbitrários de velocidade e acurácia desejados para a resposta. Neste processo, foi observado que um aumento na robustez diminui o desempenho e vice-versa, comportamento já esperado e difundido pela literatura.

Quando o sistema foi ajustado para um melhor desempenho, ou seja, uma velocidade de resposta mais rápida, a saída do controlador teve considerada oscilação por um período transitório, como mostra o Gráfico 6 abaixo.

Gráfico 6 – Resposta do Controlador Priorizando a Velocidade de Resposta

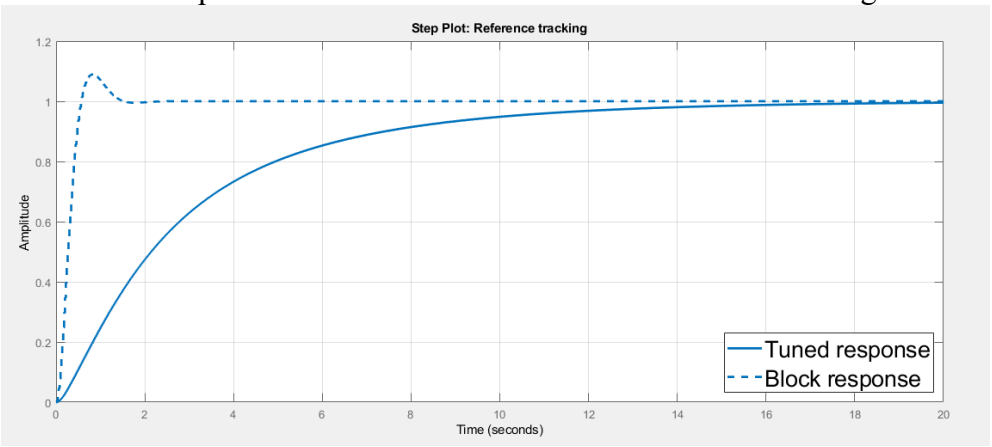


Fonte: Elaborado pelo autor.

Quando o sistema foi ajustado para uma maior robustez, ou seja, uma maior acurácia em regime permanente, a resposta ficou mais lenta, conforme o Gráfico 7 abaixo.

Portanto, buscou-se um meio termo entre desempenho e robustez, ajustando os parâmetros do controlador PID para os valores apresentados na Tabela 3. O Gráfico 8 ilustra a resposta do controlador para esses parâmetros.

Gráfico 7 – Resposta do Controlador Priorizando a Acurácia em Regime Permanente



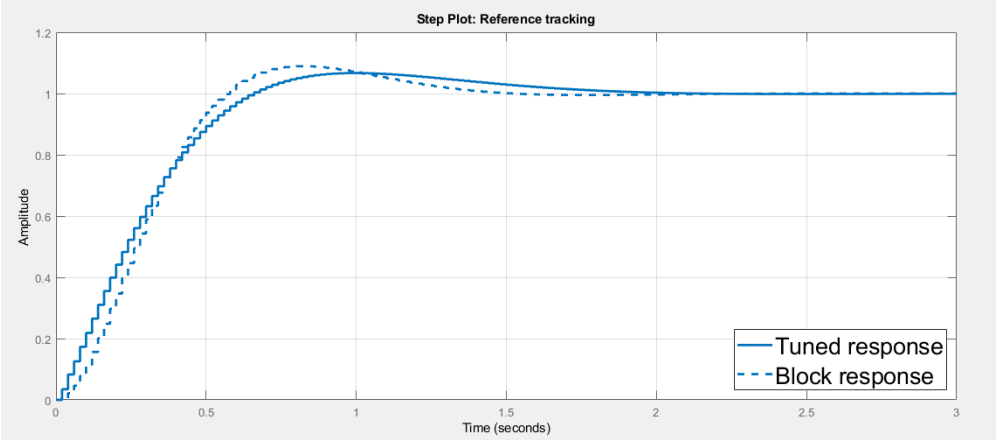
Fonte: Elaborado pelo autor.

Tabela 3 – Parâmetros Finais do Controlador PID

k_p	k_i	k_d	N
5,6086	17,2550	0,0000	100,0000

Fonte: Elaborado pelo autor.

Gráfico 8 – Resposta do Controlador para um Meio Termo



Fonte: Elaborado pelo autor.

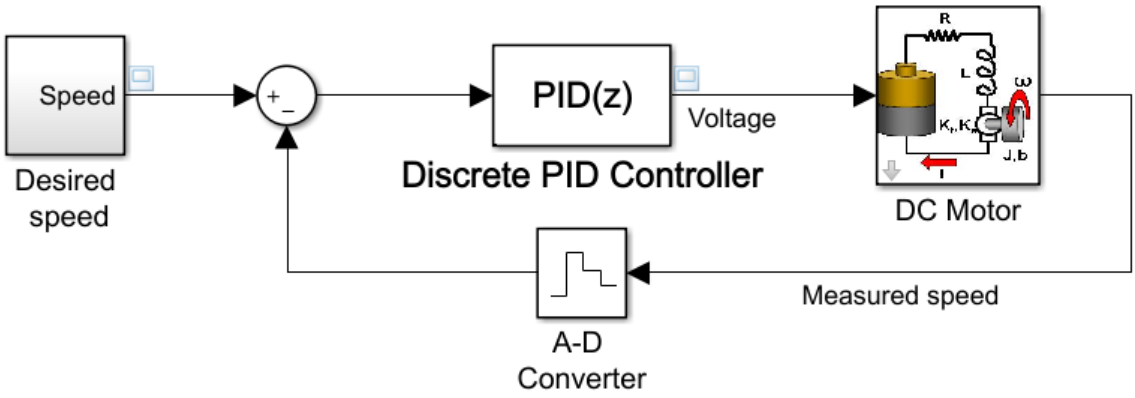
Com o motor CC e o controlador PID modelados, podemos simular todo o sistema e gerar os dados de entradas e saídas que serão utilizados para realizar o treinamento e testes da RNA.

3.7 Simulação do Sistema

O sistema completo composto pelo motor CC e o controlador PID pode ser visto na Figura 21 abaixo. Ressaltamos que o sistema original possuía dois blocos adicionais que não

foram utilizados e, portanto, foram retirados do sistema. O primeiro deles adicionava um distúrbio externo na carga mecânica no eixo do motor e o outro adicionava um ruído aleatório na leitura da velocidade do motor CC.

Figura 21 – Sistema do Motor CC e Controlador PID



Fonte: Elaborado pelo autor.

Com as modelagens e parametrizações do motor CC e do controlador PID realizadas conforme as subseções anteriores, o sistema é simulado.

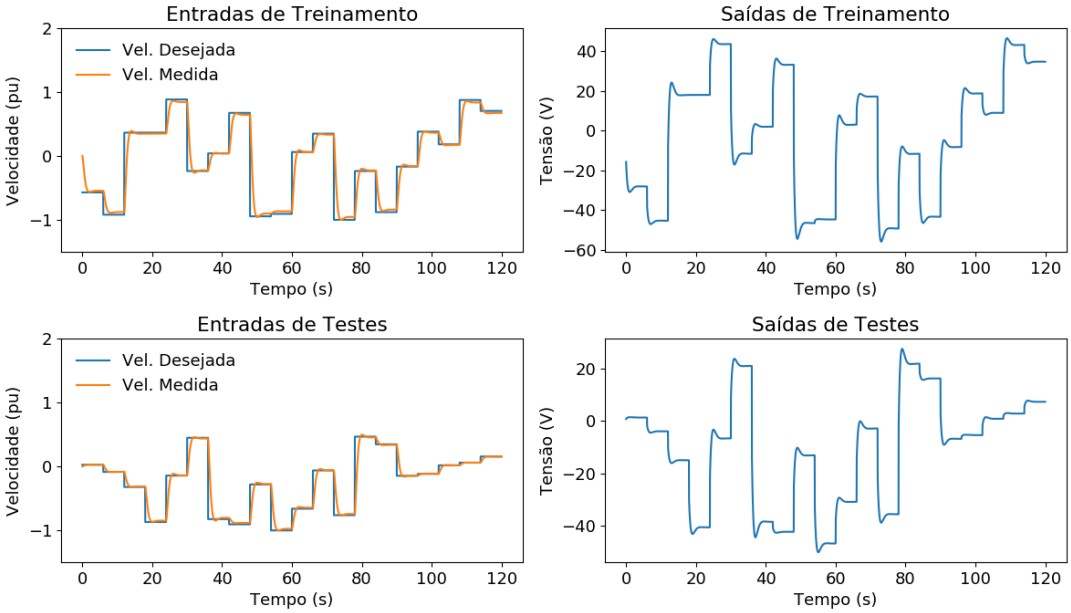
Foram realizadas várias simulações, a fim de gerar arquivos com dados de treino e teste de diversos tamanhos para que pudéssemos avaliar o impacto do volume de dados na eficácia do treinamento. O tempo total de cada simulação realizada variou entre 60 e 600 segundos, sempre aumentando em múltiplos de 60. Já o passo no tempo das simulações foi de 0,02 segundo.

As grandezas observadas nessas simulações foram a velocidade desejada, a velocidade medida do motor CC e o sinal de saída do controlador PID que representa o nível de tensão que deve ser colocado nos terminais de armadura do motor para que a velocidade desejada seja alcançada.

Com a observação dessas grandezas, é possível gerar os conjuntos de dados de entradas e saídas utilizados no treinamento e nos testes da RNA. Um desses conjuntos de dados, referente a simulação de 120 segundos, pode ser vistos no Gráfico 9 abaixo.

De posse dos dados de treinamentos e de testes, finalmente podemos nos dedicar a modelagem da RNA. Veremos na próxima subseção um método criado por Chollet (2018) utilizado em nosso trabalho. Esse método é aplicado na solução de qualquer tipo de problema de DL e consiste em um conjunto de procedimentos formais para construir, avaliar e refinar RNAs.

Gráfico 9 – Dados de Treinamento e Teste da RNA



Fonte: Elaborado pelo autor.

3.8 Fluxo de Trabalho na Modelagem de Problemas de DL

Nessa subseção, apresentaremos uma metodologia universal proposta por Chollet (2018) que pode ser utilizada para resolver qualquer problema de DL. Ela possui sete passos que discutiremos a seguir:

1) Definir o problema e montar o conjunto de dados;
2) Escolher uma métrica de sucesso;
3) Dividir os dados de treinamento, validação e teste;
4) Realizar o pré-processamento nos dados;
5) Encontrar um modelo com poder estatístico;
6) Encontrar um modelo que se sobreajusta aos dados;
7) Regular o modelo através do ajuste de seus hiperparâmetros.

3.8.1 Definir o Problema e Montar o Conjunto de Dados

O primeiro passo é definir o problema e montar o conjunto de dados. Nessa etapa, a primeira pergunta que devemos responder é qual é o tipo de problema que está sendo modelado. É uma classificação binária ou classificação multiclasse? É uma regressão escalar

ou regressão vetorial? Também é necessário determinar quais serão os dados de entrada e saída e como conseguiremos esses dados.

Responder essas questões é crucial, pois a identificação do tipo de problema orienta na determinação de vários aspectos da RNA como a escolha da arquitetura a ser utilizada e a definição da função do erro.

Além disso, a disponibilidade de dados é um fator limitante e bastante crítico, pois a RNA só é capaz de aprender a resolver um problema se houver dados representativos disponíveis para seu treinamento.

Nessa etapa, tem-se como hipótese que as saídas podem ser previstas com base nas entradas e que os dados disponíveis são suficientemente informativos para que a RNA aprenda o relacionamento elas. Entretanto, Chollet (2018) afirma que nem todos os problemas podem ser resolvidos e que o fato de agruparmos entradas X e saídas Y não significa que X contém informações suficientes para prever Y. Portanto, essas hipóteses só podem ser validadas ou não quando um modelo é montado e ajustado.

As discussões referentes a esse passo já foram apresentadas nesse trabalho.

3.8.2 Escolher uma Métrica de Sucesso

A segunda etapa é escolher uma métrica para medir o sucesso da rede, ou seja, definir um parâmetro que será analisado para determinar se o modelo está sendo capaz de resolver o problema ou não. A escolha dessa métrica é importante, pois ela serve como orientação na definição da função do erro, ou seja, a função que será minimizada pelo MGD, como visto no referencial teórico.

A métrica escolhida para o nosso estudo foi o MSE (*Mean Square Error* ou o Erro Quadrático Médio), um conceito estatístico bastante conhecido e consagrado. Ele mede a distância entre a resposta da RNA e a resposta esperada gerada pelo PID. O MSE serve como um termômetro da qualidade de resposta da rede e orienta os ajustes a serem realizados.

3.8.3 Dividir os Dados em Subconjuntos de Treinamento, Validação e Teste

O terceiro passo é definir a forma como os dados serão divididos e apresentados a RNA para que o modelo seja treinado, validado e testado. Chollet (2018) mostra três formas

para se fazer essa divisão. A escolha de qual método utilizar depende da quantidade de dados disponível. São eles:

1) Validação aleatória;
2) Validação com divisão aleatória em k-partições;
3) Validação iterada com divisão aleatória em k-partições.

A validação aleatória é o método mais simples de se dividir os dados e é indicada quando se tem abundância de amostras. Os outros métodos são utilizados quando se tem poucos ou pouquíssimos dados, respectivamente.

Como os dados utilizados em nosso estudo de caso são gerados em uma simulação no Simulink®, isso significa que a quantidade de dados não é um problema, pois podemos gerar quanta informação quisermos. Por isso, utilizamos a validação aleatória em nosso estudo de caso.

Nesse método, o conjunto total de dados disponíveis é dividido aleatoriamente em três subconjuntos:

1) Treino;
2) Validação;
3) Teste.

O subconjunto de treino é utilizado no processo de aprendizagem da RNA. Utilizando esses dados, ela armazena as características do problema em seus parâmetros de rede na esperança de ser capaz de mapear a relação entre as entradas e saídas.

O subconjunto de validação é utilizado na análise da capacidade de generalização da rede que é realizada durante o processo de treinamento. Nessa avaliação, é calculada a métrica de sucesso com a apresentação de um conjunto de dados não utilizado no treinamento, ou seja, dados inéditos que não foram utilizados na atualização de seus parâmetros de rede.

Por fim, o subconjunto de teste é utilizado em uma análise final, após o fim do treinamento, para atestar o potencial de generalização da rede e identificar se houve um vazamento de informação durante o treinamento. Para entender esse fenômeno, precisamos saber como a evolução de um modelo se desdobra através do ajuste dos seus hiperparâmetros.

3.8.3.1 Ajuste dos Hiperparâmetros de um Modelo

No processo de desenvolvimento de uma RNA, é realizado o ajuste de seus hiperparâmetros. Os principais hiperparâmetros de uma RNA são:

1) Quantidade de camadas;
2) Quantidade de neurônios por camada;
3) Função de ativação utilizada;
4) Métrica de sucesso adotada;
5) Função do erro utilizada;
6) Tamanho do lote de treinamento;
7) Quantidade de épocas de treinamento;
8) Taxa de aprendizagem utilizada.

O ajuste desses hiperparâmetros é embasado no desempenho demonstrado pelo modelo em relação ao conjunto de validação. Modifica-se sucessivamente cada um desses hiperparâmetros na esperança de encontrar sucessivas configurações de RNA que modelem o problema de maneira progressivamente melhor. Entretanto, o ajuste dos hiperparâmetros pode resultar no sobreajuste excessivo do modelo ao conjunto de validação, mesmo que a rede nunca tenha sido treinada diretamente com esses dados.

Esse fenômeno é o que Chollet (2018) chama de vazamento de informações. Sempre que um hiperparâmetro da RNA é ajustado com base no desempenho do modelo no conjunto de validação, algumas informações sobre esses dados acabam vazando no modelo.

Portanto, corre-se o risco de o desempenho do modelo ser artificialmente bom nos dados de validação porque, durante o processo de ajuste dos hiperparâmetros, acabamos sobreajustando o modelo aos dados de maneira indireta.

Com isso, percebemos a importância do subconjunto de dados de teste e da necessidade desses dados nunca serem utilizados antes, tanto na validação como no treinamento da RNA. Se algum hiperparâmetro do modelo foi ajustada com base no desempenho da rede em relação ao conjunto de testes, sua medida de generalização estará comprometida.

Com esses três subconjuntos disjuntos, é possível treinar, avaliar e atestar a capacidade de generalização do modelo de maneira confiável.

3.8.4 Realizar o pré-processamento nos dados

O quarto passo é adequar os conjuntos de dados em um formato conveniente no qual eles possam ser inseridos na RNA. Segundo Chollet (2018), para que o processo de treinamento seja otimizado e a rede alcance um resultado mais satisfatório, é recomendado que os dados sejam apresentados da seguinte forma:

1) Agrupados em tensores, ou seja, matrizes n-dimensionais, onde o número de dimensões depende do tipo de dado – séries temporais, imagens, vídeos etc.;
2) É recomendado que os valores contidos nesses tensores estejam em uma escala entre -1 e 1 ou entre 0 e 1;
3) Se os dados forem heterogêneos, ou seja, estiverem em diversas escalas diferentes, então deve-se normalizar os dados, ou seja, transformá-los para uma mesma escala;
4) A depender do problema, é recomendado realizar uma engenharia de recursos nos dados, ou seja, usar o nosso conhecimento sobre os dados e sobre o problema para aplicar transformações codificadas, não aprendidas pela rede, aos dados antes de inseri-los no modelo. O objetivo dessa transformação é facilitar o aprendizado da rede, ou seja, fazer com que ela consiga mapear mais facilmente as entradas e saídas e resolva o problema.

Com exceção da engenharia de recursos, todas essas recomendações foram seguidas, ou seja, todos os dados utilizados foram agrupados em tensores e normalizados para uma escala entre -1 e 1.

3.8.5 Encontrar um Modelo com Poder Estatístico

Com os dados formatados, podemos iniciar o processo de treinamento e refinamento da RNA. Para isso, devemos executar o quinto passo que é definir um ponto de partida determinando uma topologia de rede com poder estatístico. Isso significa definir os hiperparâmetros iniciais da rede que resultem em respostas certas em uma porcentagem melhor que o aleatório. Por exemplo, no caso de uma classificação binária, uma topologia que acerte mais do que 50% dos testes é um bom ponto de partida.

Contudo, Chollet (2018) afirma que nem todo problema é solucionável e se após várias tentativas não for possível encontrar um modelo com poder estatístico, devemos voltar à primeira etapa e refazer os passos iniciais.

Assumindo que as hipóteses feitas na primeira etapa são verdadeiras, ou seja, de que as saídas podem ser previstas com base nas entradas e que os dados disponíveis são suficientemente informativos para que a RNA aprenda o relacionamento entre as entradas e saídas, definimos arbitrariamente alguns dos hiperparâmetros da rede. Em especial, nesse

momento, estamos preocupados principalmente como a quantidade de camadas, a quantidade de neurônios por camada, a função de ativação, a métrica de sucesso, a função do erro e a taxa de aprendizagem.

Nesse momento, o importante é encontrar um modelo com poder estatístico, não havendo a preocupação com os valores desses hiperparâmetros, pois eles serão refinados em um momento posterior. Portanto, ressaltamos que devido à natureza empírica dessa etapa podem existir vários modelos iniciais distintos. Porém, o que nos interessa é apenas definir um ponto de partida inicial encontrando qualquer um deles.

Os hiperparâmetros do modelo inicial foram arbitrariamente determinados e podem ser visto na Tabela 4 abaixo.

Tabela 4 – Hiperparâmetros do Modelo Inicial

Hiperparâmetros	**Valor**
Número de Camadas	1
Neurônios por Camada	45
Épocas de treinamento da etapa não-supervisionada	3
Iterações de retropropagação da etapa supervisionada	40
Quantidade de dados utilizados no treinamento	120s
Tamanho do Lote de Treinamento	16
Dropout	0%
Taxa de aprendizagem da etapa não-supervisionada	0,01
Taxa de aprendizagem da etapa supervisionada	0,01
Função de ativação	Sigmóide

Fonte: Elaborado pelo autor.

3.8.6 Encontrar um Modelo que se Sobreajusta aos Dados

Chollet (2018) afirma que a topologia ideal de uma RNA é aquela que fica bem na fronteira entre o subajuste e o sobreajuste dos dados, ou seja, entre a subcapacidade e a sobrecapacidade de resolver um problema.

Para descobrir onde essa fronteira está localizada, é necessário atravessá-la. Isso significa encontrar um modelo que se sobreajusta aos dados de treinamento. Portanto, o sexto passo é alterar os hiperparâmetros do modelo inicial encontrado na etapa anterior até sobreajustá-lo aos dados. Isso é facilmente alcançado de três formas:

1) Adicionando camadas ao modelo;
2) Aumentando o número de neurônios por camada;
3) Treinando o modelo por mais épocas.

Para determinar se um modelo está sobreajustado, devemos monitorar os erros calculados em relação aos dados de treinamento e aos dados de validação. Evidencia-se o sobreajuste quando o erro referente aos dados de validação aumentar enquanto o erro nos dados de treinamento diminuir, tendendo a zero.

De maneira semelhante a etapa anterior, não há uma fórmula fechada para encontrar um modelo sobreajustado e sua obtenção é totalmente empírica. Portanto, existem infinitas configurações de redes sobreajustadas para um mesmo problema, mas qualquer uma delas é suficiente para prosseguirmos para a próxima etapa.

Veremos mais detalhes sobre essa etapa na seção 3.9, onde abordamos sobre o ajuste dos hiperparâmetros do modelo.

3.8.7 Regular o Modelo Através do Ajuste de seus Hiperparâmetros

A sétima e última etapa consiste na regulagem do modelo através do ajuste fino de seus hiperparâmetros objetivando chegar o mais próximo possível do modelo ideal.

Essa etapa é a mais demorada e consiste em realizar pequenas modificações no modelo, seguidas de seu treinamento e avaliação do desempenho em relação aos dados de validação. Esse processo é repetido até que o desempenho da RNA seja tão bom quanto for possível de se obter.

Para alcançar o objetivo desejado, Chollet (2018) recomenda algumas ações que devem ser realizadas empiricamente:

1) Adicionar regularização L1 ou L2;
2) Adicionar camadas de *Dropout*;
3) Adicionar ou remover camadas;
4) Alterar a quantidade de neurônios por camada;
5) Mudar a função de ativação;
6) Alterar a taxa de aprendizagem;
7) Tentar diferentes engenharias de recursos nos dados;
8) Testar diferentes configurações para os outros hiperparâmetros.

Todas essas ações têm como objetivo reduzir gradativamente o sobreajuste aos dados de treinamento para alcançar uma capacidade de generalização maior.

Dessas ações, não utilizaremos a regularização L1 ou L2. Portanto, resta discutir apenas a adição de camadas de *Dropout*.

A adição de camadas de *Dropout* faz com que uma parte das saídas da camada imediatamente anterior a ela seja aleatoriamente definida como zero, independentemente de suas entradas. Essa porcentagem é definida pelo usuário e ocorre durante o processo de treinamento.

A ideia por trás disso é introduzir ruídos nos valores de saída de uma camada para quebrar a detecção de padrões fracos e não significativos que a RNA aprenderia se não houvesse essa intervenção. Sem essa interferência proposital, o desempenho da rede é prejudicado, pois o mapeamento entre as entradas e saídas contém, muito provavelmente, relações irrelevantes que atrapalham na resolução do problema.

A próxima subseção se dedica a apresentação dos principais resultados do processo de ajuste dos hiperparâmetros da RNA. Apresentaremos a evolução da topologia da RNA desde seu modelo inicial até a sua forma final.

3.9 Ajuste dos Hiperparâmetros do Modelo

Essa subseção se concentra na apresentação do processo de ajuste dos hiperparâmetros da rede a fim de encontrarmos uma topologia que resolva o problema de forma adequada.

Para determinação dos hiperparâmetros do modelo final, testamos aproximadamente 2500 topologias diferentes. Em cada um desses testes, variou-se um dos hiperparâmetro por vez, mantendo os outros constantes.

Em seguida, treinou-se a RNA com o conjunto de dados de treino e se calculou o MSE (*Mean Square Error* ou o Erro Quadrático Médio) das saídas com o conjunto de dados de teste.

Os hiperparâmetros variados e testados foram:

1) O número de camadas;
2) A quantidade de neurônios por camada;
3) O número de épocas de treinamento da etapa não-supervisionada (pré-treino);

4) O número de iterações de retropropagação da etapa supervisionada (ajuste-fino);
5) O tamanho do lote de treinamento;
6) A quantidade de dados utilizados no treinamento;
7) As taxas de *Dropout*.

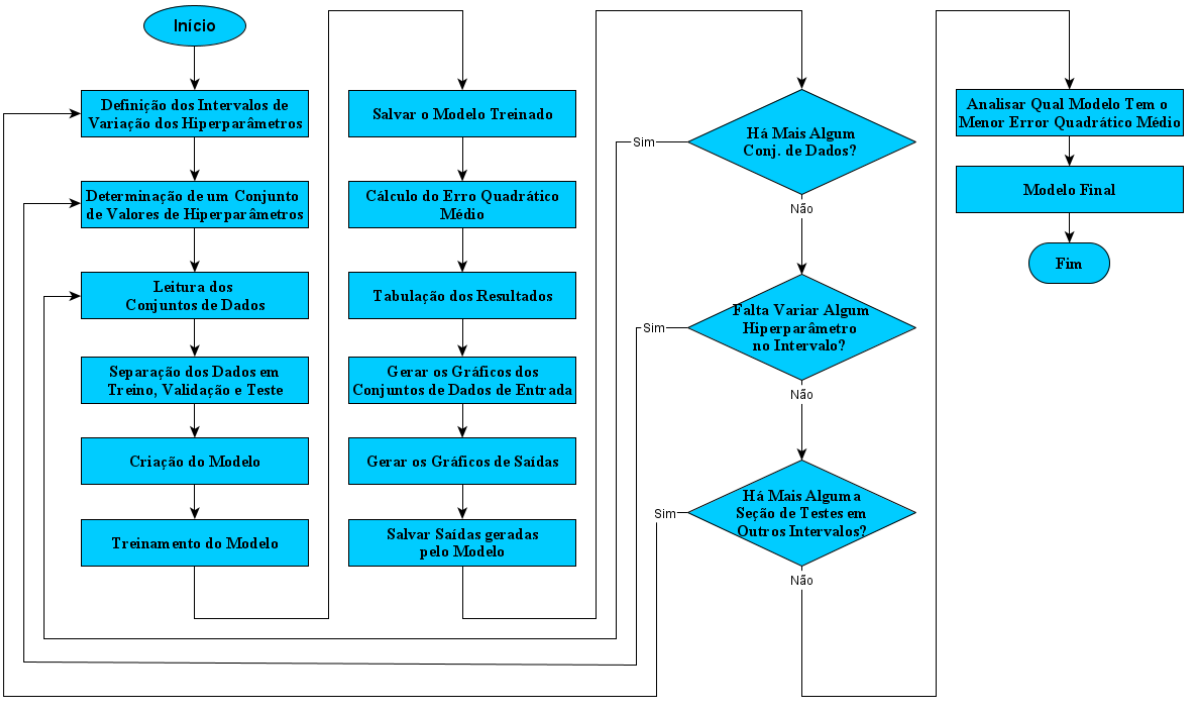
Foram realizados, ao todo, 16 seções de testes dos hiperparâmetros. Cada uma dessas seções testavam um subconjunto de possibilidades para esses hiperparâmetros.

O computador utilizado nesse processo foi um ASUS K45VM com um processador Intel Core I7-3610QM de 2,30 GHz de 8 núcleos, 8 GB de memória RAM, HD SATA de 1 TB de armazenamento e placa de vídeo Nvidia GeForce GT 630M de 2 GB de memória dedicada.

De maneira geral, as primeiras seções de testes foram as mais demoradas, levando até 32 horas ininterruptas para serem concluídas. Porém, ao passo que os testes se sucediam, sinais de quais direções seguir se evidenciavam e os valores dos hiperparâmetros afunilavam, os tempos de teste se tornaram gradualmente menores. Por exemplo, a última seção de testes durou aproximadamente 1 hora.

A Figura 22 abaixo apresenta um fluxograma do algoritmo utilizado no processo de ajuste dos hiperparâmetros.

Figura 22 – Fluxograma do Algoritmo de Ajuste dos Hiperparâmetros



Fonte: Elaborado pelo autor.

Nos próximos parágrafos, apresentaremos os principais resultados alcançados em alguns desses testes.

Inicialmente, foram realizadas todas as combinações dos seguintes hiperparâmetros apresentados na Tabela 5 para que pudéssemos ter um noção de onde começar.

Tabela 5 – Combinações Testadas dos Hiperparâmetros do Modelo do CN (A)

Hiperparâmetros	**Valor**
Número de Camadas	1, 2 e 3
Neurônios por Camada	45, 60, 75 e 90
Épocas de treinamento da etapa não-supervisionada	3, 5 e 7
Iterações de retropropagação da etapa supervisionada	40, 50 e 60
Quantidade de dados utilizados no treinamento	120s
Tamanho do Lote de Treinamento	16
Dropout	0%
Taxa de aprendizagem da etapa não-supervisionada	0,01
Taxa de aprendizagem da etapa supervisionada	0,01
Função de ativação	Sigmóide

Fonte: Elaborado pelo autor.

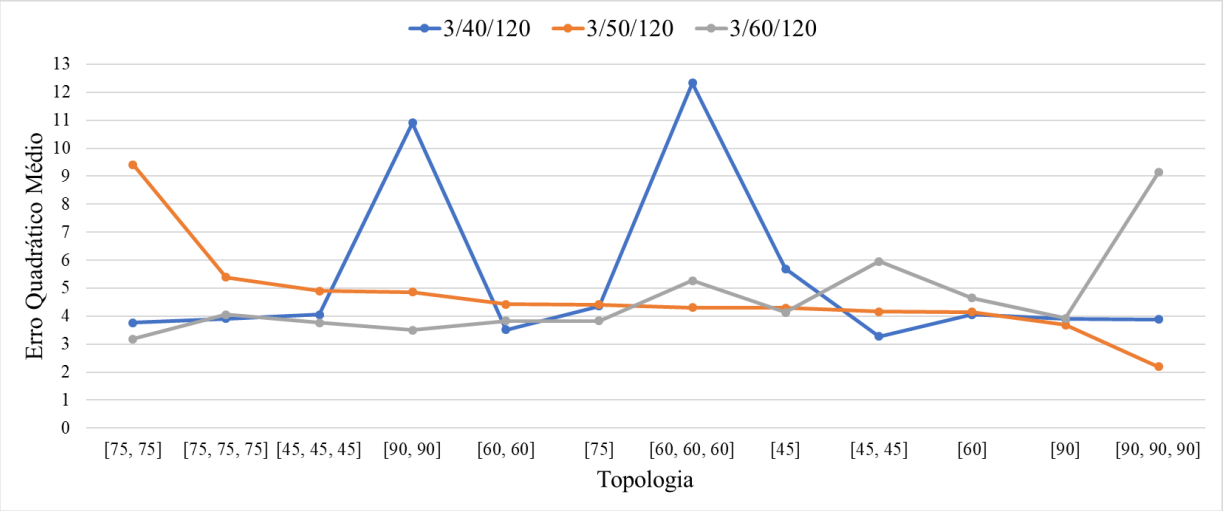
O Gráfico 10 abaixo mostra o erro quadrático médio da resposta do modelo para um subconjunto das topologias testadas. No eixo-x, cada conjunto de número entre colchetes representa um modelo testado, onde a quantidade de números significa a quantidade de camadas e cada número em si é o número de neurônios da respectiva camada. Por exemplo, [45] é uma topologia com uma camada de 45 neurônios. Já [75, 75, 75] é uma outra topologia com três camadas de 75 neurônios cada.

Cada série de dados é identificado por 3 números separados por uma barra "/". O primeiro número desse identificador diz respeito ao número de épocas de treinamento da etapa não-supervisionada. O segundo é o número de iterações de retropropagação da etapa supervisionada. O terceiro é quantidade de dados utilizados no treinamento, em segundos.

No Gráfico 10, enquanto variou-se o número de camadas, a quantidade de neurônios por camadas e o número de iterações de retropropagação da etapa supervisionada, manteve-se constante o número de épocas de treinamento da etapa não-supervisionada em 3 e a quantidade de dados utilizados no treinamento em 120s, além de todos os outros

hiperparâmetros. O menor erro quadrático médio foi de 2,1883 V^2 para a topologia [90, 90, 90] 3/50/120.

Gráfico 10 – Erro Quadrático Médio em Função da Topologia (A)

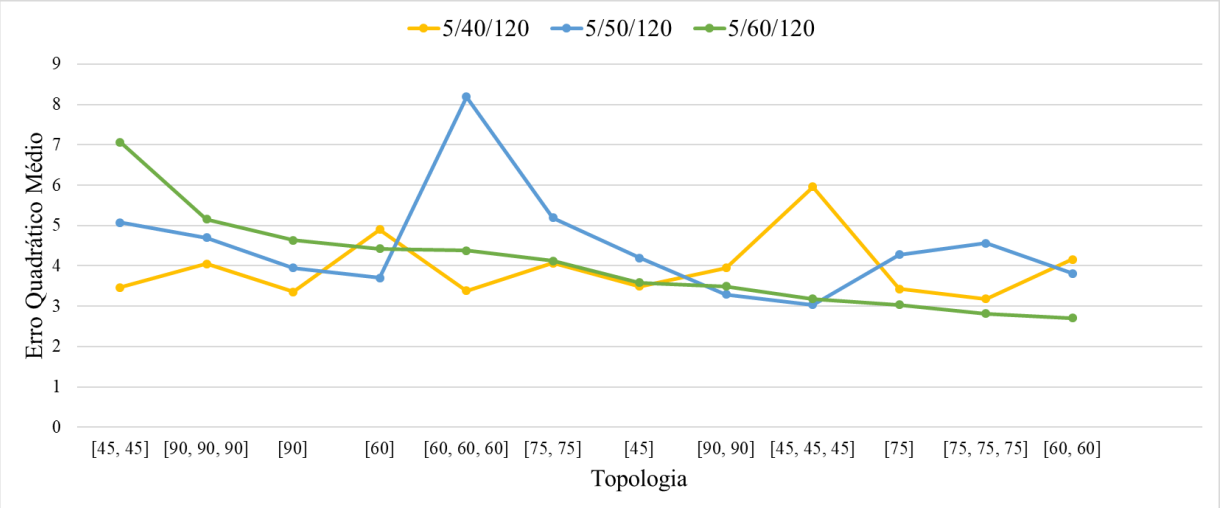


Fonte: Elaborado pelo autor.

O Gráfico 11 é semelhante ao anterior, entretanto o número de épocas de treinamento da etapa não-supervisionada é 5. O menor erro quadrático médio foi de 2,7056 V^2 para a topologia [60, 60] 5/60/120.

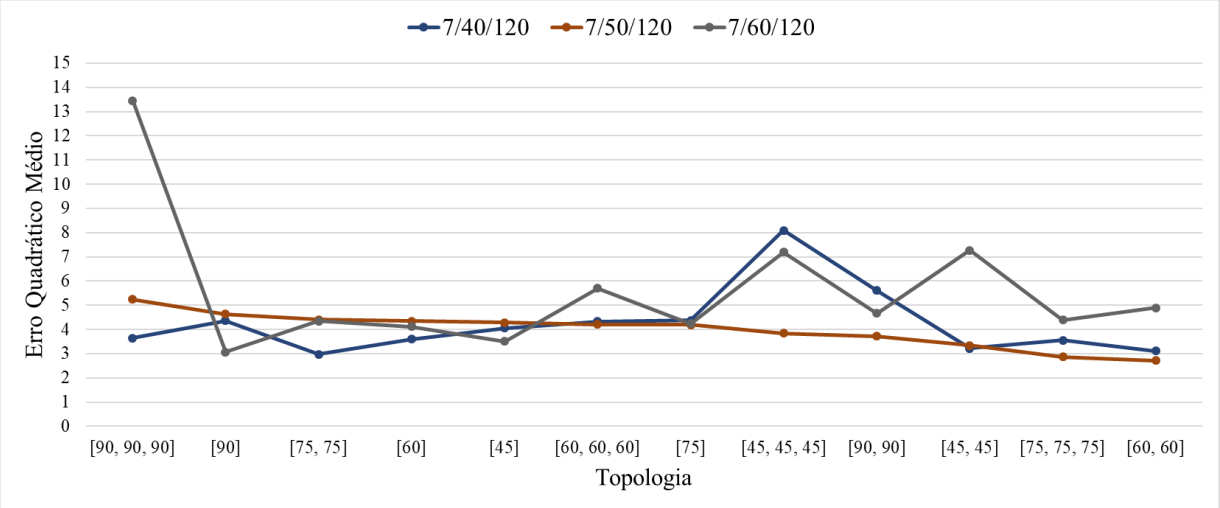
No Gráfico 12, o número de épocas de treinamento da etapa não-supervisionada é 7. O menor erro quadrático médio foi de 2,7113 V^2 para a topologia [60, 60] 7/50/120.

Gráfico 11 – Erro Quadrático Médio em Função da Topologia (B)



Fonte: Elaborado pelo autor.

Gráfico 12 – Erro Quadrático Médio em Função da Topologia (C)



Fonte: Elaborado pelo autor.

Não foi possível estabelecer uma correlação entre a quantidade de camadas, o número de neurônios por camada e o erro quadrático médio. Entretanto, observou-se que o menor erro ocorreu na topologia [90, 90, 90] 3/50/120 – três camadas de 90 neurônios cada com 3 épocas de treinamento da etapa não-supervisionada, 50 iterações de retropropagação na etapa supervisionada e 120 segundos de dados de treinamento.

Tabela 6 – Combinações Testadas dos Hiperparâmetros do Modelo do CN (B)

Hiperparâmetros	**Valor**
Número de Camadas	3
Neurônios por Camada	90
Épocas de treinamento da etapa não-supervisionada	3, 5 e 7
Iterações de retropropagação da etapa supervisionada	40, 50 e 60
Quantidade de dados utilizados no treinamento	120s, 180s e 240s
Tamanho do Lote de Treinamento	16
Dropout	0%
Taxa de aprendizagem da etapa não-supervisionada	0,01
Taxa de aprendizagem da etapa supervisionada	0,01
Função de ativação	Sigmóide

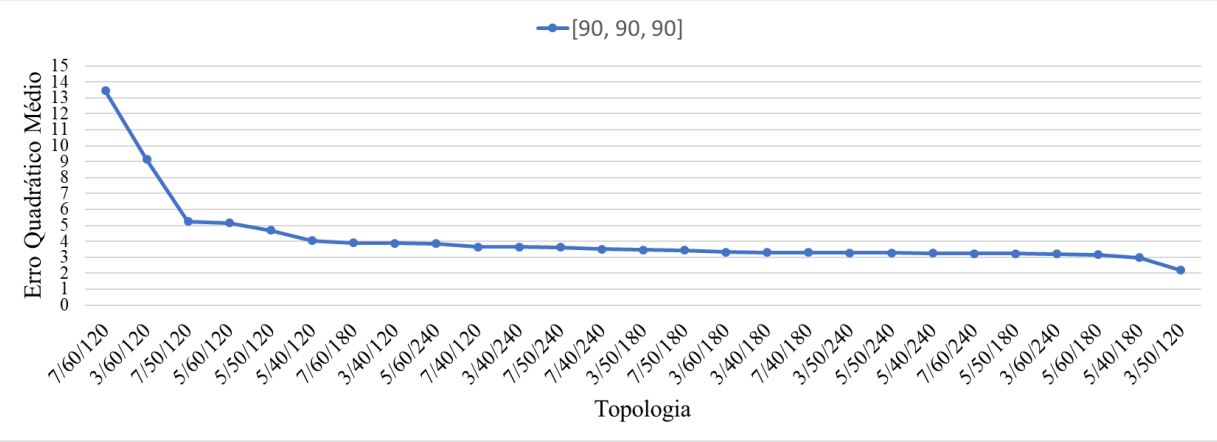
Fonte: Elaborado pelo autor.

Resolveu-se explorar mais a topologia [90, 90, 90] em outras seções de testes. Iniciamos com uma análise do impacto do volume de dados de treino na eficiência da RNA.

Para isso, variou-se o número de épocas de treinamento da etapa não-supervisionada, o número de iterações de retropropagação da etapa supervisionada e a quantidade de dados utilizados no treinamento. Todos os outros hiperparâmetros foram mantidos constantes. O conjunto de hiperparâmetros testados nessa seção pode ser visto na Tabela 6 acima.

No Gráfico 13, pode-se visualizar o erro quadrático médio para todas as variações da topologia [90, 90, 90] que foram testadas nessa seção. Podemos ver que o menor erro quadrático médio foi de 2,1883 V^2 para a topologia [90, 90, 90] 3/50/120 e percebemos que o aumento do volume de dados de treino não contribuiu para a diminuição do erro quadrático médio da RNA.

Gráfico 13 – Erro Quadrático Médio em Função da Topologia (D)



Fonte: Elaborado pelo autor.

Tabela 7 – Combinações Testadas dos Hiperparâmetros do Modelo do CN (C)

Hiperparâmetros	**Valor**
Número de Camadas	1, 2 e 3
Neurônios por Camada	90, 120 e 150
Épocas de treinamento da etapa não-supervisionada	1, 2, 3, 4, e 5
Iterações de retropropagação da etapa supervisionada	50, 55 e 60
Quantidade de dados utilizados no treinamento	120s e 180s
Tamanho do Lote de Treinamento	16
Dropout	0%
Taxa de aprendizagem da etapa não-supervisionada	0,01
Taxa de aprendizagem da etapa supervisionada	0,01
Função de ativação	Sigmóide

Fonte: Elaborado pelo autor.

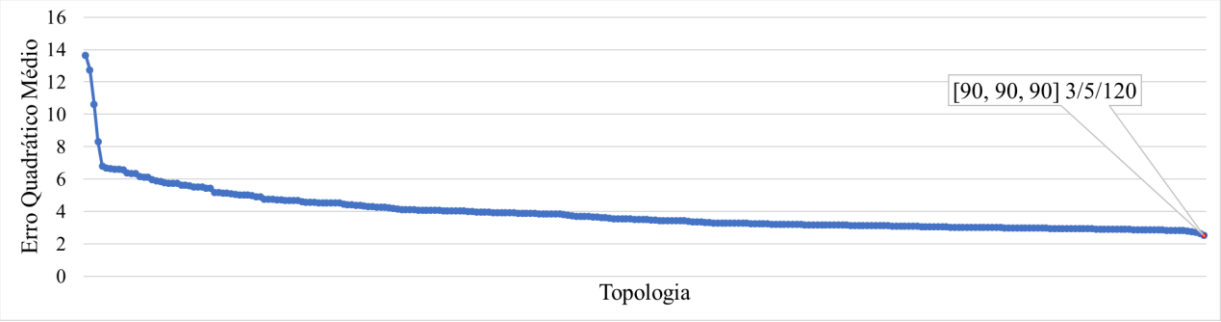
Em seguida, realizou-se uma outra seção de testes, variando a topologia [90, 90, 90] mais uma vez, a fim de descobrir se mais algum refinamento em seus hiperparâmetros poderia ser feito. Nessa seção de testes, foram realizadas todas as combinações dos hiperparâmetros apresentados na Tabela 7 acima.

Fizemos algumas análises com as combinações dos hiperparâmetros acima de forma similar à seção de testes anterior. Nossa intenção foi verificar os impactos gerados pelo aumento da quantidade de neurônios por camada e pela redução da quantidade de épocas de treinamento da etapa não-supervisionada, além de tentar refinar o número de iterações de retropropagação da etapa supervisionada.

Constatamos que o aumento da quantidade de neurônios por camada não influenciou positivamente no aprendizado da rede, levando, em alguns casos, a um erro quadrático médio maior. Além disso, a redução da quantidade de épocas de treinamento da etapa não-supervisionada levou a modelos com maiores MSE. Por fim, verificamos que a variação do número de iterações de retropropagação da etapa supervisionada não levou a resultados melhores.

O Gráfico 14 consiste em todos MSE calculados para todas as combinações dos hiperparâmetros da Tabela 7. A topologia que apresentou o menor MSE foi a [90, 90, 90] 3/50/120, já encontrada anteriormente.

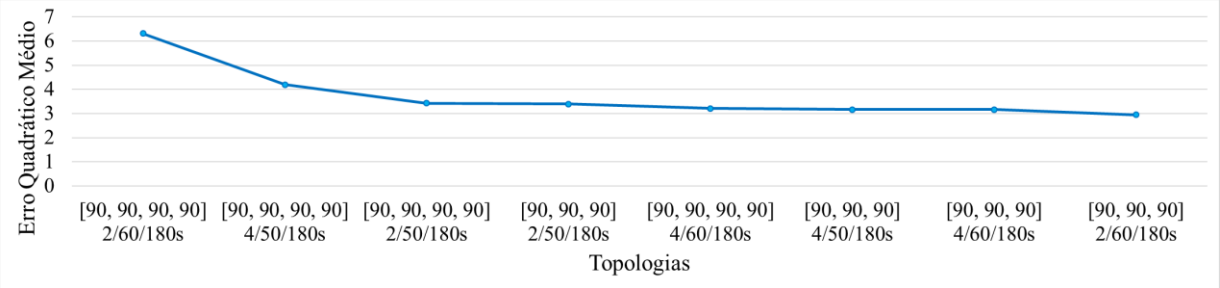
Gráfico 14 – Erro Quadrático Médio em Função da Topologia (E)



Fonte: Elaborado pelo autor.

Em seguida, realizamos uma outra seção de testes para determinar se uma camada a mais de 90 neurônios melhoraria os resultados da rede. Conforme indicado pelo Gráfico 15, constatamos que o aumento da quantidade de camadas não influenciou positivamente no aprendizado da rede, levando, em alguns casos, a um erro maior do que sua versão com 3 camadas.

Gráfico 15 – Erro Quadrático Médio em Função da Topologia (F)

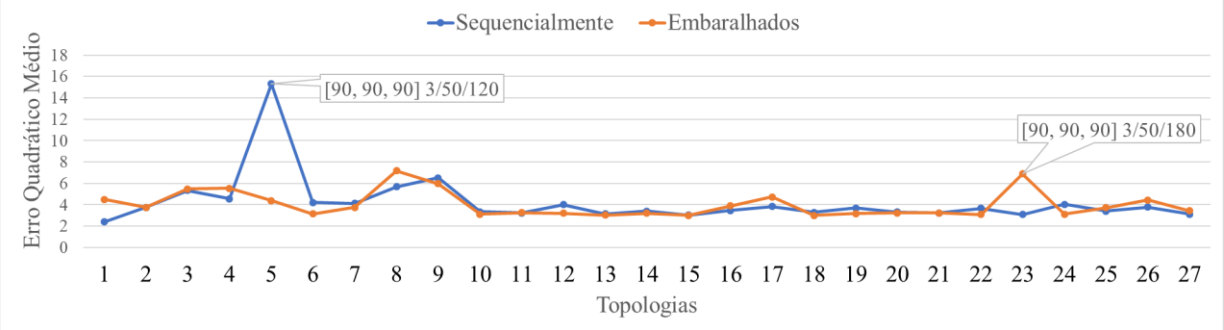


Fonte: Elaborado pelo autor.

Também foram conduzidas duas seções de testes para verificar o impacto da ordem de apresentação dos dados no treinamento da rede. Em uma das seções, os dados foram apresentados sequencialmente, da mesma forma como foram gerados no MATLAB®. Na outra, os dados foram embaralhados e utilizados de forma aleatória.

Conforme pode ser visto no Gráfico 16, com exceção das topologias [90, 90, 90] 3/50/120 e [90, 90, 90] 3/50/180, os erros quadráticos médios foram praticamente os mesmos, independentemente da ordem de apresentação dos dados. Os motivos para a existência das diferenças nos pontos destacados são desconhecidos.

Gráfico 16 – Erro Quadrático Médio em Função da Topologia (G)



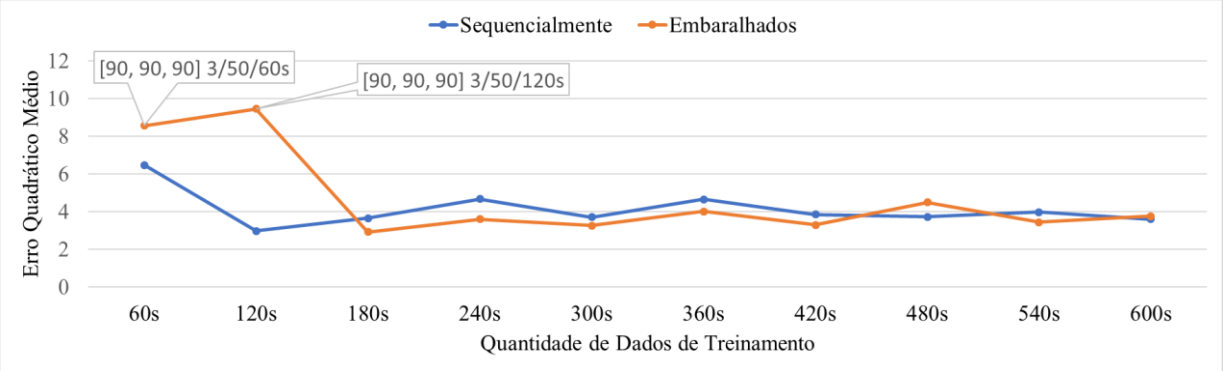
Fonte: Elaborado pelo autor.

Em outras duas seções de testes, variou-se o tamanho dos conjuntos de dados de treinamento, apresentando-os ora sequencialmente como foram gerados no MATLAB® e ora embaralhados. Nesses testes, a topologia foi mantida constante com 3 camadas de 90 neurônios, 3 épocas de treinamento da etapa não-supervisionada e 50 iterações de retropropagação da etapa supervisionada. Todos os outros hiperparâmetros foram os mesmos utilizados anteriormente.

Conforme vemos no Gráfico 17, com exceção da topologia [90, 90, 90] 3/50/60s e [90, 90, 90] 3/50/120s, os erros quadráticos médios foram praticamente os mesmos,

independentemente da ordem de apresentação dos dados. Além disso, o aumento da quantidade de dados de treino não levou a uma redução do MSE. Os motivos para a existência das diferenças nos pontos destacados são desconhecidos.

Gráfico 17 – Erro Quadrático Médio em Função da Quantidade de Dados de Treinamento

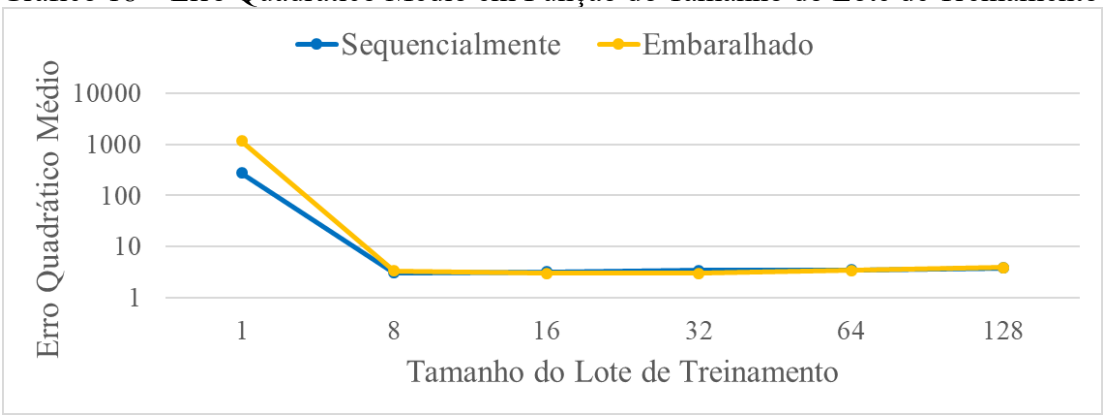


Fonte: Elaborado pelo autor.

Em outras duas seções de testes, variou-se o tamanho dos lotes de treinamento entre 1, 8, 16, 32, 64 e 128 dados por lote. Esses lotes foram gerados ora por dados apresentados sequencialmente e ora embaralhados. A topologia foi mantida constante com 3 camadas de 90 neurônios, 3 épocas de treinamento da etapa não-supervisionada e 50 iterações de retropropagação da etapa supervisionada. Todos os outros hiperparâmetros foram os mesmo já apresentados anteriormente.

Analisando o Gráfico 18, com exceção do caso de 1 dado por lote, vemos que os erros quadráticos médios foram basicamente os mesmos, independentemente da ordem de apresentação dos dados e do tamanho dos lotes. Os motivos para a existência da diferença no ponto destacado são desconhecidos.

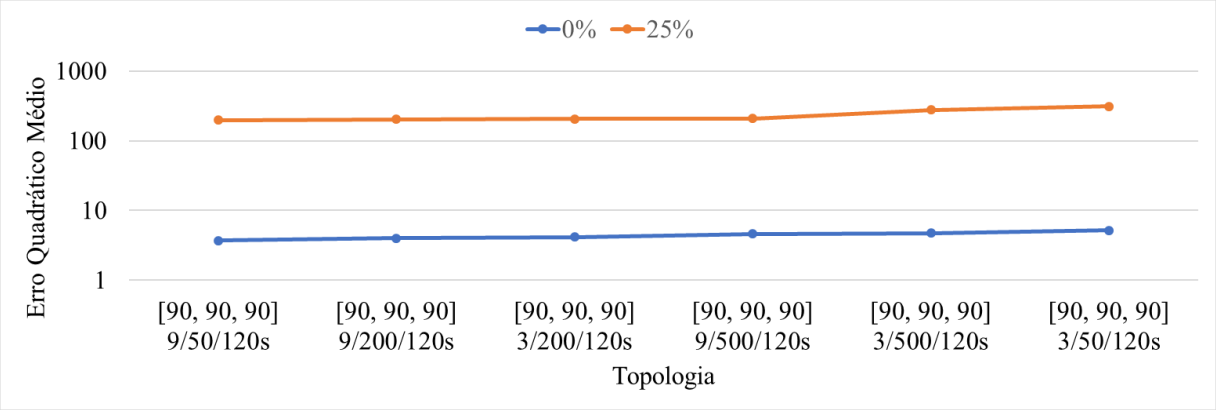
Gráfico 18 – Erro Quadrático Médio em Função do Tamanho do Lote de Treinamento



Fonte: Elaborado pelo autor.

Finalmente, avaliou-se o impacto da utilização de *Dropout* no desempenho da rede. Testou-se taxas de 0% e 25% de *Dropout* em uma rede de 3 camadas de 90 neurônios por camada com 120 segundos de dados de treino apresentados em lotes de 16 dados.

Gráfico 19 – Influência do *Dropout*



Fonte: Elaborado pelo autor.

Nessa seção de testes, também foi analisado se o aumento do número de épocas de treinamento da etapa não-supervisionada para 9 ou o aumento do número de iterações de retropropagação para 200 ou 500, combinados com o *Dropout,* resultaria em um desempenho melhor da rede. Todos os outros hiperparâmetros foram mantidos constantes.

Analisando o Gráfico 19, tem-se as expectativas contrariadas, pois percebemos que a adição de *Dropout* foi prejudicial para o desempenho da rede. Os menores MSE foram obtidos pelas topologias sem *Dropout*. Além disso, o aumento do número de épocas de treinamento da etapa não-supervisionada e o aumento do número de iterações de retropropagação não interferiram de forma expressiva no desempenho da rede.

Finalizamos aqui as discussões das principais análises topológicas realizadas. A próxima subseção apresenta um resumo da topologia final do modelo.

3.10 Modelo da RNA para Controle de Velocidade

Diante dos resultados expostos na subseção anterior, fomos capazes de determinar os hiperparâmetros de um modelo com um MSE de 2,1883 V^2. Ressaltamos que, por conta desse modelo ter sido encontrado empiricamente, ele não consiste em uma solução ótima

global. Entretanto, como veremos mais adiante, suas características de respostas estão bastante adequadas. Os principais hiperparâmetros desse modelo estão listados na Tabela 8.

Tabela 8 – Hiperparâmetros do Modelo do CN

Hiperparâmetros	**Valor**
Número de Camadas	3
Neurônios por Camada	90
Épocas de treinamento da etapa não-supervisionada	3
Iterações de retropropagação da etapa supervisionada	50
Quantidade de dados utilizados no treinamento	120s
Tamanho do Lote de Treinamento	16
Dropout	0%
Taxa de aprendizagem da etapa não-supervisionada	0,01
Taxa de aprendizagem da etapa supervisionada	0,01
Função de ativação	Sigmóide

Fonte: Elaborado pelo autor.

Na próxima subseção, analisaremos a eficiência e eficácia do modelo proposto, ou seja, vamos verificar seu comportamento transitório e em regime permanente. A fim de validar o modelo, realizaremos uma comparação entre o CN e o controlador PID, analisando alguns parâmetros de respostas clássicos da teoria de controle.

3.11 Análise das Respostas do Controlador Neural

Vamos dividir nossa análise das respostas do CN em duas partes. A primeira avaliará o modelo utilizando uma entrada degrau. Na segunda parte, apresentaremos alguns resultados do modelo utilizando entradas mais complexas – os dados de treino e teste.

3.11.1 Respostas para a Entrada Degrau

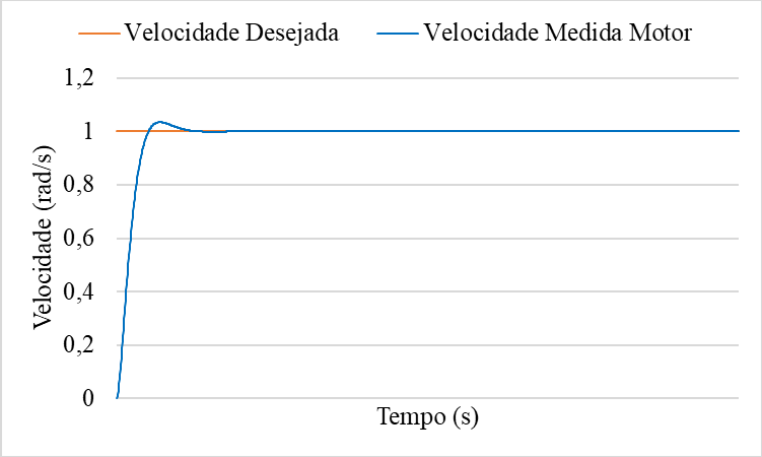
Vimos anteriormente que o desempenho de um controlador PID é analisado com base na sua resposta transitória e de regime permanente a uma entrada degrau unitário.

Portanto, o primeiro passo da análise das respostas do CN foi gerar um conjunto de dados que representasse uma entrada degrau unitário para ser inserido na RNA. Esses dados

foram gerados na *toolbox* criada por Turevskiy (2016) de forma semelhante aos dados de treinos e testes, já abordados anteriormente. A diferença é que retiramos o bloco *Uniform Random Number* que definia valores aleatórios de velocidade desejada e colocamos um bloco *step* com um valor de saída constante em 1.

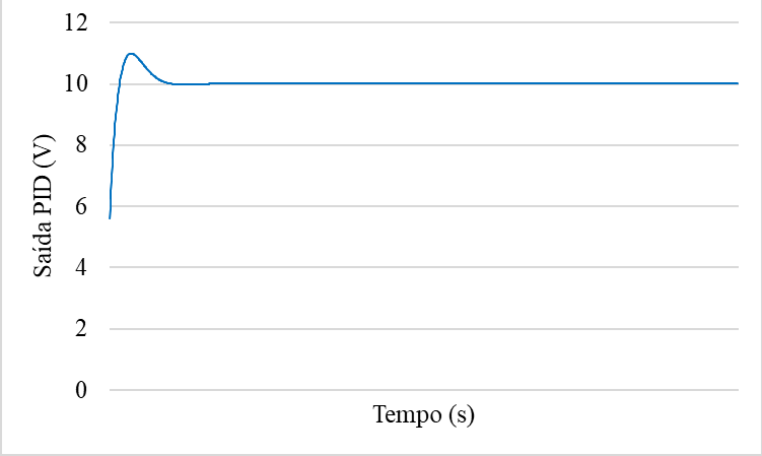
A entrada degrau unitário do controlador PID pode ser visto no Gráfico 20 abaixo e o Gráfico 21 mostra sua resposta a essa entrada.

Gráfico 20 – Entrada Degrau Unitário do Controlador PID



Fonte: Elaborado pelo autor.

Gráfico 21 – Resposta do Controlador PID ao Degrau Unitário



Fonte: Elaborado pelo autor.

Já comentamos que a velocidade do motor CC simulado varia arbitrariamente entre -5 e 5 rad/s e que para uma melhor aprendizagem da rede, seus dados de entrada devem ser normalizados para valores compreendidos entre -1 e 1.

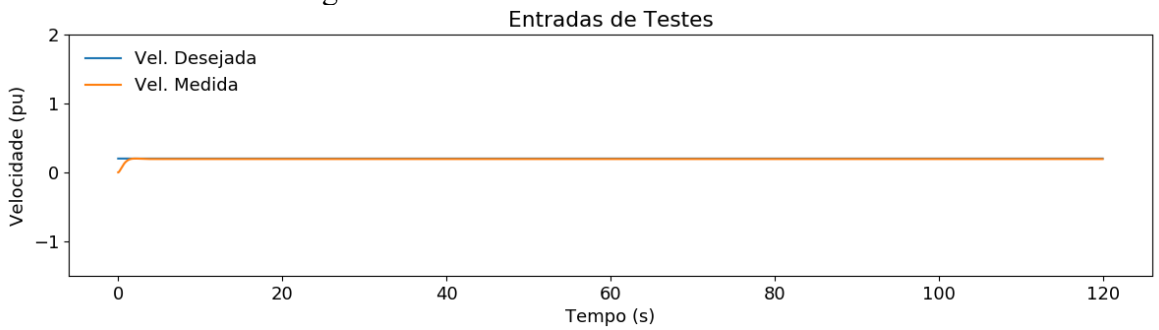
Logo, se inserirmos a função degrau unitária do Gráfico 20 na entrada da rede, na verdade estaríamos inserindo uma função degrau referente a 5 rad/s e não a 1 rad/s. Isso ocorre

devido a normalização que foi feita no pré-processamento dos dados para realização do treinamento da rede.

Dessa forma, não poderíamos comparar os resultados do comportamento do controlador PID com as respostas do CN utilizando a mesma função degrau unitário, já que ela é vista com uma magnitude diferente em cada controlador, ou seja: 1 rad/s no PID e 5 rad/s no CN.

Portanto, como queremos comparar o comportamento transitório e o comportamento em regime permanente em ambos os controladores, a entrada degrau utilizada no CN deve possuir um valor de 0,2 rad/s. Assim, essa função degrau de entrada será vista com uma magnitude de 1 rad/s no CN e poderemos comparar seus resultados com os do controlador PID. A entrada degrau do CN pode ser visto no Gráfico 22.

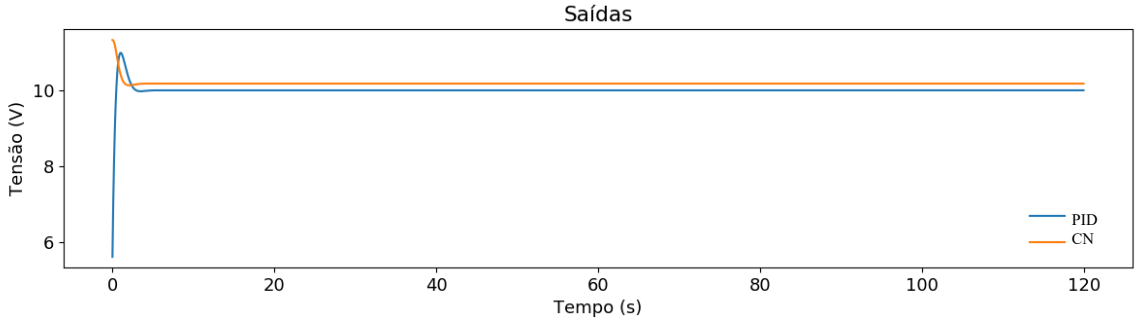
Gráfico 22 – Entrada Degrau do CN



Fonte: Elaborado pelo autor.

Esses dados foram inseridos no modelo treinado apresentado na seção anterior e, com isso, conseguimos gerar sua resposta ao degrau unitário que pode ser visualizada e comparada com a resposta do controlador PID no Gráfico 23. Nesse gráfico, a saída do PID está sobreposta a saída do CN.

Gráfico 23 – Respostas do CN e PID a Entrada Degrau



Fonte: Elaborado pelo autor.

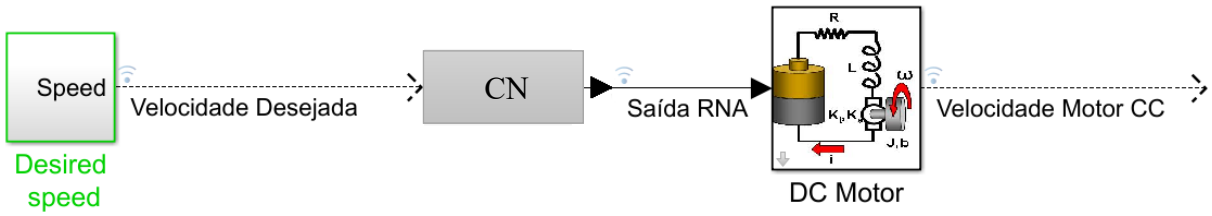
Reforçamos que as saídas, tanto do controlador PID como do CN, consistem em curvas que representam o nível de tensão que deve ser colocado nos terminais do motor em cada instante t para controlar sua rotação e alcançar a velocidade desejada.

Analisando o Gráfico 23, percebemos que as saídas de tensão do PID e do CN não são exatas, o que é um resultado esperado. Calculamos um erro quadrático médio de apenas 0,0778 V^2 e um erro estacionário absoluto de 0,1774 V entre as duas respostas. Além disso, calcularmos um erro estacionário E_s de apenas -1,74%. Portanto, percebemos que a saída gerada pela RNA está bastante próxima da saída do PID.

Com a saída da RNA determinada, o próximo passo foi controlar o motor CC com o CN e obter sua curva de velocidade para que pudéssemos compará-la com a curva de velocidade do motor controlado pelo PID, ambos submetidos as suas respectivas entradas degrau.

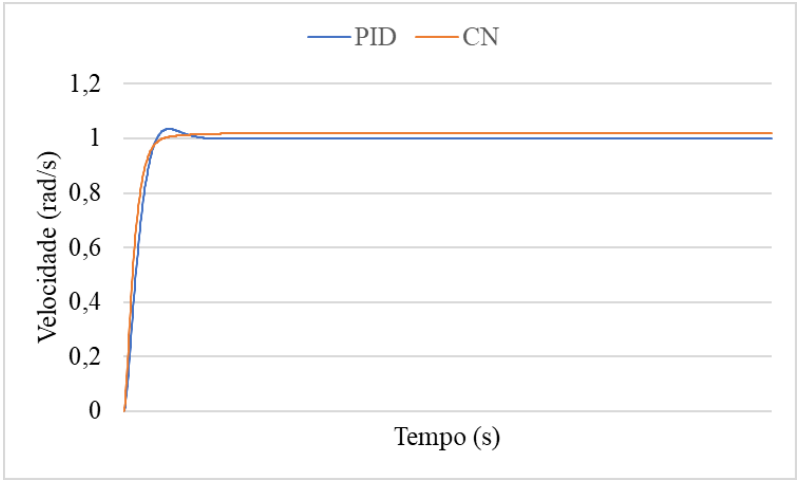
Para isso, alteramos a *toolbox* criada por Turevskiy (2016) e removemos o controlador PID, colocando no lugar a saída do CN, conforme a Figura 23. O Gráfico 24 mostra a curva de velocidade do motor controlado pelo PID sobreposta a curva de velocidade do motor controlado pelo CN.

Figura 23 – *Toolbox* Alterada para o que CN Controle o Motor CC



Fonte: Elaborado pelo autor.

Gráfico 24 – Curvas de Velocidade do PID e CN – Entrada Degrau

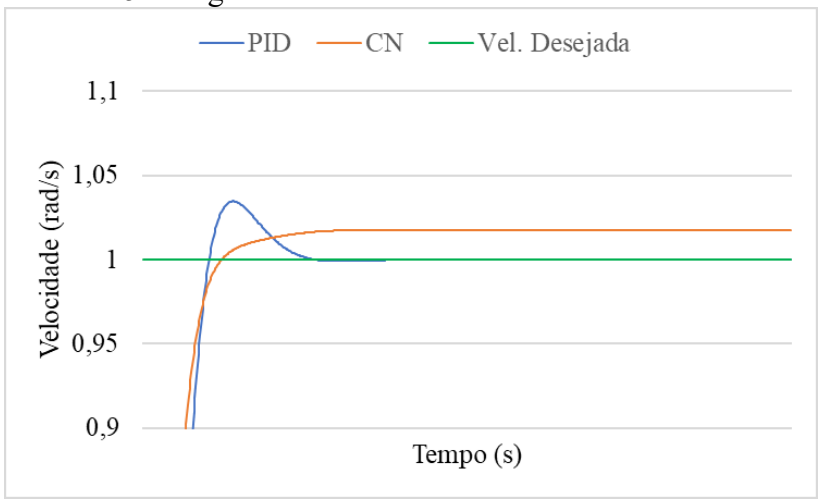


Fonte: Elaborado pelo autor.

Analisando o Gráfico 24, percebemos que o perfil de velocidade do motor controlado pelo PID e pelo CN obviamente não são iguais. Entretanto, calculamos um erro quadrático médio de apenas 0,0004 rad^2/s^2 e um erro estacionário absoluto de 0,0177 rad/s entre as duas respostas. Além disso, calculamos um erro estacionário E_s de apenas -1,74%. Portanto, percebemos que o perfil de velocidade do motor CC controlado pela RNA está bastante próximo a sua contrapartida proveniente do controle do PID.

O Gráfico 25 abaixo consiste em uma ampliação em uma região de interesse das curvas de velocidade do controlador PID e do CN. Analisando esse gráfico, podemos percebemos visualmente que o CN apresentar um erro estacionário maior do que o controlador PID. Em contrapartida, o CN apresenta um sobressinal máximo menor em relação ao controlador PID.

Gráfico 25 – Região de Interesse das Curvas de Velocidade – Entrada Degrau



Fonte: Elaborado pelo autor.

Finalmente, calculamos alguns parâmetros clássicos da teoria de controle que são utilizados para avaliar as características da resposta transitória e de regime permanente do sistema. Os valores desses parâmetros, calculados tanto para o controlador PID como para o CN, estão apresentados na Tabela 9. A última linha dessa tabela contém um valor percentual que indica o quão menor (valores negativos) ou o quão maior (valores positivos) cada um desses parâmetro de resposta do CN é em relação ao controlador PID.

Tabela 9 – Parâmetros da Resposta Transitória e de Regime Permanente do Sistema

	t_d (s)	t_r (s)	t_p (s)	M_p (%)	t_s (s)	E_s (%)
PID	0,56	1,54	2,06	3,47	2,74	0
CN	0,40	1,82	5,62	1,78	1,44	-1,74
Δ	-28,57%	18,18%	172,82%	-48,70%	-47,44%	∞

Fonte: Elaborado pelo autor.

Analisando esses resultados, percebemos que o tempo de atraso t_d é 28,57% menor no CN, o que significa que o CN leva menos tempo para alcançar 50% de seu valor final pela primeira vez. Entretanto, o tempo de subida t_r, que diz respeito ao tempo necessário para que a resposta alcance 100% de seu valor final pela primeira vez é 18,18% maior no CN. Isso significa que, nos instantes iniciais do transitório, o CN tende mais rapidamente para a resposta do que o controlador PID. Entretanto, nos instantes posteriores do transitório, esse comportamento se inverte e o CN tende a ser mais lento.

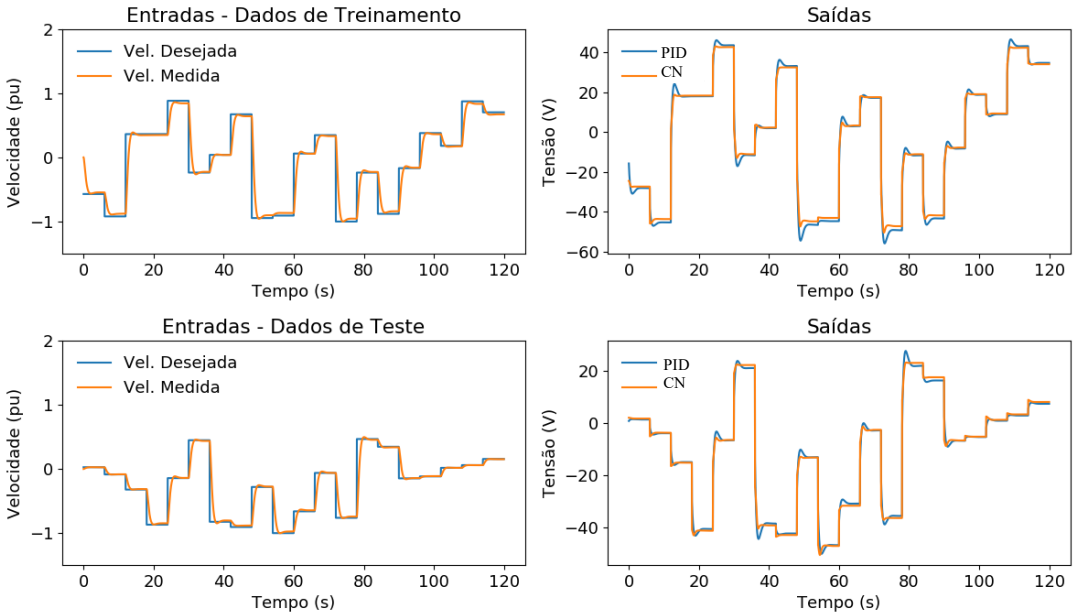
Além disso, o sobressinal máximo M_p, que é o valor máximo da curva da resposta, e o tempo de acomodação t_s, que é o tempo necessário para que a curva de resposta alcance valores em uma faixa de 2% em torno do valor final, são, respectivamente, 48,70% e 47,44% menores no CN, o que significa que o CN oscila menos em relação ao PID, tanto em magnitude do valor do sinal, como em duração de tempo de oscilação. O tempo de pico t_p, que é o tempo necessário para que a resposta atinja o primeiro pico de sobressinal, é 172,82% maior no CN. Entretanto, visto o resultados de M_p e t_s, esse valor não tem importância.

Contudo, o erro estacionário E_s, que é a diferença percentual entre o estado desejado para o sistema e o estado efetivo do sistema em regime permanente, é de -1,74% no CN e 0% no controlador PID. Isso significa que o CN inseriu um erro em regime permanente na velocidade do motor CC, enquanto o controlador PID alcançou a velocidade desejada exata.

3.11.2 Respostas para os Dados de Treinamento e Testes

Os conjuntos de dados de entradas utilizados para treinar e testar o modelo final podem ser vistos na coluna esquerda do Gráfico 26. Esses dados foram inseridos no modelo final já treinado para que pudéssemos gerar as respostas da rede para cada um desses conjuntos. As respectivas saídas podem ser vistas na coluna direita desse gráfico. Com isso, podemos comparar visualmente as respostas do controlador PID com as do CN.

Gráfico 26 – Entradas e Saídas dos Conjuntos de Dados de Treinamento e Teste



Fonte: Elaborado pelo autor.

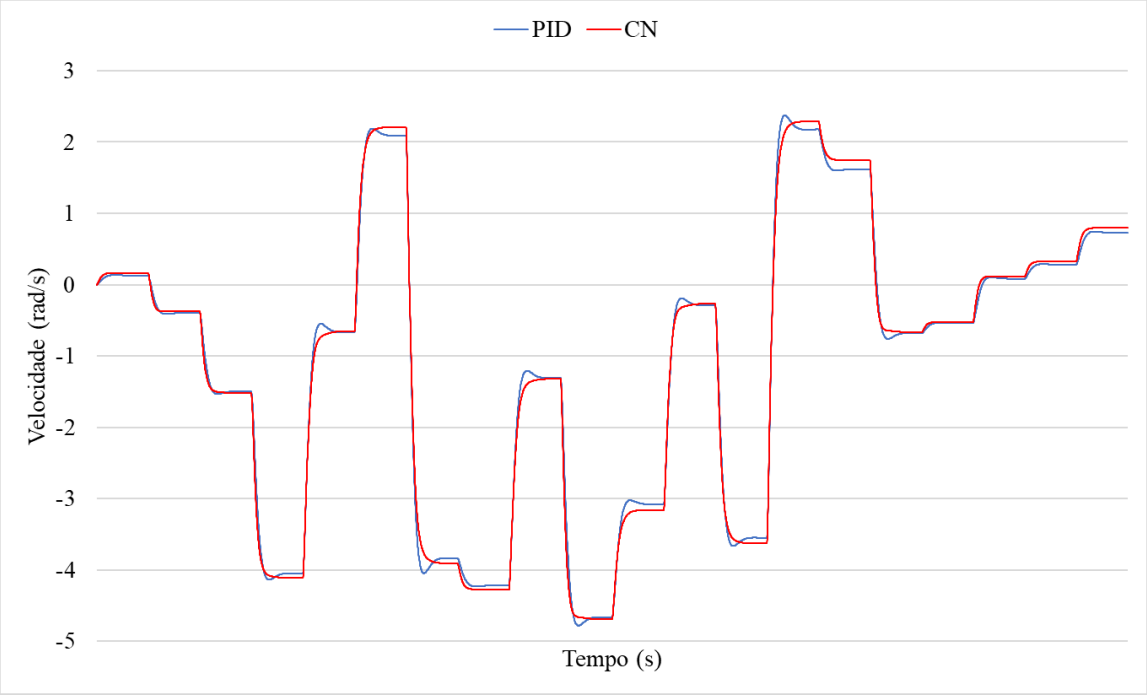
Analisando o Gráfico 26, percebemos que as saídas de tensão do controlador PID e do CN não coincidem, mesmo para os dados de treinamento. Isso é um bom sinal e significa que não ocorreu um sobreajuste excessivo do modelo aos dados de treinamento, o que causaria uma redução da sua capacidade de generalização, diminuindo seu desempenho para entradas desconhecidas, ou seja, para dados não utilizados no treinamento.

Como já apresentado anteriormente, calculamos um erro quadrático médio entre as respostas da RNA e do controlador PID de apenas 2,1883 V^2 para o conjunto de dados de teste.

Além disso, de forma semelhante ao que fizemos para a entrada degrau, removemos o controlador PID e realizamos novamente o controle do motor CC com o CN. Com isso, obtivemos o perfil de velocidade do motor CC controlado pelo CN sobreposto ao perfil de velocidade proveniente do controle do PID, ambos submetidos ao conjunto de dados de teste. O Gráfico 27 abaixo mostra essas curvas.

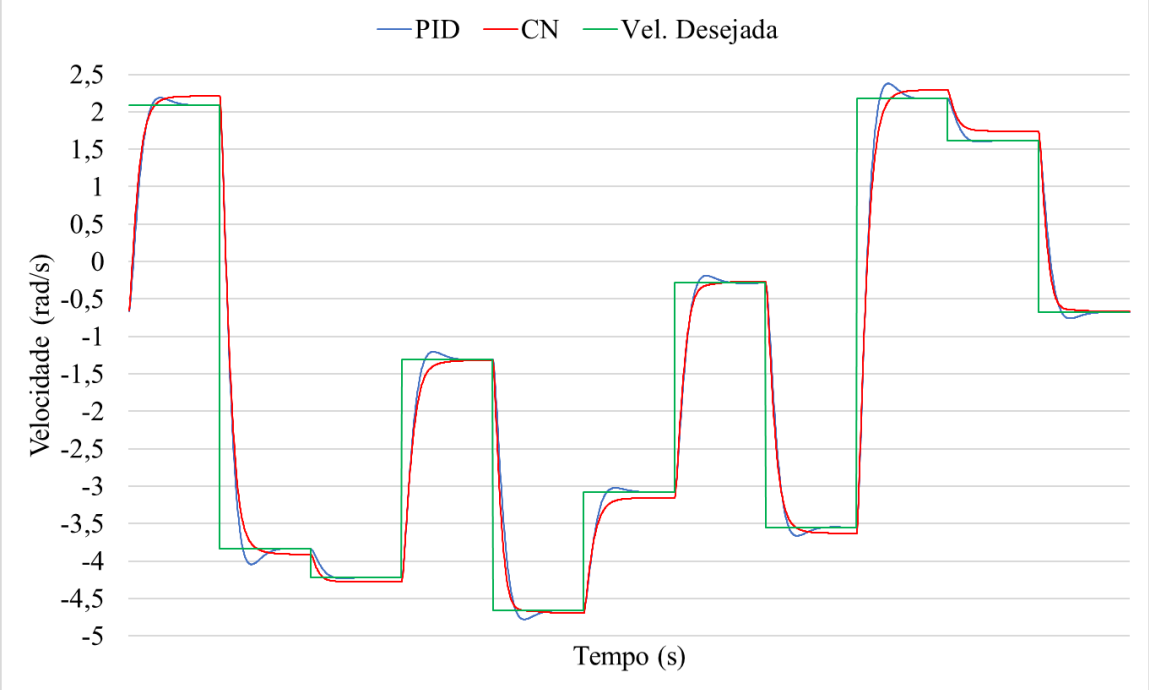
Analisando o Gráfico 27, percebemos que há algumas diferenças entre os perfis de velocidade do motor. Porém, as diferenças mais significativas ocorrem em momentos de transitórios e as velocidades costumam ficar bastante próximas nos momentos estacionários. Calculamos um erro quadrático médio de apenas 0,0097 rad^2/s^2 entre os dois perfis.

Gráfico 27 – Curvas de Velocidades do Controlador PID e CN – Dados de Teste



Fonte: Elaborado pelo autor.

Gráfico 28 – Região de Interesse das Curvas de Velocidade – Dados de Teste



Fonte: Elaborado pelo autor.

O Gráfico 28 acima consiste em uma ampliação em uma região de interesse dos perfis de velocidade do motor CC controlado pelo PID e pelo CN. Analisando esse gráfico, podemos identificar visualmente os mesmos fenômeno percebido na análise da entrada degrau

feita anteriormente, ou seja, o CN apresenta um erro estacionário maior do que o controlador PID, porém um sobressinal máximo menor.

Em se tratando do conjunto de dados de treino, ressaltamos que propositalmente não calculamos o erro quadrático médio das saídas de tensão entre o controlador PID e o CN e não esboçamos o perfil de velocidade do motor CC. Como a RNA utiliza os dados de treino para aprender as relações entre as entradas e saídas, não faz sentido calcular esses parâmetros para esse conjunto de dados, pois não há um significado relevante em seus valores, já que a rede já foi exposta a esses dados anteriormente e extraiu informações deles. A única análise referente a esse conjunto de dados que faz sentido realizar diz respeito ao sobreajuste do modelo aos dados de treinamento, já abordado anteriormente.

Portanto, levando-se em conta todas essas considerações, concluímos que o perfil de velocidade gerado pelo CN para uma entrada complexa é bastante próximo do perfil gerado pelo controlador PID.

3.12 Considerações Sobre os Resultados

Analisando os resultados como um todo, é possível afirmar que o CN é mais lento na busca da velocidade desejada no período transitório. Além disso, possui um erro estacionário maior. Contudo, sua resposta alcança uma estabilidade mais rapidamente e possui um sobressinal menor, em comparação ao controlador PID. Os perfis de velocidade gerados pelo CN e pelo PID são bastante próximos, existindo um erro quadrático médio de apenas 0,0097 rad^2/s^2 entre eles.

A referência utilizada para determinar a qualidade desse resultado e definir qual solução é superior é determinada pelo processo a ser controlado pelo motor. Como abstraímos esse requisito, ou seja, não temos uma aplicação em mente para esse motor CC, não temos condições de aprofundar nossas análises e afirmar se o CN é uma solução superior ao controlador PID.

O que podemos afirmar, sem sombra de dúvidas, é que os resultados encontrados comprovam que é possível utilizar *Deep Learning* e montar uma RNA profunda, utilizando uma arquitetura *Deep Belief Network* com retropropagação, capaz de aprender a se comportar como um controlador PID e controlar a velocidade de um motor CC.

4 CONCLUSÃO

Nosso trabalho alcançou todos os objetivos gerais e específicos. Utilizamos *Deep Learning* para modelar uma RNA capaz de controlar a velocidade de um motor CC.

Para isso, realizamos um estudo bibliográfico sobre *Machine Learning*, *Deep Learning*, controladores PID e motores CC. Apresentamos os principais conceitos desses assuntos, além de outras ferramentas utilizadas, para proporcionar um entendimento completo do estudo. Assim, foi possível discutir assertivamente todos os tópicos abordados no trabalho.

De posse dos dados provenientes das simulações do motor CC e do controlador PID, gerados no Simulink®, aplicamos um método empírico de abordagem de problemas de DL para construir, avaliar e refinar o nosso modelo. Fomos capazes de coletar e analisar informações que nos auxiliaram no processo de ajuste dos hiperparâmetros da RNA. Graças ao método, apresentamos a evolução da topologia da RNA do modelo inicial até a sua forma final.

Com o modelo final definido, realizamos uma comparação entre o CN e o PID. Vimos que o CN é mais lento no período transitório e possui um erro estacionário maior. Porém, sua resposta alcança uma estabilidade mais rapidamente e possui um sobressinal menor. Vimos que os perfis de velocidade gerados pelo CN e pelo PID são bastante próximos.

Para trabalhos futuros, sugerimos utilizar um motor CC real e montar um *hardware* para coletar os dados de treinamento e teste. Com isso, é possível substituir o controlador desse motor por uma implementação física do CN e avaliar essa solução na prática.

Além disso, sugerimos analisar a capacidade das Redes Neurais Recorrentes para resolver esse problema de controle. Acreditamos que essa arquitetura resolva bem o problema estudado devido à natureza temporal e sequencial do comportamento da velocidade do motor CC – justamente o tipo de fenômeno que as Redes Neurais Recorrentes se propõem a modelar.

Uma outra sugestão é utilizar um método de otimização para os hiperparâmetros da rede como a otimização Bayesiana, em vez do método empírico que utilizamos. É provável que se alcance melhores resultados, pois o modelo será ótimo e possuirá um erro mínimo.

Sugerimos a criação de conjuntos de dados de treino e teste que incluam mais possibilidades de variação entre dois estados de velocidade consecutivos. Com isso, é provável que o CN esteja mais preparado para responder eficientemente às mudanças de estado do motor.

Além disso, sugerimos que a possibilidade de usar o CN em conjunto com o controlador PID seja analisada, em vez de ser um substituto. Dessa forma, o motor CC poderá operar com dois sistemas de controle distintos em paralelo, cada um com modos de falhas

diferentes. Assim, com a falha de um deles, entra o outro para realizar o controle do motor, o que leva a um processo mais robusto.

Uma outra forma de usar redes neurais no controle de motores é na parametrização do controlador PID. É interessante estudar essa possibilidade em um trabalho futuro, no qual uma RNA é utilizada para encontrar os parâmetros k_p, k_i e k_d do controlador PID, em vez de substituí-lo na realização do controle.

Além disso, sugerimos que mais testes sejam realizados para validar o CN. É interessante analisar a estabilidade da resposta do CN alterando alguns parâmetros do sistema, como o momento de inércia do rotor, por exemplo.

Finalmente, é interessante analisar o comportamento do CN com diferentes perfis de carga no eixo do motor. Ressaltamos que, no nosso trabalho, foi considerado apenas o caso do motor operando a vazio. É fundamental analisar o comportamento do CN para uma carga constante, linear, quadrática etc.

REFERÊNCIAS

ABIODUN, O. I. *et al.* State-of-the-art in artificial neural network applications: A survey. **Heliyon**, v. 4, n. 11, p. e00938, 2018.

ABRAHAM, J.; SHRIVASTAVA, S. Dc Motor Speed Control Using Machine Learning Algorithm. **International Journal of Engineering Research & Technology**, v. 7, n. 04, p. 456–470, 2018.

ALEXANDER, C. K.; SADIKU, M. **Fundamentos de Circuitos Elétricos**. 5ª ed. Porto Alegre: AMGH, 2013.

ANG, K. H. *et al.* PID control system analysis and design. **IEEE Control Systems**, v. 26, n. 1, p. 32–41, 2006.

APRUZZESE, G. *et al.* On the effectiveness of machine and deep learning for cyber security. **International Conference on Cyber Conflict, CYCON**, v. 2018- May, p. 371–389, 2018.

AREL, I.; ROSE, D. C.; KARNOWSKI, T. P. Deep Machine Learning - A New Frontier in AI Research. **IEEE Computational Intelligence Magazine**, n. November, p. 13–18, 2010.

ARNOLD, L. *et al.* An introduction to deep learning. **ESANN 2011 proceedings, 19th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning**, n. April, p. 477–488, 2010.

ARUNAVA. **Towards Data Science**. Disponível em: <https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e>. Acesso em: 4 nov. 2019.

BRAGA, ANTÔNIO; CARVALHO, ANDRÉ; LUDERMIR, T. **Redes Neurais Artificiais - Teoria e Aplicações**. 2. ed. Rio de Janeiro: LTC, 2007.

BUP, A. **A Python implementation of Deep Belief Networks built upon NumPy and TensorFlow with scikit-learn compatibility**. Disponível em: <https://github.com/albertbup/deep-belief-network>. Acesso em: 13 set. 2019.

CHAPMAN, S. J. **Fundamentos de Máquinas Elétricas**. 5. ed. Porto Alegre: AMGH, 2013.

CHEN, P. *et al.* Control strategy of speed servo systems based on deep reinforcement learning. **Algorithms**, v. 11, n. 5, 2018.

CHEON, K. *et al.* On Replacing PID Controller with Deep Learning Controller for DC Motor System. **Journal of Automation and Control Engineering**, v. 3, n. 6, p. 452–456, 2015.

CHO, D.; TAI, Y. W.; KWEON, I. S. Deep Convolutional Neural Network for Natural Image Matting Using Initial Alpha Mattes. **IEEE Transactions on Image Processing**, v. 28, n. 3, p. 1054–1067, 2019.

CHOLLET, F. **Keras**. Disponível em: <https://keras.io/>. Acesso em: 13 set. 2019.

CHOLLET, F. **Deep Learning with Python und Keras: Das Praxis-Handbuch vom**

Entwickler der Keras-Bibliothek. 2. ed. Shelter Island: Manning Publications Co., 2018.

DENG, L. Three classes of deep learning architectures and their applications: a tutorial survey. **APSIPA transactions on signal and information processing**, 2012.

FITZGERALD, A. E.; KINGSLEY, C.; UMANS, S. D. **Máquinas Elétricas**. 7ª ed. Porto Alegre: AMGH, 2014.

FLEMMING, D. M.; GONÇALVES, M. B. **Cálculo A: Funções, Limite, Derivação e Integração**. 6. ed. Porto Alegre: Pearson, 2006.

GRAF, J. **PID Control Fundamentals**. 1ª ed. Scotts Valley: CreateSpace, 2016.

HAYKIN, S. **Redes Neurais: Princípios e Prática**. 2. ed. Porto Alegre: Bookman, 2003.

HAYKIN, S.; VEEN, B. VAN. **Sinais e Sistemas**. 1ª ed. Porto Alegre: Bookman, 2001.

HINTON, G.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. **Neural Computation**, v. 18, n. 7, p. 1527–1554, 2006.

HODGKING, A. L.; HUXLEY, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. **Journal of Physiology**, v. 117, p. 500–544, 1952.

HONDA, F. **Motores de Corrente Contínua: Guia rápido para uma especificação precisa**. 1ª ed. São Paulo: Siemens, 2006.

HUA, Y.; GUO, J.; ZHAO, H. Deep Belief Networks and deep learning. **Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things, ICIT 2015**, 2015.

JOHNSON, M. A.; MORADI, M. H. (EDS.). **PID Control: New Identification and Design Methods**. 1ª ed. New Yorrk: Springer Publishing, 2005.

KOVÁCS, Z. **Redes Neurais Artificiais - Fundamentos e Aplicações**. 4. ed. São Paulo: Livraria da Física, 2002.

LATHI, B. P. **Sinais e Sistemas Lineares**. 2ª ed. Porto Alegre: Bookman, 2007.

LAZZERI, S. G.; HELLER, R. An intelligent consultant system for chess. **Computers and Education**, v. 27, n. 3–4, p. 181–196, 1996.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, n. 7553, p. 436–444, 2015.

LIU, W. *et al.* A survey of deep neural network architectures and their applications. **Neurocomputing**, v. 234, n. October 2016, p. 11–26, 2017.

MA, J. *et al.* Improving air quality prediction accuracy at larger temporal resolutions using deep learning and transfer learning techniques. **Atmospheric Environment**, v. 214, n. April, p. 116885, 2019.

MAHMOUD, T. K.; DONG, Z. Y.; MA, J. A Developed Integrated Scheme Based Approach for Wind Turbine Intelligent Control. **IEEE Transactions on Sustainable Energy**, v. 8, n. 3, p. 927–937, 2017.

MAIER, A. *et al.* A gentle introduction to deep learning in medical image processing. **Zeitschrift fur Medizinische Physik**, v. 29, n. 2, p. 86–101, 2019.

MATHWORKS. **MATLAB**. Disponível em: <https://www.mathworks.com/products/matlab.html>. Acesso em: 13 set. 2019.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biology**, v. 5, n. 4, p. 115–133, 1943.

MEHDY, M. M. *et al.* Artificial neural networks in image processing for early detection of breast cancer. **Computational and Mathematical Methods in Medicine**, v. 2017, 2017.

MITCHELL, T. M. **Machine Learning**. 1. ed. New York: McGraw-Hill, 1997.

MOE, S.; RUSTAD, A. M.; HANSSEN, K. G. Machine Learning in Control Systems: An Overview of the State of the Art. **SGAI-AI 2018**, v. 11311, p. 250–265, 2018.

MOHAMMADI, M. *et al.* Deep learning for IoT big data and streaming analytics: A survey. **IEEE Communications Surveys and Tutorials**, v. 20, n. 4, p. 2923–2960, 2018.

NISE, N. S. **Engenharia de Sistemas de Controle**. 6ª ed. Rio de Janeiro: LTC, 2012.

OGATA, K. **Engenharia de Controle Moderno**. 5ª ed. São Paulo: Pearson Prentice Hall, 2010.

ROSSUM, G. VAN. **Python Organization**. Disponível em: <https://www.python.org/about/>. Acesso em: 13 set. 2019.

SARIYILDIZ, E.; YU, H.; OHNISHI, K. A practical tuning method for the robust PID controller with velocity feed-back. **Machines**, v. 3, n. 3, p. 208–222, 2015.

SILVA, IVAN; SPATTI, DANILO; FLAUZINO, R. **Redes Neurais Artificiais**. 2. ed. São Paulo: Artliber Editora Ltda., 2016.

SILVA, E. L.; MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. 4. ed. Florianópolis: Universidade Federal de Santa Catarina, 2005.

SMOLENSKY, P. Information Processing in Dynamical Systems: Foundations of Harmony Theory. In: RUMELHART, D. E.; MCCLELLAND, J. L. (Eds.). . **Parallel Distributed Processing: Explorations in the Microstructure of Cognition**. 1. ed. Cambridge: MIT Press, 1986. p. 194–281.

STEWART, J. **Cálculo - Volume 1**. 6. ed. São Paulo: Cengage, 2009a.

STEWART, J. **Cálculo - Volume 2**. 6. ed. São Paulo: Cengage, 2009b.

TAI, L. *et al.* A Survey of Deep Network Solutions for Learning Control in Robotics: From Reinforcement to Imitation. **Journal of Latex Class Files**, v. 14, n. 8, 2015.

TEAM, G. B. **TensorFlow**. Disponível em: <https://www.tensorflow.org/about/>. Acesso em: 13 set. 2019.

TORO, V. DEL. **Fundamentos de Máquinas Elétricas**. 1ª ed. Rio de Janeiro: LTC, 1999.

TRIANNI, A.; CAGNO, E.; ACCORDINI, D. A review of energy efficiency measures within electric motors systems. **Energy Procedia**, v. 158, p. 3346–3351, 2019.

TUREVSKIY, A. **PID Controller Design for a DC Motor**. Disponível em: <https://www.mathworks.com/matlabcentral/fileexchange/26275-pid-controller-design-for-a-dc-motor>. Acesso em: 13 set. 2019.

VERMEER, S. A. M. *et al.* Seeing the wood for the trees: How machine learning can help firms in identifying relevant electronic word-of-mouth in social media. **International Journal of Research in Marketing**, 2019.

WANG, D. *et al.* Detection of power grid disturbances and cyber-attacks based on machine learning. **Journal of Information Security and Applications**, v. 46, p. 42–52, 2019.

YUE, W. *et al.* Machine Learning with Applications in Breast Cancer Diagnosis and Prognosis. **Designs**, v. 2, n. 2, p. 13, 2018.

ZENG, N. *et al.* Deep Belief Networks for Quantitative Analysis of a Gold Immunochromatographic Strip. **Cognitive Computation**, v. 8, n. 4, p. 684–692, 2016.