UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

LUCIO SERGIO DE PAULA GURGEL DO AMARAL FILHO

IDENTIFICAÇÃO DO MODELO DINÂMICO DE UMA INCUBADORA NEONATAL MEDIANTE A APLICAÇÃO DE UMA REDE NEURAL CONVOLUCIONAL UNIDIMENSIONAL

FORTALEZA
2021

LUCIO SERGIO DE PAULA GURGEL DO AMARAL FILHO

IDENTIFICAÇÃO DO MODELO DINÂMICO DE UMA INCUBADORA NEONATAL MEDIANTE A APLICAÇÃO DE UMA REDE NEURAL CONVOLUCIONAL UNIDIMENSIONAL

Monografia apresentada ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará como requisito parcial à obtenção do título de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Arthur Plínio de Souza Braga

FORTALEZA

2021

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

A515i Amaral Filho, Lucio Sergio de Paula Gurgel do.
Identificação do modelo dinâmico de uma incubadora neonatal mediante a aplicação de uma rede neural convolucional unidimensional / Lucio Sergio de Paula Gurgel do Amaral Filho. – 2021.
116 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2021.
Orientação: Prof. Dr. Arthur Plínio de Souza Braga.

1. Identificação de sistemas. 2. Rede neural convolucional. 3. Incubadora neonatal. 4. Atraso de transporte. 5. Sinal PRBS. I. Título.

CDD 621.3

LUCIO SERGIO DE PAULA GURGEL DO AMARAL FILHO

IDENTIFICAÇÃO DO MODELO DINÂMICO DE UMA INCUBADORA NEONATAL MEDIANTE A APLICAÇÃO DE UMA REDE NEURAL CONVOLUCIONAL UNIDIMENSIONAL

Monografia apresentada ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará como requisito parcial à obtenção do título de bacharel em Engenharia Elétrica.

Aprovada em: ___ / ___ / _____.

BANCA EXAMINADORA

_____

Prof. Dr. Arthur Plínio de Souza Braga (Orientador)

Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Wilkley Bezerra Correia

Universidade Federal do Ceará (UFC)

_____

Eng. Me. René Descartes Olímpio Pereira

Universidade Federal do Ceará (UFC)

Àqueles que defendem a inclusão social, especialmente a dos mais vulneráveis, no ensino superior de excelência e em tantos outros espaços.

AGRADECIMENTOS

Aos meus pais, Célia e Lucio Sergio, pelo carinho, paciência, dedicação, amor incondicional e apoio constante; sem eles, eu nunca teria conseguido enfrentar os obstáculos da vida. Não existem palavras para expressar o tanto de amor e gratidão que sinto por ambos.

À minha irmã, Gabriela, pelo carinho, atenção, convívio e pelas várias vezes em que me ajudou em momentos conturbados.

Aos amigos da graduação, Edilan, Samara, Sabrina, Máurisson, Mayane, Natan, Monilson e Watson, que enfrentaram comigo as dificuldades que o curso nos lançou e com quem pude compartilhar momentos alegres e engraçados.

Aos meus amigos, Rafaela, Augusto, Ronie, Vanessa, Alice e Eliseu, com quem posso desabafar, descontrair, me divertir e cuja amizade eu valorizo muito.

Ao Matheus Bringel, amigo de infância que admiro muito e com quem, mesmo diante da correria e dos desencontros, nunca quero perder contato, porque é uma pessoa extraordinária.

Ao Eduardo, pelo carinho, incentivo, honestidade e confiança.

À turma da Casa de Cultura Alemã, em especial à Marianna, pela amizade, e ao prof. Alexander Ribeiro, pela atenção e pelas lições valiosas.

Ao prof. Arthur, por ter aceitado conduzir a orientação desse trabalho e pelos conselhos dados para que ele fosse melhorado.

Ao prof. Wilkley, pela orientação durante o período de iniciação científica e pela disposição em participar da banca examinadora.

Ao eng. René, pela paciência e auxílio neste trabalho e pelo tempo disposto a compor a banca examinadora.

A todos que estiveram ao meu lado e contribuíram de alguma forma para o meu desenvolvimento pessoal e profissional.

“Mais que máquinas, precisamos de humanidade; mais que inteligência, precisamos de bondade e ternura. Sem essas qualidades, a vida será violenta, e tudo estará perdido”.

(Charles Chaplin, **o Grande Ditador**, 1940)

RESUMO

Este trabalho propõe a identificação da dinâmica da temperatura de uma incubadora neonatal com o auxílio de uma rede neural convolucional unidimensional (*1D-CNN – One-dimensional Convolutional Neural Network*). O objetivo é usar a rede convolucional para conduzir uma regressão não linear de múltiplas saídas em alta dimensão que estima os parâmetros de um modelo dinâmico *Single-Input Single-Output* (*SISO*) capaz de representar a ação da temperatura ao longo do tempo. A rede neural aprende um mapeamento entre vetores que representam a dinâmica entrada-saída de modelos de 1ª ordem com atraso de transporte (*FOPDT – First Order Plus Dead-Time*) e seus respectivos parâmetros. A metodologia envolve inicialmente obter os sinais de resposta de milhares de modelos criados via simulação no *software MATLAB*®; aplicar um sinal degrau na incubadora, a fim de realizar uma identificação com um método tradicional de otimização no *MATLAB*®, obtendo assim um modelo dinâmico adicional, cuja constante de tempo é usada para projetar um sinal binário pseudoaleatório (*PRBS – Pseudo-random Binary Sequence*); obter um sinal de resposta da incubadora via ensaio de medição utilizando o sinal *PRBS* como entrada do sistema; treinar a *1D-CNN* com os sinais de simulação; realizar a predição dos parâmetros do sistema físico; e, por fim, testar o desempenho do modelo identificado. Esse teste consistiu em construir um gráfico contendo os sinais de resposta do sistema físico e dos modelos identificados (pelo uso da rede neural e do método de otimização) e em calcular o erro quadrático médio (*MSE – Mean Squared Error*) e o desvio padrão do erro. Além disso, os modelos criados por simulação também foram estimados, com o propósito de validar o treinamento da rede neural e de avaliar a disparidade entre os resultados obtidos com esses modelos e aqueles obtidos com o modelo dinâmico da incubadora neonatal. O valor de *MSE* resultante do modelo dinâmico proposto (modelo-*CNN*) foi aproximadamente 0,25, ao passo que o *MSE* do modelo identificado por otimização (modelo-*PRBS*) foi em torno de 2,10.

Palavras-chave: Identificação de sistemas. Rede neural convolucional. Incubadora neonatal. Atraso de transporte. Sinal *PRBS*.

ABSTRACT

This work puts forward the identification of the temperature dynamics of a neonatal incubator with the aid of a one-dimensional convolutional neural network (1D-CNN). The goal is to use the convolutional network in order to carry out a high-dimensional multi-output nonlinear regression which estimates the parameters of a single-input single-output (SISO) dynamic model capable of depicting the temperature's activity over time. The neural network learns a mapping between the vectors that represent the input-output dynamics of first order plus dead-time (FOPDT) models and their respective parameters. The methodology entails initially obtaining the response signals from thousands of models created via MATLAB® software simulation, applying a step signal into the incubator in order to perform an identification with a traditional optimization method in MATLAB® thus obtaining an aditional dynamic model whose time constant is used to design a pseudo-random binary sequence (PRBS) signal, obtaining a response signal from the incubator via measurement experiment using the PRBS signal as the system input, training the 1D-CNN with the simulation signals, executing the prediction of the physical system's parameters and lastly testing the identified model's performance. That test consisted of plotting a graph containing the response signals of the physical system and the identified models (by using the neural network and the optimization method) and of evaluating the mean squared error (MSE) and the error standard deviation. Furthermore, the models created through simulation were also estimated, for the purpose of validating the neural network's training and assessing the disparity between the results obtained with these models and the ones obtained with the neonatal incubator dynamic model. The MSE value resulting from the proposed dynamic model (CNN-model) was aproximately 0.25, whilst the identified-by-optimization model's MSE (PRBS-model) was about 2.10.

Keywords: System identification. Convolutional neural network. Neonatal incubator. Dead time. PRBS signal.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

UTI	Unidade de Terapia Intensiva
CNN	*Convolutional Neural Network*
ConvNet	*Convolutional Neural Network*
1D-CNN	*One-dimensional Convolutional Neural Network*
NLP	*Natural Language Processing*
PRBS	*Pseudo-random Binary Sequence*
MIMO	*Multiple-Input Multiple-Output*
SISO	*Single-Input Single-Output*
MSE	*Mean Squarred Error*
GPAR	Grupo de Pesquisa em Automação, Controle e Robótica
DEE	Departamento de Engenharia Elétrica
UFC	Universidade Federal do Ceará
PWM	*Pulse-Width Modulation*
FOPDT	*First Order Plus Dead-Time*
RNA	Rede Neural Artificial
ReLU	*Rectified Linear Unit*
MLP	*Multilayer Perceptron*
GD	*Gradient Descent*
Adam	*Adaptive Moment Estimation*
IDE	*Integrated Development Environment*
GPU	*Graphics Processing Unit*
CPU	*Central Processing Unit*
csv	*Comma-Separated Values*

LISTA DE SÍMBOLOS

$G(s)$	Função de transferência do modelo dinâmico
$G_T(s)$	Função de transferência do modelo dinâmico de temperatura
$G_U(s)$	Função de transferência do modelo dinâmico de umidade
$G_{PRBS}(s)$	Modelo dinâmico usado no projeto do sinal *PRBS*
$G_{CNN}(s)$	Modelo dinâmico identificado com a *1D-CNN*
$X(s)$	Entrada da função de transferência
$Y(s)$	Saída da função de transferência
s	Variável da Transformada de Laplace
e	Constante de Euler
K	Ganho estático
τ	Constante de tempo
θ	Atraso de transporte (*dead time*)
K_T	Ganho estático de temperatura
τ_T	Constante de tempo de temperatura
θ_T	Atraso de transporte (*dead time*) de temperatura
K_U	Ganho estático de umidade
τ_U	Constante de tempo de umidade
θ_U	Atraso de transporte (*dead time*) de umidade
T_S	Tempo de amostragem
T	Período do sinal *PRBS*
T_b	Intervalo entre níveis do sinal *PRBS*
n	Ordem do sinal *PRBS*
N	Comprimento do sinal *PRBS* em termos de T_b
$\tau_{mín}$	Menor constante de tempo do sistema dinâmico
$\sum$	Operador de somatório
m	Combinação linear no somador do neurônio

x_i	i-ésimo atributo de entrada da RNA
$\boldsymbol{x}$	Vetor de atributos de entrada
w_i	i-ésimo peso sináptico do neurônio
$\boldsymbol{w}$	Vetor de pesos sinápticos
y	Rótulo
y_n	Rótulo da n-ésima amostra
$\hat{y}$	Valor de saída do neurônio
$\widehat{y_n}$	Valor de saída do neurônio a partir da n-ésima amostra
f	Função de ativação
z	Argumento da função de ativação
b	*Bias* do neurônio
α	Coeficiente angular
$\in$	É elemento de / pertence a
J	Função de perda (*loss function*) / função de custo (*cost function*)
M	Quantidade de amostras de treinamento
lr	Taxa de aprendizagem (*learning rate*)
∇	Operador de gradiente
$\frac{df}{dz}$	Derivada da função f em relação à variável z
$c(t)$	Função contínua de convolução
$c[t]$	Função discreta de convolução
$a(j)$	Função contínua convolvida
$a[j]$	Função discreta convolvida
$b(j)$	Função contínua convolvente
$b[j]$	Função discreta convolvente
$*$	Operador de convolução
$\circledast$	Operador de convolução
v_K	Vetor contendo os ganhos estáticos do espaço amostral

i_K	limite inferior do intervalo v_K
f_K	limite superior do intervalo v_K
p_K	passo do intervalo v_K
$v_{\boldsymbol{\tau}}$	Vetor contendo as constantes de tempo do espaço amostral
$i_{\boldsymbol{\tau}}$	limite inferior do intervalo $v_{\boldsymbol{\tau}}$
$f_{\boldsymbol{\tau}}$	limite superior do intervalo $v_{\boldsymbol{\tau}}$
$p_{\boldsymbol{\tau}}$	passo do intervalo $v_{\boldsymbol{\tau}}$
v_θ	Vetor contendo os atrasos de transporte do espaço amostral
i_θ	limite inferior do intervalo v_θ
f_θ	limite superior do intervalo v_θ
p_θ	passo do intervalo v_θ
q	dado numérico sem *feature scaling*
q^*	dado numérico com *feature scaling*
q_{max}	maior valor de q do *dataset*
q_{min}	menor valor de q do *dataset*
μ	Média
σ	Desvio padrão
®	Marca registrada

SUMÁRIO

1 INTRODUÇÃO

Este capítulo introduz o leitor à temática do estudo e está segmentado em 5 seções; inicialmente apresenta-se o tema do trabalho e sua justificativa, depois os objetivos gerais e específicos, um resumo da metodologia do trabalho e a estrutura dos capítulos que compõem esta monografia.

1.1 Justificativa e Apresentação do Tema

Segundo (MINISTÉRIO DA SAÚDE, 2020, p. 1), “340 mil bebês nascem prematuros todo ano no Brasil”. Trata-se de mais de 12% dos nascimentos no país que aconteceram antes de a gestação completar 37 semanas, sendo o dobro do índice de países europeus e concedendo a posição lamentável de 10º país no *ranking* de prematuridade (REDE NACIONAL PRIMEIRA INFÂNCIA, 2019). Em geral, as consequências do parto prematuro englobam danos fisiológicos na criança, psicológicos na família e, inclusive, econômicos na sociedade. De fato, o custo para o tratamento de recém nascidos prematuros pode ultrapassar 15 bilhões de reais por ano, considerando uma média de 51 dias de internação em Unidade de Terapia Intensiva (UTI) Neonatal, impactando tanto os serviços públicos de saúde como os de saúde suplementar (convênios) (FILHO; PÁSSARI; NIVEIROS, 2017; REDE NACIONAL PRIMEIRA INFÂNCIA, 2019). O cenário é ainda mais agravado pelos efeitos da infecção por COVID-19, aumentando a incidência da prematuridade e, até mesmo, de aborto espontâneo (WOODWORTH *et al.*, 2020).

Os neonatos prematuros possuem um baixo peso e, em razão disso, não conseguem manter estável sua temperatura corporal (OLIVEIRA, 2007). Nesse contexto, incubadoras neonatais são os aparelhos responsáveis por impedir a mortalidade dos prematuros, constituindo um meio de transição entre o ambiente uterino e o externo e promovendo o desenvolvimento do seu organismo até funcionar autonomamente. Em síntese, proporcionam a manutenção da temperatura e da umidade em níveis favoráveis à saúde do recém-nascido por meio da medição e do controle dinâmico dessas grandezas, reproduzindo artificialmente condições semelhantes às do útero materno (GPAR, 2020).

Dada a relevância desse equipamento, é fundamental que seus controladores sejam bem projetados, o que só é permitido em decorrência de uma identificação apropriada do seu sistema dinâmico. Esse procedimento consiste na construção de

um modelo matemático (função de transferência) que descreva razoavelmente o comportamento dinâmico de um sistema (RODRIGUES, 1996). Tal procedimento envolve a estimação dos parâmetros numéricos do modelo a partir da análise do sinal de saída, gerado por algum sinal de entrada; quanto mais precisa for a estimação, mais fiel será o modelo dinâmico. A identificação de um determinado sistema tende a ser realizada reiteradamente, pois a busca por modelos dinâmicos cada vez mais precisos impacta direta e positivamente a efetividade dos controladores a serem construídos.

Várias técnicas próprias de identificação de sistemas podem ser empregadas, porém vislumbra-se uma oportunidade de aplicar conhecimentos sobre inteligência artificial, sobretudo Redes Neurais Artificiais, tendo em vista o crescimento acentuado da área nos últimos anos. Nesse sentido, redes recorrentes podem surgir como uma opção, visto que são úteis no processamento de dados temporais (GOODFELLOW; BENGIO; COURVILLE, 2016), e que os sinais de resposta gerados pelos sistemas na identificação podem ser interpretados como tais. No entanto, tem sido constatado, em trabalhos como o de (BAI; KOLTER; KOLTUN, 2018), que redes neurais convolucionais (*CNN – Convolutional Neural Network*) também podem ser utilizadas em problemas desse tipo, conseguindo inclusive obter um desempenho superior ao de redes recorrentes em tarefas como processamento de linguagem natural (*NLP – Natural Language Processing*), de músicas polifônicas e reconhecimento de caracteres manuscritos. Para tanto, deve ser implementado um tipo adaptado de rede convolucional, de modo a poder receber dados sequenciais ou séries temporais, denominado de rede neural convolucional unidimensional (*1D-CNN – One-dimensional Convolutional Neural Network*). Essa adaptação se mostra relevante porque tais redes foram originalmente pensadas para aplicações em visão computacional (LECUN *et al.*, 1998), processando dados matriciais, como imagens e vídeos. Desde que foram desenvolvidas, as redes convolucionais unidimensionais têm sido muito utilizadas em aplicações de processamento de sinais, tais como diagnósticos precoces de arritmia cardíaca em eletrocardiogramas (KIRANYAZ *et al.*, 2015), monitoramento de falhas mecânicas em motores (INCE *et al.*, 2016), entre outras.

1.2 Objetivos Gerais

O trabalho presente tem como objetivo geral identificar a dinâmica da temperatura de um protótipo de incubadora neonatal utilizando redes neurais convolucionais capazes de manipular dados temporais.

1.3 Objetivos Específicos

Tendo em vista o objetivo geral, foram delineados os seguintes objetivos específicos:

1) Descrever as principais características da incubadora neonatal;
2) Explanar o procedimento de identificação de sistemas;
3) Expor os fundamentos de redes neurais convolucionais;
4) Submeter a incubadora neonatal a um ensaio e obter seu sinal de resposta temporal;
5) Criar por meio de simulação outras amostras de sinais.
6) Definir uma arquitetura para a *1D-CNN* e realizar seu treinamento; e
7) Estimar o modelo dinâmico da incubadora com a *1D-CNN* (modelo-*CNN*) e avaliar seu desempenho.

1.4 Procedimentos da Metodologia

A metodologia do trabalho é sinteticamente descrita a seguir:

1) Criar uma grande quantidade de modelos dinâmicos lineares mediante simulações computacionais no *software MATLAB*® e salvar seus sinais de resposta gerados com sinais (sequências) binários pseudoaleatórios de entrada (*PRBS – Pseudo-random Binary Sequence*) para formar o conjunto de amostras (*dataset*) do problema.

2) Identificar previamente um modelo dinâmico (modelo-*PRBS*) usando uma resposta da incubadora ao sinal degrau, com a principal finalidade de projetar um sinal de entrada *PRBS* para a incubadora.

3) Submeter a incubadora a um ensaio com o estímulo do sinal *PRBS* e salvar o sinal de resposta resultante, completando assim o *dataset*. Embora a incubadora neonatal possa ser considerada um sistema *MIMO* (*Multiple-Input Multiple-Output*) (BEZERRA CORREIA; CLAURE TORRICO; OLÍMPIO PEREIRA,

2017), neste trabalho ela é tratada como um sistema *SISO* (*Single-Input Single-Output*), sendo composto apenas de sua malha de temperatura.

4) Com o *dataset* preenchido, submetê-lo à *1D-CNN*, processando-o, a fim de realizar a aprendizagem supervisionada, e estimando os parâmetros do modelo dinâmico por meio de uma regressão não linear de múltiplas saídas.

5) Por fim, aferir a precisão do modelo-*CNN* com a comparação gráfica da resposta obtida, com o erro quadrático médio (*MSE – Mean Squarred Error*) e com o desvio padrão de erro, em relação ao sinal medido na incubadora.

1.5 Estrutura do Trabalho

Esta monografia está dividida em 7 capítulos, concluindo-se nesta seção o primeiro deles, que discorreu sobre as ideias gerais da temática, as motivações pertinentes, os objetivos, uma síntese da metodologia e a estrutura do trabalho.

No Capítulo 2, são feitas considerações sobre a incubadora neonatal, apresentando-se características, modo de funcionamento e conhecimento prévio sobre sua função de transferência.

No Capítulo 3, são apresentados os fundamentos da identificação de sistemas dinâmicos.

No Capítulo 4, são abordados os conceitos fundamentais das redes neurais convolucionais e expostos os elementos e a arquitetura da *1D-CNN* aplicada.

No Capítulo 5, a metodologia, introduzida na Seção 1.4, é apresentada de forma mais detalhada.

No Capítulo 6, é feita a abordagem dos experimentos, executados a partir da metodologia com o auxílio dos *softwares MATLAB*® e *Spyder*®, e da obtenção e avaliação dos resultados.

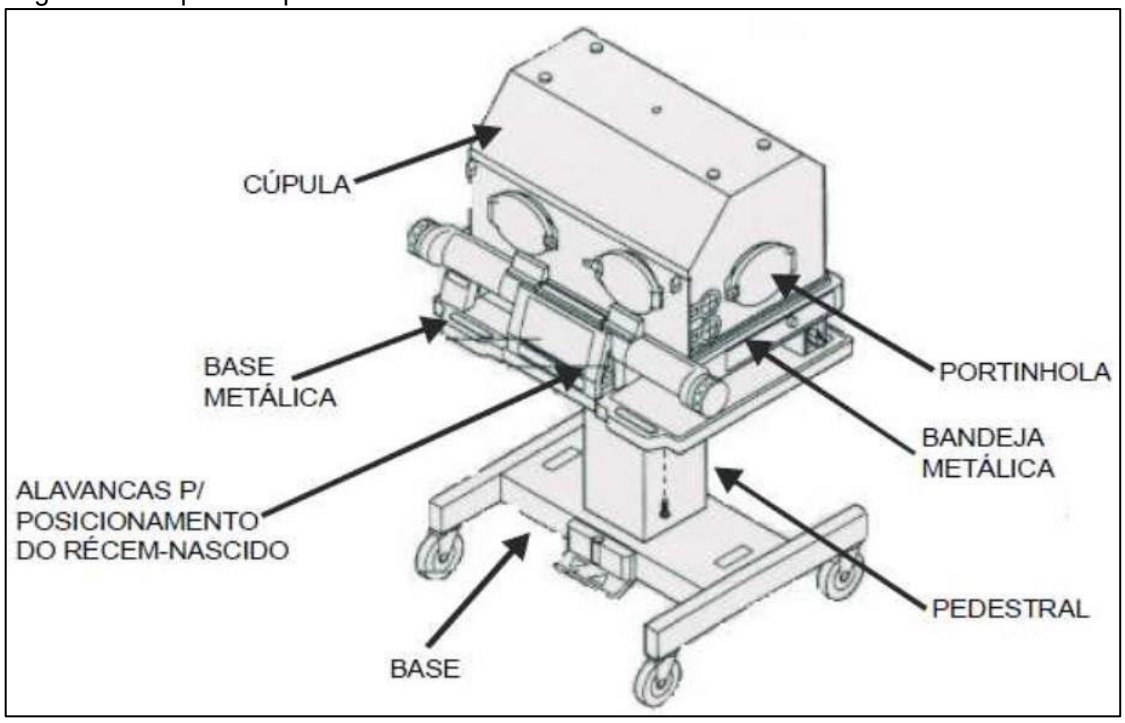
Por último, no Capítulo 7, são expostas as considerações finais sobre a problemática, contrapondo os resultados alcançados aos objetivos estabelecidos, e são sugeridas ideias para trabalhos futuros.

2 INCUBADORA NEONATAL

A incubadora neonatal é um equipamento hospitalar projetado para auxiliar bebês prematuros no desenvolvimento e na manutenção das funções de seu organismo, reduzindo as chances de mortalidade. Recém-nascidos de baixo peso encontram dificuldade em manter sua temperatura corporal, e, por isso, a principal atribuição da incubadora é oferecer-lhes um ambiente termoneutro (ALBUQUERQUE, 2012; OLIVEIRA, 2007). Desse modo, a temperatura do neonato permanece dentro dos limites aceitáveis, sua taxa metabólica é reduzida (em termos de consumo de oxigênio e calorias) e sua saúde é mais resguardada (OLIVEIRA; TAVARES; MORAES, 2007). A representação e a imagem de uma incubadora comercial são exibidas, respectivamente, na Figura 1 e na Figura 2.

Para os experimentos compreendidos neste trabalho, foi utilizado um protótipo (Figura 3) desenvolvido pelo Grupo de Pesquisa em Automação, Controle e Robótica (GPAR), formado por professores e pesquisadores do Departamento de Engenharia Elétrica (DEE) da Universidade Federal do Ceará (UFC).

Figura 1 – Esquema representativo de uma incubadora neonatal comercial

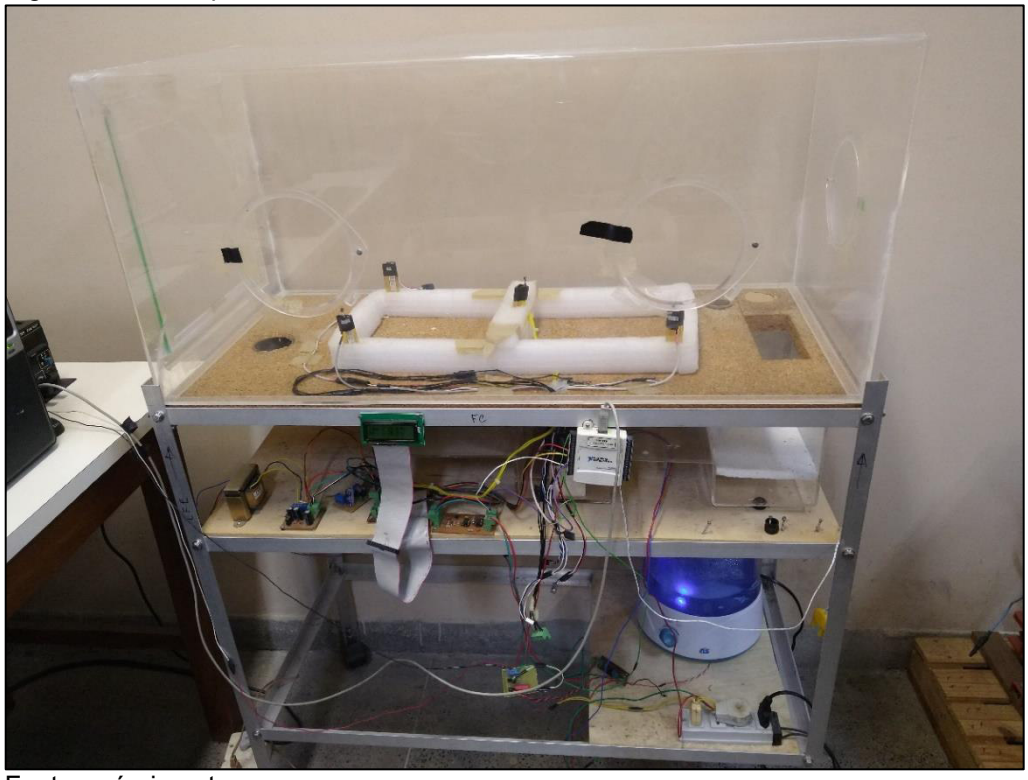


Fonte: Agostini (2003, p. 16)

Figura 2 – Modelo comercial de incubadora neonatal

Fonte: próprio autor

Figura 3 – Protótipo de incubadora neonatal



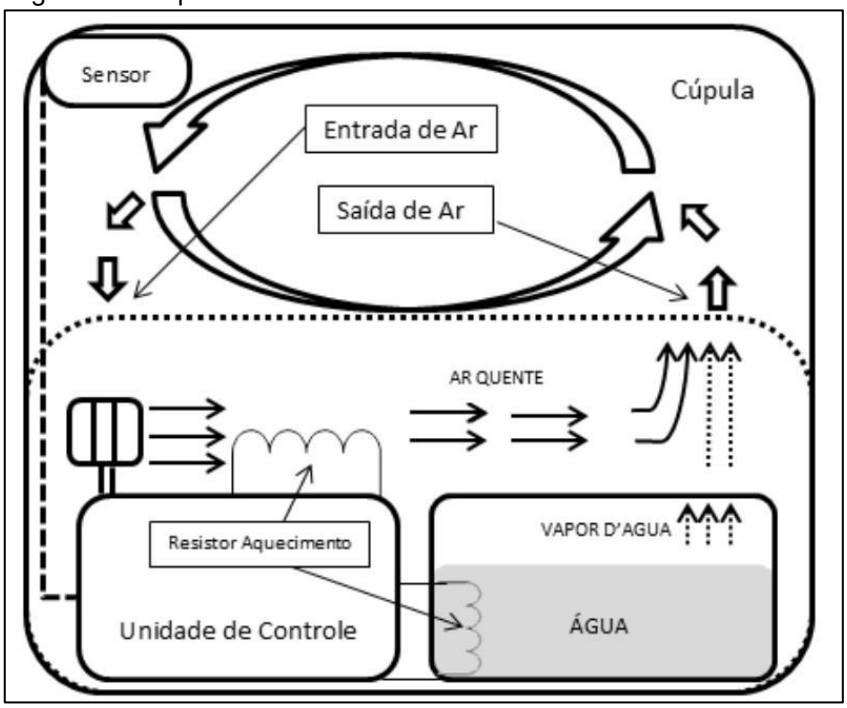
Fonte: próprio autor

Nas próximas seções, são descritos, em linhas gerais, o princípio de funcionamento da incubadora e sua função de transferência.

2.1 Funcionamento

Como já apontado, a incubadora visa a manutenção de um ambiente termoneutro, que, segundo (IFF/FIOCRUZ, 2019, p. 3), varia “de acordo com a idade gestacional e com o peso do bebê”. O intuito é promover a normotermia do bebê prematuro, isto é, conseguir manter sua temperatura corporal entre 36,5 °C e 37,5 °C (WORLD HEALTH ORGANIZATION; MATERNAL AND NEWBORN HEALTH/SAFE MOTHERHOOD, 1997; TREVISANUTO; TESTONI; ALMEIDA, 2018; IFF/FIOCRUZ, 2019). Para tal, é realizado o controle da temperatura e da umidade no interior da cúpula (doma). Um esquema representativo da operação pode ser observado na Figura 4.

Figura 4 – Esquema de funcionamento de uma incubadora neonatal



Fonte: Albuquerque (2012, p. 21)

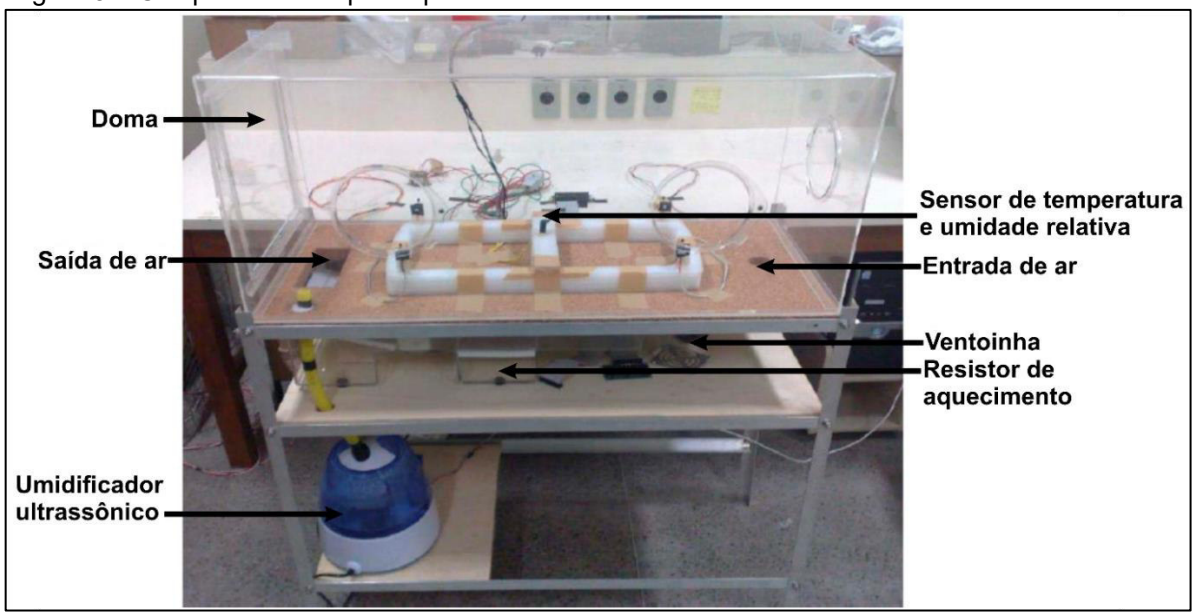
Albuquerque (2012, p. 20) descreve o funcionamento do aparelho nos seguintes termos:

> “A Incubadora Comercial realiza o controle da temperatura da cúpula onde fica o recém-nascido através de um resistor de aquecimento em conjunto

> com um sistema de circulação de ar. O controle da umidade relativa do ar no interior da cúpula é realizado através do vapor d'água produzido em função do aquecimento de uma porção de água presente em um reservatório da incubadora. Para efetuar o aquecimento desta água, o sistema de controle atua em um resistor de aquecimento mergulhado neste reservatório. Assim, o ar quente e o vapor d'água gerados são colocados no sistema de circulação de ar e chegam até a cúpula".

Em essência, o princípio de funcionamento do protótipo é o mesmo, ressalvadas algumas alterações na construção dos componentes, como pode ser indicado na Figura 5.

Figura 5 – Componentes do protótipo



Fonte: adaptado de Olímpio Pereira (2016, p. 63)

A diferença principal está no modo de produção de vapor d'água; no momento de elaboração desta monografia, não foi necessário o aproveitamento de calor por um resistor adicional, visto que o umidificador já se encarregava inteiramente disso.

A incubadora é acionada mediante modulação de largura de pulso (*PWM – Pulse-Width Modulation*) na fonte de alimentação, aplicando níveis de intensidade de temperatura conforme a largura do pulso do sinal de entrada.

A utilidade do equipamento para o estudo é fornecer informações sobre a dinâmica de sua resposta ao sinal *PRBS* (abordado na Seção 3.5), a partir das quais

é possível encontrar uma função de transferência. Na próxima seção, essa função é abordada.

2.2 Função de Transferência

Uma função de transferência é a representação matemática de alguma variável ou grandeza (temperatura, umidade, corrente elétrica, etc.) de determinado sistema físico (incubadora, circuito elétrico, etc.). A principal finalidade desse recurso é ser empregado no desenvolvimento de controladores, que garantem a manutenção das variáveis em valores desejados. No caso da incubadora, esses valores são a temperatura e a umidade ideais que permitem um ambiente equilibrado para o recém-nascido.

Na realidade, o sistema estudado possui duas funções (uma para cada variável controlada), sendo ambas de 1ª ordem, embora apenas a função referente à temperatura seja estudada neste trabalho, uma vez que a análise de sistemas *MIMO* foge ao escopo deste estudo. Um sistema dinâmico de 1ª ordem pode ser representado da seguinte forma:

$$G(s) = \frac{Y(s)}{X(s)} = \frac{K}{\tau s + 1} e^{-\theta s} \quad (1)$$

em que:

- $G(s)$ é a função de transferência;
- $X(s)$ é a variável de entrada;
- $Y(s)$ a variável de saída ou controlada;
- s é a variável da Transformada de Laplace;
- K é o ganho estático;
- τ é a constante de tempo; e
- θ é o atraso de transporte ou *dead time*.

Em razão da presença desse atraso de transporte, esse sistema dinâmico pode ser denominado de *FOPDT* (*First Order Plus Dead-Time*). Dessa forma, as funções de transferência da incubadora em relação a cada uma de suas variáveis são:

$$G_T(s) = \frac{K_T}{\tau_T s + 1} e^{-\theta_T S} \quad (2)$$

$$G_U(s) = \frac{K_U}{\tau_U s + 1} e^{-\theta_U S} \quad (3)$$

Idealmente, sistemas dinâmicos são especificados pela física do processo, abordagem também denominada de modelagem fenomenológica (ORENSTEIN, 2013). Isto é, “a modelagem matemática é feita com base em equações que descrevem os princípios fundamentais do comportamento desses sistemas” (AGUIRRE, 2019a) como as leis da Termodinâmica, as leis de Kirchhoff, entre outras. Na prática, contudo, as funções de transferência dos sistemas, sobretudo dos mais complexos, são comumente obtidas com a aplicação da chamada identificação de sistemas, que é a abordagem do estudo presente. Diferentemente da modelagem fenomenológica, esse procedimento é orientado aos dados (e não às equações), de modo que são conhecidos os sinais de entrada e de saída, e procura-se, a partir deles, obter um modelo matemático do sistema (AGUIRRE, 2019a). Mais adiante, no próximo capítulo, o assunto é detalhado.

3 IDENTIFICAÇÃO DE SISTEMAS DINÂMICOS

Neste capítulo são abordadas noções de identificação de sistemas, discorrendo-se sobre os tipos de modelagens, em um dos quais a identificação se encaixa; as propriedades dos modelos; as etapas do procedimento; a técnica selecionada para achar o modelo-*PRBS*; e esse tipo de sinal escolhido como entrada.

Existem algumas maneiras de definir o conceito de identificação de sistemas. Essencialmente, é o procedimento através do qual se obtém representações ou modelos matemáticos que, com uma precisão tolerável, descrevem os sistemas dinâmicos e a sua relação com as entradas e as saídas. Outra definição é a de que se trata de uma “interface entre os modelos do mundo matemático e o mundo real” (JOHANSSON, 1993 *apud* COELHO; COELHO, 2004, p. 35).

Os modelos matemáticos em questão são úteis para analisar o comportamento de sistemas através de simulações, mas, sobretudo, são os objetos imprescindíveis para se aplicar as técnicas de controle de sistemas. Estes assumem muitas categorias, desde sistemas mecânicos até sistemas econômicos. Em relação a sistemas econômicos, sociais ou ambientais, comumente não se dispõe de um grande conjunto de dados nem de muito conhecimento prévio sobre suas operações dinâmicas. Por outro lado, sistemas elétricos e mecânicos podem ser mais facilmente analisados, devido às eventuais características previamente conhecidas e à maior facilidade de submetê-los a ensaios (KEESMAN, 2011).

O procedimento de construir modelos implica determinar qual tipo de modelagem se deseja realizar, como pode ser visto a seguir.

3.1 Tipos de Modelagem

Existem 3 (três) tipos de modelagem (AGUIRRE, 2019a):

- **Modelagem caixa branca:** o sistema tem seu funcionamento analisado por meio das relações matemáticas da Física ou de outra área da ciência. Também é denominado de “modelagem pela física ou natureza do processo”, “modelagem conceitual” ou “modelagem fenomenológica”.
- **Modelagem caixa preta:** o sistema tem seu funcionamento analisado por meio dos dados. O sistema é submetido a um sinal de entrada, gerando

uma saída. O objetivo, então, passa a ser estudar esses sinais de modo a achar uma relação capaz de explicar as transformações observadas no sinal de saída. Também é denominado de “modelagem empírica” ou, simplesmente, de identificação caixa preta.

- **Modelagem caixa cinza:** o sistema é analisado de forma orientada a dados, igualmente ao tipo caixa preta, mas também dispõe de informações auxiliares não apresentadas nos dados inseridos ou gerados. Um exemplo dessas informações é o conhecimento qualitativo da proporcionalidade entre dois determinados fatores (direta ou inversamente proporcionais entre si). À medida que a quantidade de informação aumenta, esse tipo passa a adquirir um padrão cada vez mais “claro”. A informação e a maneira com que ela é usada dependem dos métodos de identificação aplicados. Também é denominado de identificação caixa cinza.

O significado das designações dos tipos é o nível necessário de transparência do sistema. Em outras palavras, um sistema sob modelagem caixa branca precisa fornecer informações muito detalhadas, por meio das equações, a fim de que possa ser modelado apropriadamente. Isto é, ele precisa ser transparente em relação ao seu conhecimento prévio. Em contrapartida, no que concerne aos tipos caixa cinza e caixa preta, é necessário, respectivamente, pouco ou nenhum conhecimento prévio sobre o sistema. Isto é, o objeto de um estudo assim são os dados de entrada e de saída, pelo fato de a transparência em relação ao conhecimento prévio não ser (ou ser apenas parcialmente) um requisito para a modelagem (AGUIRRE, 2019a).

Na prática, para muitos sistemas, as equações que os descrevem não são extraídas com facilidade; ou elas não são conhecidas, ou são conhecidas, mas determinar seus parâmetros se torna inviável por razões de excessiva complexidade ou de limitações do experimento. Nesse contexto, a identificação de sistemas (modelagens caixa preta e caixa cinza) surge como uma alternativa à modelagem caixa branca, uma vez que a determinação detalhada dos princípios físicos do sistema é dispensada ao se fazer o estudo da relação de causa e efeito dos dados de entrada e de saída.

A identificação realizada nesta monografia é do tipo caixa cinza, em virtude do prévio conhecimento da ordem da função de transferência do sistema físico (1ª

ordem) e da presença de um atraso de transporte θ, além dos dados empíricos extraídos nos experimentos.

Logo a seguir, são expostas as principais propriedades dos modelos a serem consideradas.

3.2 Propriedades dos modelos

Como explicitado anteriormente, a identificação é responsável por produzir modelos matemáticos que atendam especificidades desejadas. Esses modelos, por sua vez, assumem diferentes propriedades, de acordo com o método empregado no procedimento. As propriedades de maior interesse são (KEESMAN, 2011):

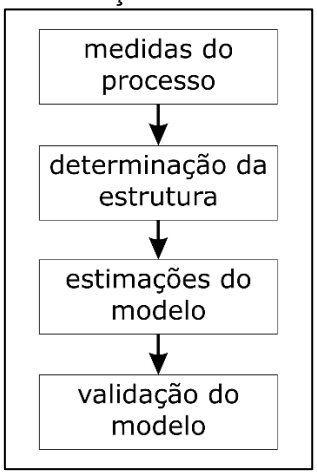
- **Linearidade:** considerando duas entradas $u_1(t)$ e $u_2(t)$, duas saídas correspondentes $y_1(t)$ e $y_2(t)$ e duas constantes α e β, um sistema é considerado linear quando a ele é submetida uma entrada $\alpha u_1(t) + \beta u_2(t)$, e é produzida uma saída $\alpha y_1(t) + \beta y_2(t)$. Ou seja, quando as propriedades de superposição e de homogeneidade são satisfeitas.
- **Invariância no tempo:** considerando uma entrada $u_1(t)$, uma saída correspondente $y_1(t)$ e um deslocamento no tempo τ, um sistema é considerado invariante no tempo quando a ele é submetida uma entrada $u_1(t+\tau)$, e é produzida uma saída $y_1(t+\tau)$.
- **Causalidade:** considerando duas entradas $u_1(t)$ e $u_2(t)$, uma constante t_1 e, ainda, que $u_1(t) = u_2(t)$ sempre que $t < t_1$ (os dois sinais tem o mesmo comportamento histórico), um sistema é considerado causal quando $y_1(t_1) = y_2(t_1)$; e estritamente causal quando o mesmo resultado ocorre e $u_1(t) = u_2(t)$ sempre que $t \leq t_1$. Ou seja, quando o sinal de saída depende de entradas passadas e da presente ou, *a contrario sensu*, quando não depende de entradas futuras. Todos os sistemas físicos são causais.
- **Dinamicidade**: um sistema é considerado dinâmico quando seu sinal de saída $y(t)$ depende do seu histórico, não somente da entrada presente. Ou seja, quando possui memória, podendo ser representado por equações diferenciais (no regime de tempo contínuo) ou por equações de diferenças (no regime de tempo discreto). *A contrario sensu*, um sistema estático não possui memória e é representado por equações algébricas.

A especificação das propriedades do modelo em estudo é de relevante interesse, sobretudo para a determinação da sua estrutura, na medida em que proporciona informação e orientação necessárias para o levantamento de modelos adequados para o problema. Essa determinação de estrutura compõe uma das etapas da identificação, que são apresentadas logo a seguir.

3.3 Etapas de Identificação

A fim de melhor detalhá-lo, o procedimento da identificação, ilustrado na Figura 6, é separado em 4 (quatro) etapas ou fases, sendo elas (LJUNG, 1999; COELHO; COELHO, 2004):

Figura 6 – Etapas de identificação



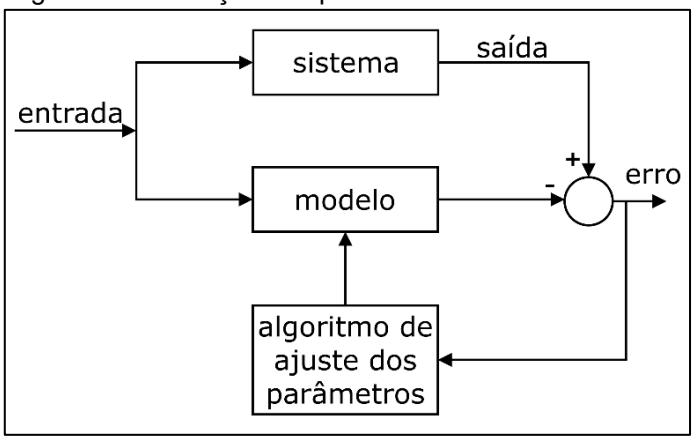
Fonte: adaptado de Coelho e Coelho (2004, p. 32)

1) **Aquisição de dados (medidas do processo):** um sinal de entrada de certa duração é selecionado, em conformidade com a técnica de identificação a ser realizada, estimulando o sistema a produzir um sinal de saída com um determinado tempo de amostragem T_S. Os dados gerados são, então, armazenados digitalmente. Exemplos comuns de sinais de entrada são o sinal degrau, o ruído branco (*white noise*) e o *PRBS*.
2) **Determinação da estrutura:** um conjunto específico de modelos candidatos é escolhido com base nos dados coletados, nas propriedades do modelo e em qualquer conhecimento prévio existente sobre ele. Essa

escolha é a mais difícil do procedimento de identificação, sobretudo quando não há conhecimento prévio ou quando não há uma relação clara o suficiente entre o sinal de entrada e o de saída para aferir adequadamente os parâmetros envolvidos.

3) **Estimação dos parâmetros do modelo:** consiste em um procedimento numérico que tem como propósito determinar os parâmetros do modelo, utilizando algum método de identificação e determinando o melhor modelo no conjunto escolhido. Essa operação é comumente caracterizada como um problema de otimização, e a avaliação do modelo é realizada por um determinado critério de qualidade, tipicamente baseado no desempenho do modelo em tentar reproduzir os dados medidos. Iterativamente, os parâmetros são ajustados de forma a atender o critério estabelecido, sendo este frequentemente descrito como uma função de erro. Essa dinâmica está retratada na Figura 7.

Figura 7 – Estimação dos parâmetros



Fonte: adaptado de Coelho e Coelho (2004, p. 33)

4) **Validação do modelo:** encontrado o modelo ótimo, ele é submetido a um teste, indicando se é válido ou não para representar o sistema. Esse teste é executado pela comparação gráfica entre os sinais de resposta do sistema físico (real) e os do modelo estimado. Nos principais métodos de identificação, existem duas opções para tal (AGUIRRE, 2019b):

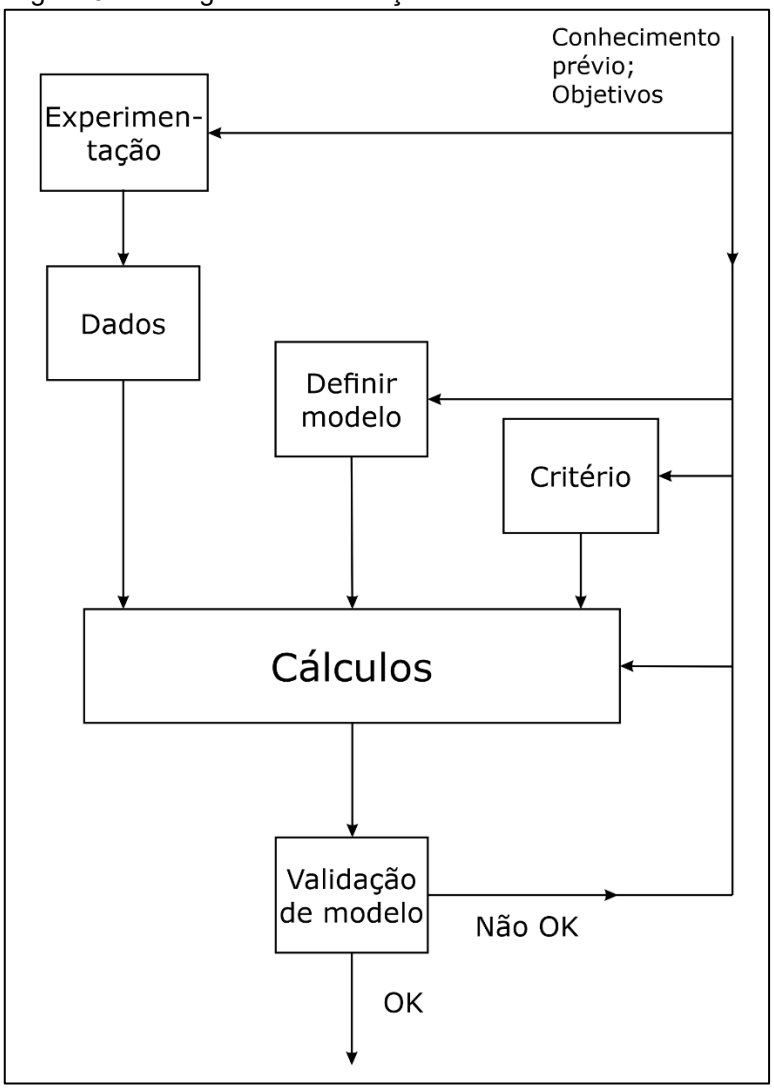
   a. Validar o modelo estimado com uma fração do sinal de resposta real, sendo a fração restante empregada na construção do modelo.

As proporções 75%-25% e 50%-50% são frequentemente escolhidas; ou

b. Validar o modelo estimado com um sinal de medição adicional no sistema físico, distinto do sinal empregado na construção do modelo.

Se considerado válido, o modelo pode ser usado para descrever o sistema naquele ponto ou faixa de operação; do contrário, o procedimento é repetido, na forma da Figura 8, a partir da etapa que resultou em prejuízos para a qualidade do modelo.

Figura 8 – Fluxograma de validação de modelos



Fonte: adaptado de Keesman (2011, p. 11)

Com isso, é possível observar muita semelhança compartilhada entre a identificação de sistemas e as Redes Neurais Artificiais (Capítulo 4); para ambos é exigida a validação em dados que não foram previamente submetidos aos modelos. Devido a essa semelhança, para os experimentos deste trabalho em específico, a validação de modelos descrita nesta seção se torna redundante, sendo utilizada somente na identificação do modelo-*PRBS* da Seção 3.4, ocasião na qual a proporção de 75%-25% foi aplicada. De fato, para testar a confiabilidade do modelo dinâmico estimado com a *1D-CNN* (modelo-*CNN*), é necessário proceder de acordo com a experimentação relatada na Seção 6.4, analisando-se a curva de perdas (Figura 26) e o desempenho das estimações no *validation set* (Apêndice A). Na Seção 5.1, encontra-se uma justificativa mais detalhada sobre essa temática.

A estimação de parâmetros é uma etapa crucial, visto que, como mencionado, é nela que ocorre o emprego de um método de identificação propriamente dito. Na próxima seção, é abordado o método utilizado, no início dos experimentos, para obter o modelo-*PRBS*.

3.4 Identificação por Otimização

Nesta seção, pretende-se discorrer brevemente sobre a identificação do modelo-*PRBS* implementada nos experimentos. Ela possui a finalidade de conhecer previamente um valor aproximado da constante de tempo $\tau_{mín}$, a fim de projetar corretamente o sinal de entrada *PRBS* (Seção 3.5) a ser submetido à incubadora.

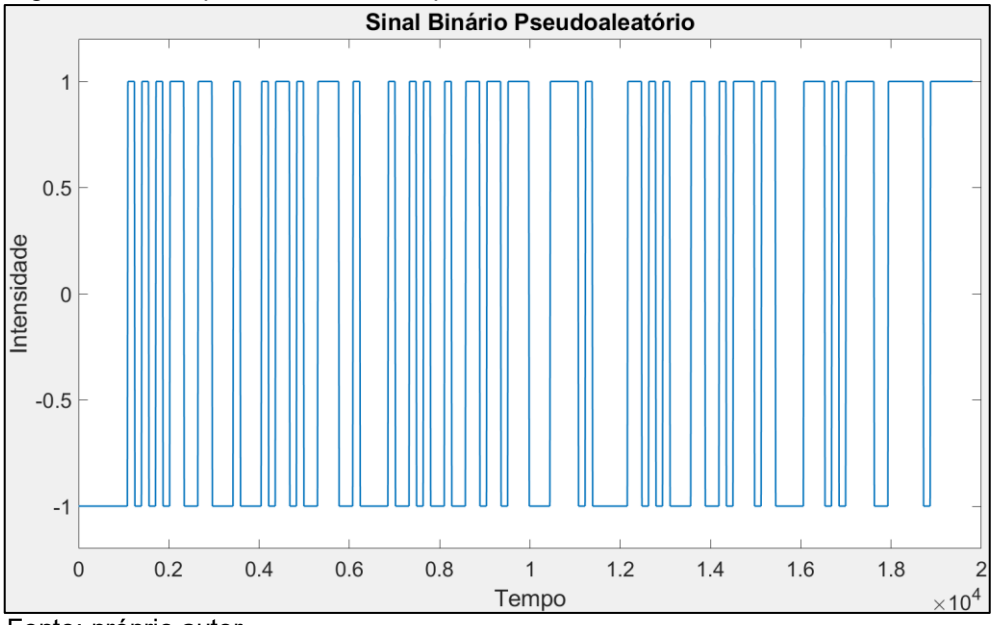
Esse procedimento foi utilizado a partir da resposta do sistema ao sinal degrau, por meio do *System Identification Toolbox 9.9* no *software MATLAB®* e da função *fmincon*. Com essa função, encontra-se um “valor mínimo de uma função-objetivo não linear multivariável delimitada por um intervalo” (MATHWORKS, 2021, p. 1). Nesse caso, a função-objetivo é a soma (ou média) dos erros quadráticos entre a resposta ao degrau medida e a resposta do modelo-*PRBS*. A maioria dos outros algoritmos disponibilizados no *toolbox*, a exemplo do método de Gauss-Newton (LAI; KEK; GAIK, 2017), opera de forma semelhante, estimando os modelos com base em otimização e minimização de uma função-objetivo por aproximações sucessivas. Essa identificação prévia é retratada na Seção 6.2.

O sinal de resposta a ser analisado pode possuir diferentes formas, a depender do sistema e do sinal de entrada que o excita. O tipo de sinal escolhido como estímulo foi o *PRBS*, que é explanado na seção a seguir.

3.5 Sinal Binário Pseudoaleatório

Dentre os sinais usualmente aplicados, se destaca o sinal binário pseudoaleatório (*PRBS – Pseudo-random Binary Sequence*) (Figura 9), porque permitem atingir uma faixa de diferentes pontos de operação, sendo bastante utilizados em processos industriais (MACHADO, 2004). Além disso, são gerados com facilidade, necessitando assumir somente dois valores ($+V$ e $-V$ ou $+V$ e 0), justificando a qualificação de “binário”.

Figura 9 – Exemplo de sinal binário pseudoaleatório



Fonte: próprio autor

Esses valores são alternados em instantes discretizados de tempo, sendo um sinal periódico com período T, tal que:

$$T = N.T_b \tag{4}$$

em que T_b é o intervalo de tempo entre as mudanças de valor, e N é o número de intervalos T_b que resulta no período T. O valor de N é determinado com base na variável inteira n, a ordem do sinal *PRBS*, que é regida pela Equação 5. Os valores

atribuídos a essas variáveis durante os experimentos são $n = 7$ e, consequentemente, $N = 127$. Essa especificação também é feita na Seção 6.1 e no código-fonte do Apêndice B.

$$N = 2^n - 1 \tag{5}$$

As mudanças de valores ao longo do sinal são estipuladas de forma determinística e, por isso, o sinal não é inteiramente aleatório, e sim pseudoaleatório (AGUIRRE, 2007).

O tempo T_b é determinado de acordo com a menor constante de tempo $\tau_{mín}$ observada no sistema, devendo respeitar o intervalo da Equação 6. O valor de $\tau_{mín}$ pode ser determinado por algum método de identificação, como aquele retratado na Seção 3.4.

$$\frac{\tau_{min}}{10} \leq T_b \leq \frac{\tau_{min}}{3} \tag{6}$$

Em síntese, uma quantidade entre 3 e 10 de alternâncias de valor em uma sequência *PRBS* deve ser gerada dentro do intervalo de tempo equivalente à menor constante de tempo. O valor de T_b não pode ser tão grande; do contrário, o sinal percebido pelo sistema seria um sinal degrau. Por outro lado, T_b não pode ser tão pequeno, senão o sistema não tem tempo suficiente de produzir uma resposta aos valores de cada intervalo. Assim, os intervalos do sinal *PRBS* assumem uma certa duração que permite que o sistema reaja de forma adequada, enquanto satisfatoriamente impedem que ele atinja o regime permanente (AGUIRRE, 2007).

Neste estudo, sinais *PRBS* são projetados para formar os dados a serem processados com a *1D-CNN*. É necessário, para tal, compreender o modo com o qual essa rede convolucional processa suas amostras de entrada. Portanto, esse e outros assuntos pertinentes são discutidos no Capítulo 4 a seguir.

4 REDES NEURAIS ARTIFICIAIS E CONVOLUCIONAIS

Neste capítulo objetiva-se montar uma base teórica sobre as redes neurais convolucionais, um dos principais modelos de aprendizagem profunda (*deep learning*), a fim de que a metodologia do experimento seja plenamente compreendida. Para isso, são apresentados inicialmente, na Seção 4.1, os fundamentos de redes neurais artificiais. Esses conceitos são requisitos para o entendimento das redes convolucionais, já que estas são um tipo específico de redes neurais artificiais. Em seguida, são abordadas a Rede *Multilayer Perceptron* (Seção 4.2), a rede convolucional (Seção 4.3) e a aprendizagem das redes neurais (Seção 4.4).

4.1 Redes Neurais Artificiais

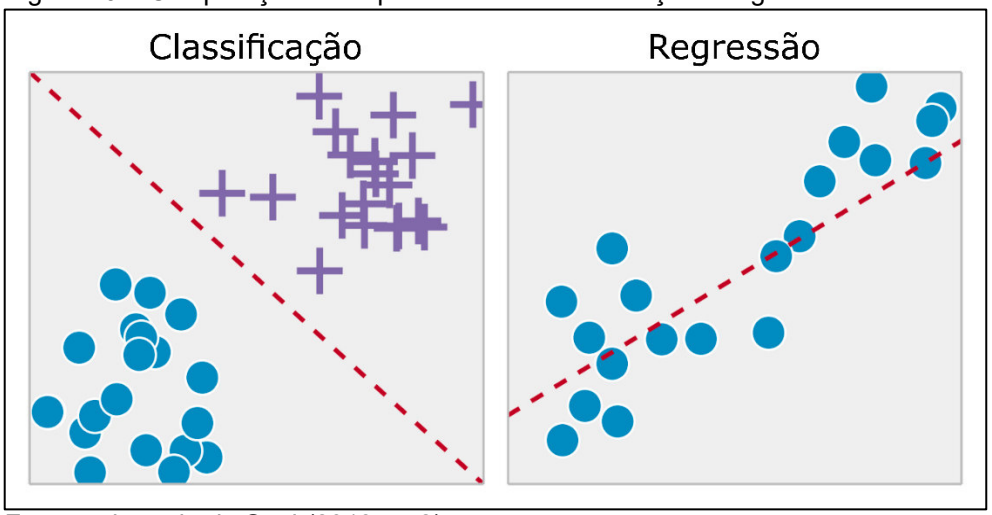
A Rede Neural Artificial (RNA) é uma ferramenta computacional que reproduz a funcionalidade e a estrutura do cérebro humano, no que tange ao armazenamento e à aprendizagem de informações. Faceli *et al.* (2011, p. 109) afirma que “esse sistema artificial é composto de unidades de processamento simples extremamente interconectados”, responsáveis por realizar transformações matemáticas. Por esse motivo, as RNAs compartilham duas principais semelhanças com as redes neurais biológicas, segundo (HAYKIN, 1999, p. 24):

- **Aprendizagem:** por meio de uma técnica (algoritmo) de aprendizagem, uma rede é capaz de obter conhecimento.
- **Conexão:** a aprendizagem é efetivada de acordo com a importância (peso) atribuída a cada uma das conexões da rede.

Modelos de aprendizagem supervisionada são destinados a solucionar dois principais tipos de tarefa, quais sejam, classificação e regressão (Figura 10). Essas tarefas são descritas por (GOODFELLOW; BENGIO; COURVILLE, 2016, p. 100-101) nos seguintes termos:

- **Classificação:** neste tipo de tarefa, é solicitado ao programa computacional especificar a qual das k categorias alguns dados pertencem [...]. Um exemplo de tarefa de classificação é o reconhecimento de objetos, em que o dado de entrada é uma imagem (usualmente descrita como uma matriz contendo níveis de brilhos de píxeis) e o valor de saída é um código numérico identificando o objeto na imagem [...].
- **Regressão:** neste tipo de tarefa, é solicitado ao programa computacional predizer um valor numérico a partir de alguns dados [...]. Um exemplo de tarefa de regressão é a previsão do valor esperado do sinistro que um segurado receberá (usado para definir prêmios de seguros), ou a previsão de preços futuros dos títulos de seguro [...].

Figura 10 – Comparação entre problemas de classificação e regressão



Fonte: adaptado de Soni (2018, p. 2)

Embora uma rede convolucional bidimensional comumente se direcione a problemas de classificação, como por exemplo saber distinguir cães de gatos a partir de imagens, a rede convolucional unidimensional (*1D-CNN*) deste trabalho é destinada a realizar uma regressão não linear de múltiplas saídas, já que tem por fim estimar matematicamente 3 (três) parâmetros contínuos por meio de uma ferramenta capaz de adquirir conhecimento sobre atributos não lineares das amostras. Por se tratar de uma regressão, a função de ativação aplicada na camada de saída da *1D-CNN* desta monografia é a função identidade (Equação 11 e Figura 13), em vez da função *softmax*, que é usada em classificação (GOODFELLOW; BENGIO; COURVILLE, 2016).

Caracterizado o tipo de tarefa da aplicação, o processo de aprendizagem pode começar a ser executado. O que torna possível essa obtenção de conhecimento pela rede neural são as unidades que a constituem, chamadas de neurônios.

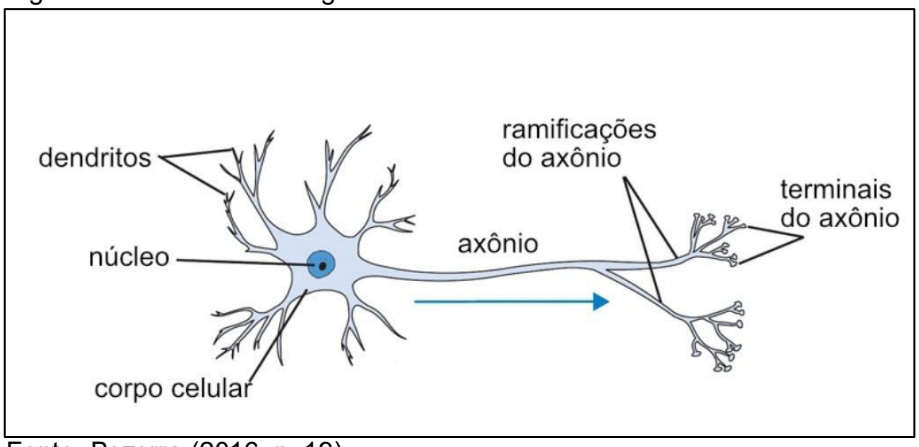
4.1.1 Neurônio

As células neuronais de um cérebro humano podem ser comparadas a formigas de uma colônia; se consideradas isoladamente, não são capazes de muitas proezas, mas podem gerar resultados valiosos quando trabalham de forma sistêmica e em grande número (SASAKI; PRATT, 2018). Da mesma forma, um único neurônio artificial é bastante limitado, não sendo hábil a realizar tarefas complexas. Por outro

lado, quando existem dezenas ou centenas de neurônios interconectados em determinadas configurações, seu potencial de desempenho e sua utilidade passam a ser evidenciados.

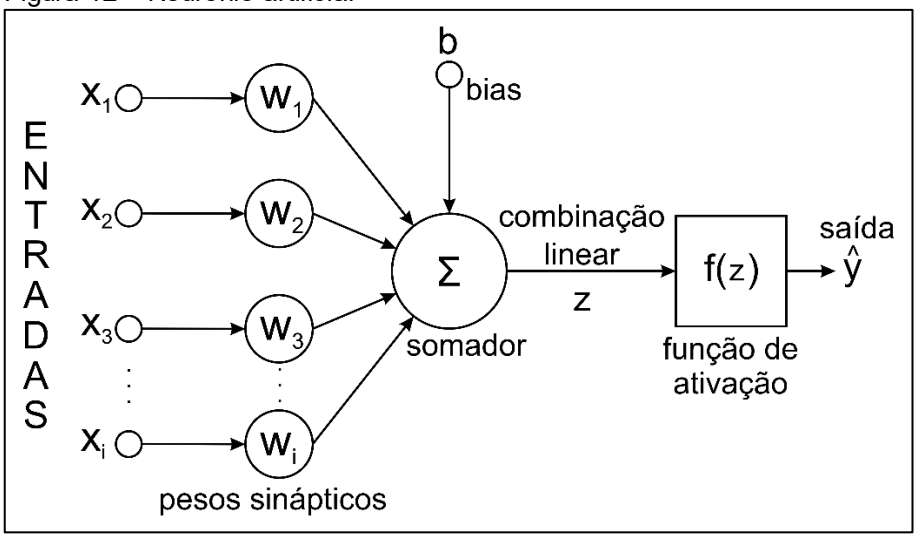
O neurônio é a unidade de processamento simples das RNAs e modela a célula do sistema nervoso. A Figura 11 ilustra a representação do neurônio biológico; e a Figura 12, a do neurônio artificial.

Figura 11 – Neurônio biológico



Fonte: Bezerra (2016, p. 19)

Figura 12 – Neurônio artificial



Fonte: adaptado de Nakamura (2013, p. 15)

O modelo da Figura 12 também é denominado de *Perceptron*, a RNA com a mais básica arquitetura (formada por apenas um neurônio), e foi idealizado por

(ROSENBLATT, 1958). Ele é formado por terminais de entradas (equivalentes aos dendritos), pesos sinápticos (equivalentes às sinapses químicas), um somador (equivalente ao núcleo celular) e um terminal de saída (equivalente à extremidade do axônio). Seu princípio de funcionamento é receber dados através dos terminais de entradas, atribuir pesos sinápticos às conexões de cada terminal, computá-los em seu somador por meio de uma combinação linear e uma função de ativação e gerar um valor através do terminal de saída (HAYKIN, 1999). O valor de saída $\hat{y}$ produzido pelo modelo da Figura 12 pode ser descrito pela Equação 7 e pela Equação 8, nas quais $\boldsymbol{x} = [x_1, x_2, \dots, x_i]$ e $\boldsymbol{w} = [w_1, w_2, \dots, w_i]^T$.

$$m = \sum_{a=1}^{i} x_a . w_a = \boldsymbol{x} . \boldsymbol{w} \tag{7}$$

$$\hat{y} = f(z) = f(m + b) \tag{8}$$

O parâmetro b é denominado de *bias* e, para fins de simplificação, pode ser incorporado no modelo como um peso $w_0 = b$ com uma entrada unitária fixa $x_0 = 1$, alterando as Equações (7) e (8) para a seguinte maneira (HAYKIN, 1999):

$$m = \sum_{a=0}^{i} x_a . w_a \tag{9}$$

$$\hat{y} = f(z) = f(m) \tag{10}$$

A notação $\hat{y}$ indica que essa saída é um valor estimado com base nos atributos de entrada $[x_1, x_2, \dots, x_i]$ e no seu rótulo (*label*) correspondente y. Na Seção 4.4, são explicados os conceitos sobre atributos de entrada e rótulos; e, na Seção 5.1, são apresentados os formatos que os elementos citados assumem neste trabalho.

Como sugere a Equação 10, a saída $\hat{y}$ é produzida pela função f, chamada de função de ativação.

4.1.2 Função de Ativação

A função de ativação transforma os estímulos de entrada do neurônio em uma saída final ou em uma entrada para a próxima camada da rede. Além disso, impacta diretamente na eficácia e precisão das predições.

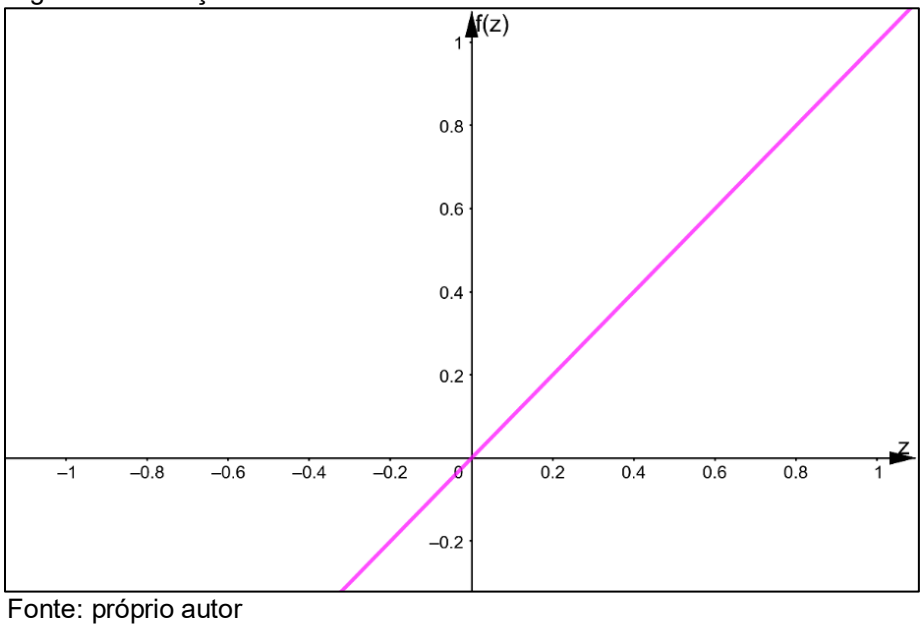
Variados tipos de função são usados, e achar a função de ativação ideal para cada problema, tipo de rede ou arquitetura é, por si só, um campo de estudo vasto, ativo e complexo. Nesse sentido, é possível citar os artigos de (KARLIK; OLGAC, 2011), (APICELLA, *et al.*, 2021) e (XU, *et al.*, 2015). Este último, diante da comparação de desempenho entre algumas funções, aponta resultados não conclusivos, reforçando a ideia de que o assunto ainda carece de amadurecimento em aspectos matemáticos.

Todavia, a adoção de boas práticas na construção de algoritmos de *deep learning* possibilita determinar as funções mais comumente escolhidas. As funções de ativação utilizadas neste trabalho foram as seguintes:

- Identidade: Equação 11 e Figura 13.
- Retificador linear (*ReLU – Rectified Linear Unit*): Equação 12 e Figura 14.
- *Leaky ReLU*: Equação 13 e Figura 15.

$$f(z) = z \tag{11}$$

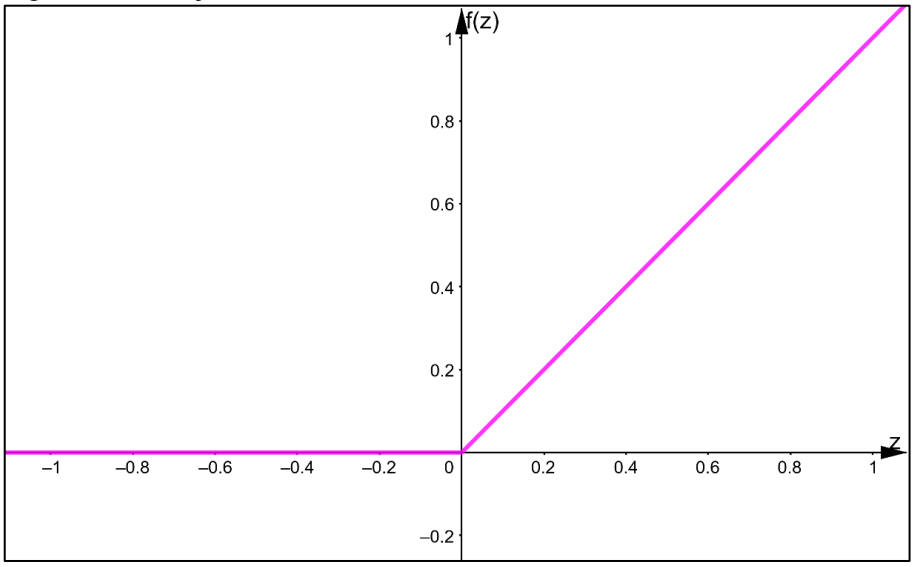
Figura 13 – Função identidade



Fonte: próprio autor

$$f(z) = \max(0, z) = \begin{cases} 0, & se\ z \leq 0 \\ z, & se\ z > 0 \end{cases} \tag{12}$$

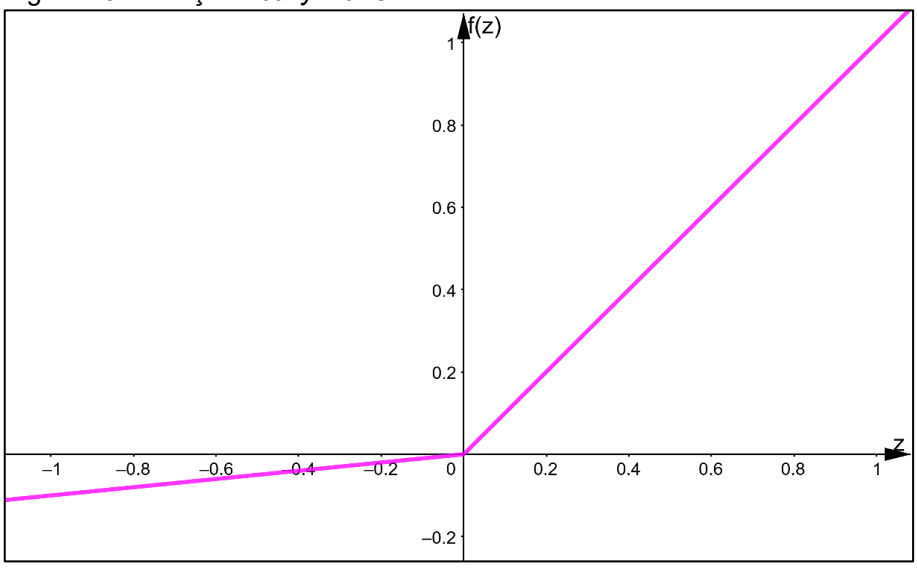
Figura 14 – Função *ReLU*



Fonte: próprio autor

$$f(z) = \max(\alpha z, z) = \begin{cases} \alpha z, & se\ z \leq 0 \\ z, & se\ z > 0 \end{cases}, \quad \alpha \in (0, 1) \tag{13}$$

Figura 15 – Função *Leaky ReLU*



Fonte: próprio autor

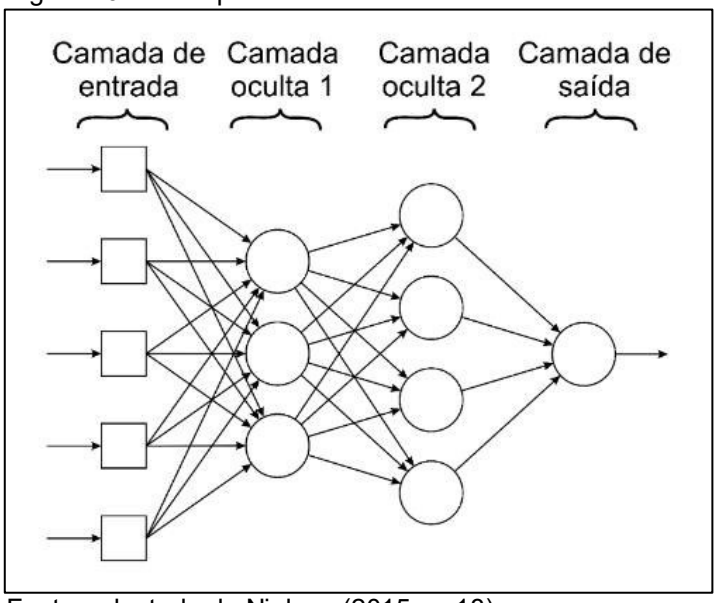
Em *deep learning*, principalmente em redes convolucionais, neurônios são ativados com *ReLU* ou alguma de suas variantes. Entre outras vantagens, *ReLU* é mais simples de calcular, por conseguinte impondo muito menos esforço computacional, e permite maior rapidez na convergência de valores ótimos de peso (LIU, 2017). Além disso, essa função adiciona não linearidade aos dados (SUPERDATASCIENCE TEAM, 2018). Imagens, séries temporais e outros dados do mundo real são normalmente não lineares, então, quando esses dados são submetidos à operação linear de convolução (Subseção 4.3.1), essa qualidade é atenuada. Logo, a *ReLU* é usada para compensar esse efeito. Entretanto, durante o treinamento, complicações podem eventualmente surgir do uso dessa função, podendo ser resolvidas com a substituição da *ReLU* pela variante *Leaky ReLU*. Essa questão é melhor abordada na Subseção 4.4.2.

Como mencionado, embora a cada neurônio seja atribuída uma função de ativação, é comum que funções idênticas sejam designadas aos neurônios de uma camada inteira. De fato, a função *Leaky ReLU* foi atribuída a todas as camadas intermediárias da *1D-CNN*. Essencialmente, os agrupamentos em camadas de unidades neuronais formam um tipo bastante comum de RNA: o *Perceptron* Multicamadas.

4.2 Rede *Multilayer Perceptron*

Um *Perceptron* Multicamadas (*MLP – Multilayer Perceptron*) é criado quando os neurônios são combinados em camadas e concatenados, de forma que as saídas da primeira camada alimentam as entradas da segunda, e assim por diante, até fornecer o valor final de saída. Na Figura 16, está ilustrado um exemplo de rede MLP.

Figura 16 – Exemplo de rede MLP



Fonte: adaptado de Nielsen (2015, p. 18)

Vários aspectos fundamentais orbitam em torno das camadas intermediárias ou ocultas (*hidden layers*). Alguns deles são a não linearidade (já mencionada) e a profundidade de uma RNA, conceitos primordiais em relação à capacidade de solucionar problemas complexos. A não linearidade é incorporada nas camadas intermediárias de uma RNA com funções de ativação não lineares, como é o caso da *ReLU* e da *Leaky ReLU*. Por sua vez, a profundidade é adquirida à medida que o número de camadas intermediárias aumenta, possibilitando modelar atributos (características ou *features*) de forma hierarquizada, isto é, construindo conceitos complexos a partir de conceitos mais simples. Matematicamente, isso pode ser representado por uma composição de funções de ativação, como na Equação 14 (PONTI; COSTA, 2017). Por exemplo, considerando que uma rede tenha a tarefa de reconhecer a presença de cubos em imagens, ela possivelmente empreenderia a seguinte estratégia: detectar linhas retas (verticais, horizontais e diagonais) em sua 1ª *hidden layer*, o que permitiria a detecção de quadrados, retângulos e paralelogramos na 2ª *hidden layer*, o que permitiria, por fim, a detecção de cubos na camada de saída.

$$f(z) = f_N(\dots f_2(f_1(z))\dots) \tag{14}$$

O conhecimento sobre redes *MLPs* é relevante porque elas compõem a estrutura de redes neurais convolucionais. Estas, por sua vez, são abordadas na próxima seção.

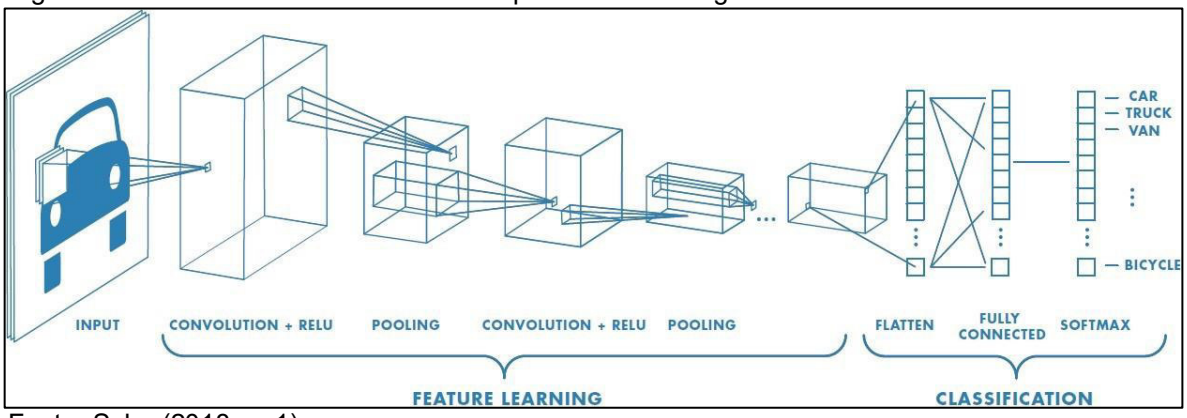
4.3 Redes Neurais Convolucionais

Redes neurais convolucionais (*CNN* ou *ConvNet – Convolutional Neural Network*) são aquelas destinadas a processar dados que apresentam uma configuração de grade ou de matriz (GOODFELLOW; BENGIO; COURVILLE, 2016), geralmente imagens ou vídeos.

O modo de funcionamento das *CNNs*, desde a inserção da entrada até a produção dos resultados, é descrito pelas operações de convolução, *pooling*, planificação (*flattening*) e conexão total (*full connection*), como resume a Figura 17. No exemplo da Figura 17 e da Figura 18, é retratada uma tarefa de classificação de imagens de elementos urbanos (carros, caminhões, pedestres, entre outros).

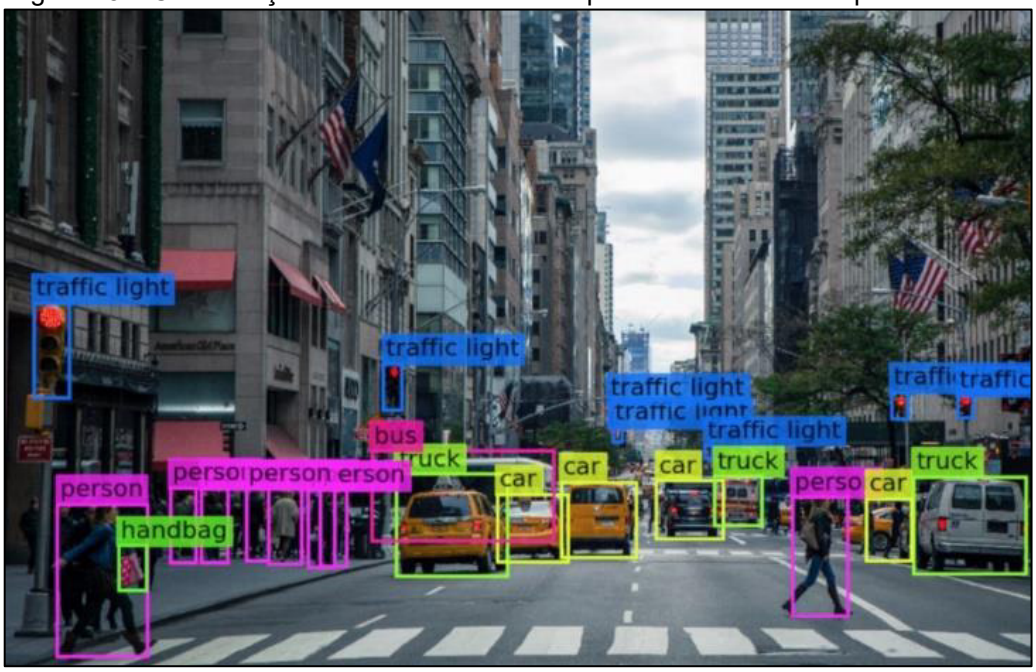
A operação de cada uma das camadas é descrita a partir da próxima subseção. Porém, a fim de proporcionar uma melhor intuição e uma maior facilidade no entendimento da exposição de tais etapas, considera-se especificamente o caso em que os dados de entrada são arranjos bidimensionais (imagens).

Figura 17 – Funcionamento de uma *CNN* que classifica imagens de elementos urbanos



Fonte: Saha (2018, p. 1)

Figura 18 – Classificação de elementos urbanos por meio de visão computacional



Fonte: Viana (2020, p. 1)

4.3.1 Convolução

Conforme (GOODFELLOW; BENGIO; COURVILLE, 2016, p. 330), “redes convolucionais são simplesmente redes neurais que usam a convolução, no lugar da multiplicação genérica de matrizes, em pelo menos uma de suas camadas”. Essa operação é denotada matematicamente pela Equação 15, para variáveis contínuas; e pela Equação 16, para variáveis discretas.

$$c(t) = (a * b)(t) = \int a(j)\, b(t - j) dj \tag{15}$$

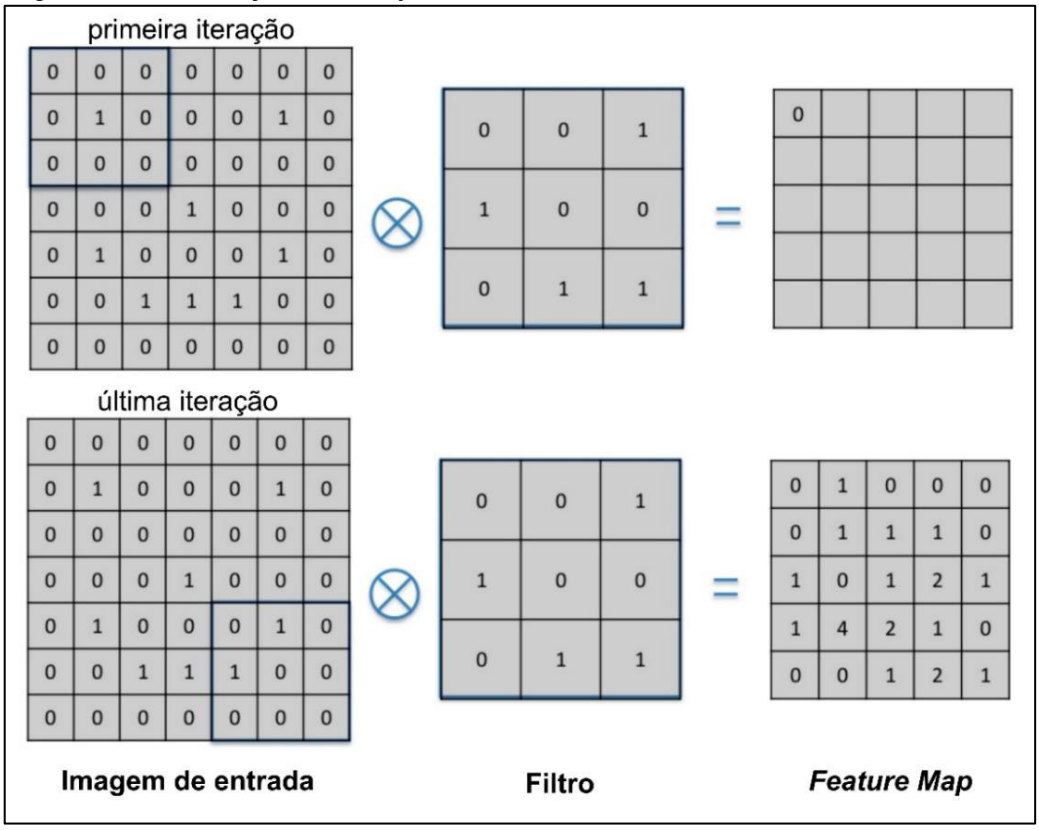
$$c[t] = (a * b)[t] = \sum_{j=-\infty}^{\infty} a[j] b[t - j] \tag{16}$$

Quando inspecionada por um prisma puramente matemático, a resultante da convolução expressa a maneira como uma função é modificada por outra (SUPERDATASCIENCE TEAM, 2018). No contexto de *deep learning*, essa ideia é preservada. A convolução dispõe de 3 (três) elementos:

1) Arranjo de entrada (*input array*), podendo ser uma imagem ou uma sequência de dados. Representa a função $a[j]$.
2) Filtros (*feature detectors* ou *kernels*), possuindo uma dimensão menor que o arranjo de entrada. Representa a função $b[t-j]$.
3) Mapas de características (*feature maps* ou *activation maps*). Representa a função resultante $c[t]$ da operação.

Na primeira iteração do procedimento, são tomados o filtro e um primeiro segmento do *input array*. Então, é aplicada uma multiplicação elemento a elemento (*elementwise multiplication*), isto é, multiplicações entre os elementos correspondentes (de mesmo índice) das duas matrizes. Ao final, a soma de todos os produtos é armazenada na primeira célula do *feature map*. Essa etapa é mostrada na parte superior da Figura 19.

Figura 19 – Convolução do arranjo de entrada



Fonte: adaptado de SuperDataScience Team (2018, p. 14)

O filtro, então, se desloca entre as colunas do *input array* (e, quando estas acabam, entre as linhas) a cada iteração, repetindo o procedimento até que o filtro

percorra todos os segmentos do *input array* e que o *feature map* seja totalmente preenchido. A última iteração está ilustrada na parte inferior da Figura 19. Nessa figura, o símbolo Ⓧ denota a operação de convolução.

A finalidade dos filtros é selecionar as características que contribuem para a correta classificação/predição e remover as que não tem utilidade. Por exemplo, em um problema de reconhecimento visual de rostos, o cérebro humano tenta perceber a presença e o posicionamento de olhos, nariz, boca, a simetria deles, entre outros elementos. A ideia é não analisar cada célula isoladamente. Então, a categorização holística é mais prática e eficaz. Neste trabalho, a *1D-CNN* possuiu 64 filtros em todas as suas 3 camadas de convolução, conforme exposições na Seção 6.3.

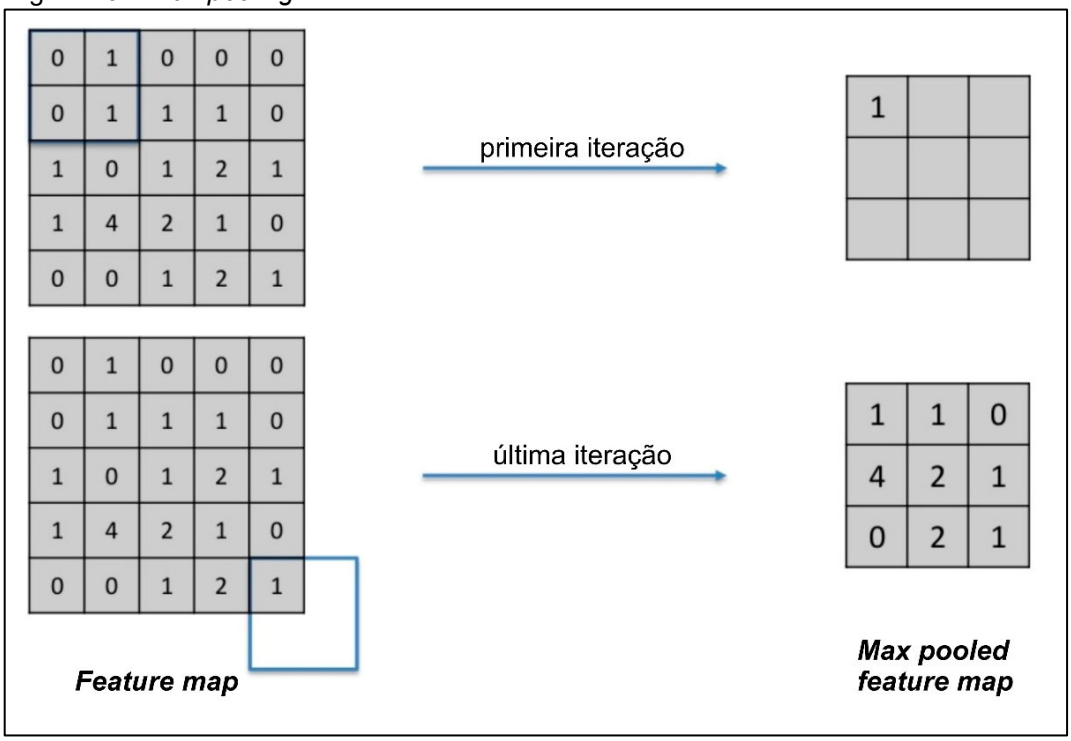
Para que se consiga identificar e categorizar adequadamente um arranjo de entrada, uma rede convolucional deve ser hábil a lidar com alguns tipos de transformação, como rotações, compressões ou extensões na dimensão desse arranjo. Por isso, é interessante a implementação do *pooling*.

4.3.2 Max pooling

Essa operação é responsável por incorporar à *CNN* uma propriedade denominada de invariância espacial (BUDUMA; LACASCIO, 2017). O *max pooling* é realizado pela operação de um *pool array* nos *feature maps*. O procedimento é resumido na Figura 20 e é similar à convolução, com uma diferença: em vez da multiplicação ponto a ponto, o *max pooling* armazena apenas o dado de maior valor.

Além disso, ao final da aplicação do *max pooling*, o mapa resultante tem seu tamanho drasticamente reduzido, em comparação ao arranjo original. Isso significa que, igualmente à convolução, o *max pooling* descarta informações e características desnecessárias, sem as quais a rede pode trabalhar de forma mais eficiente (SUPERDATASCIENCE TEAM, 2018). Isso aprimora a *CNN*, uma vez que mitiga o sobreajuste (*overfitting*) e intensifica a velocidade de processamento ao se evitar usar uma quantidade maior de pesos sinápticos. No *overfitting*, o desempenho da rede é demasiadamente inferior com novas amostras, em comparação ao obtido com as amostras de treinamento. Esse evento é melhor conceituado na Seção 4.4 e na Figura 26. Neste trabalho, a *1D-CNN* também possuiu 64 *pool arrays* em todas as suas 3 camadas de *max pooling* (Apêndice E).

Figura 20 – *Max pooling*



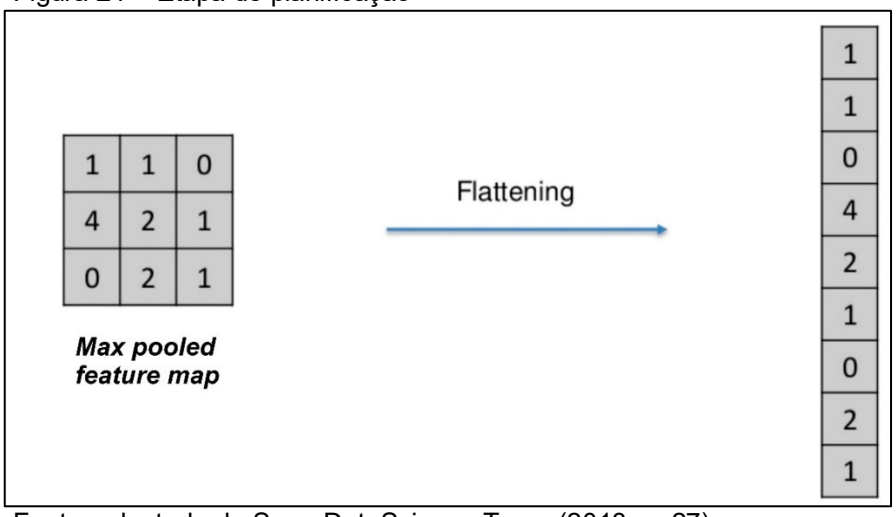
Fonte: adaptado de SuperDataScience (2018, p. 23)

O procedimento subsequente é a planificação.

4.3.3 Flattening

A etapa exibida na Figura 21 simplesmente consiste em planificar os *pooled feature maps*, tornando-os em um único vetor de longo comprimento. Isso é feito como uma preparação para o estágio seguinte, a *fully connection*.

Figura 21 – Etapa de planificação

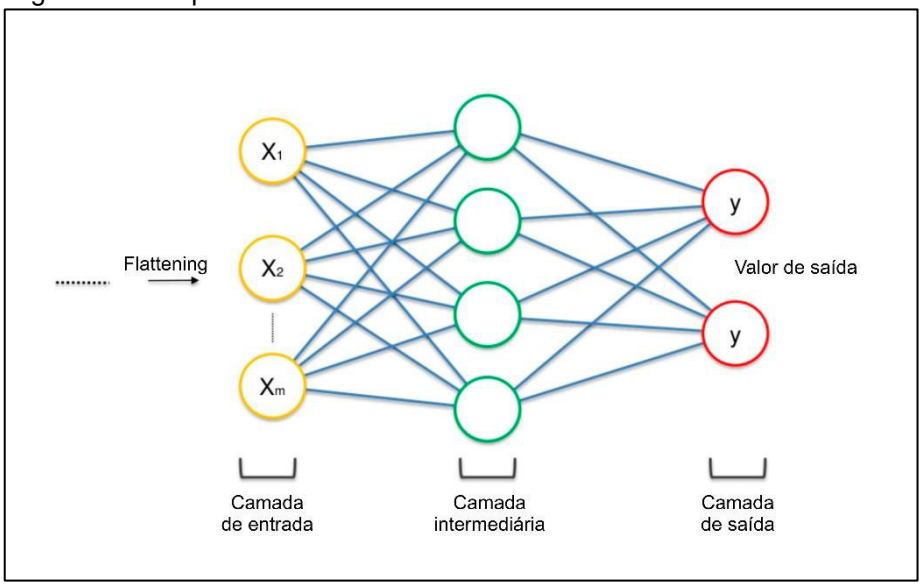


Fonte: adaptado de SuperDataScience Team (2018, p. 27)

4.3.4 Full Connection

Nesta fase, um *MLP*, englobado pela rede convolucional, é alimentado com os dados da camada planificada, operando da maneira descrita na Seção 4.2.

Figura 22 – Etapa de conexão total



Fonte: adaptado de SuperDataScience Team (2018, p. 28)

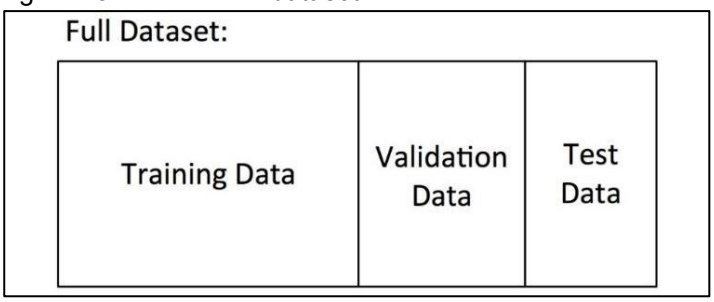
Apesar de o funcionamento descrito se referir ao caso bidimensional, a essência desses princípios é mantida no estudo de uma rede convolucional unidimensional. Esse modelo neural, a fim de realizar a tarefa de regressão, interpretada neste trabalho como o mapeamento entre sinais temporais de modelos dinâmicos e seus respectivos parâmetros, deve primeiro ajustar seus pesos sinápticos às amostras do problema. Isso só é possível por meio de um aprendizado eficaz da correlação entre os dados de entrada e de saída. Dessa forma, o processo de aprendizagem é retratado na próxima seção.

4.4 Processo de Aprendizagem

De um modo geral, para realizar a aprendizagem, uma RNA precisa ser alimentada com um conjunto de dados, chamados de amostras (ou objetos). Elas, por sua vez, são divididas em atributos de entrada e rótulos (*labels*) (FACELI *et al.*, 2011). Os atributos de entrada são as características da amostra, que são computados para efetuar a aprendizagem de sua correlação com o rótulo.

As amostras compõem um grande conjunto, chamado de *dataset*, que pode ser dividido em 3 (três) subconjuntos (BUDUMA; LACASCIO, 2017): um de treinamento (*training set*), um de validação (*validation set*) e outro de teste (*test set*), como sugere a Figura 23. Vale destacar que os subconjuntos são mutuamente exclusivos, isto é, as amostras contidas em algum subconjunto não estão contidas em nenhum outro.

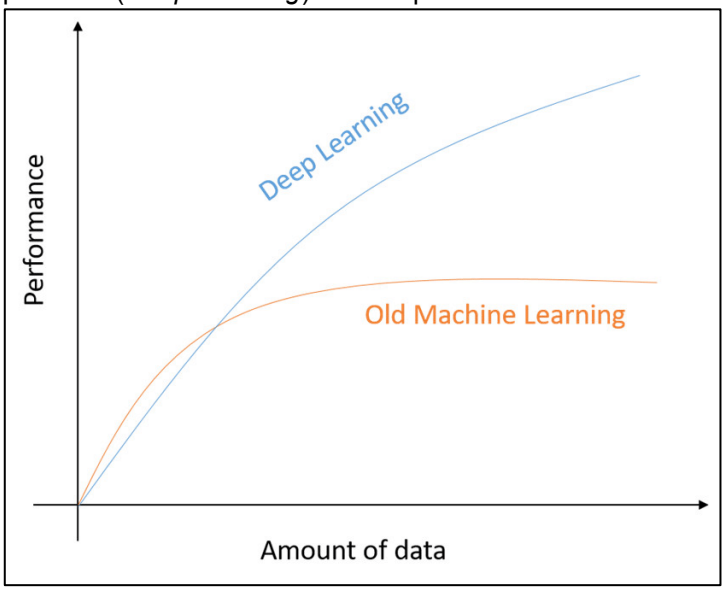
Figura 23 – Divisão do *dataset*



Fonte: Buduma e Lacascio (2017, p. 32)

Por ser um sistema de *deep learning*, o desempenho da *1D-CNN* aumenta à medida que o número de amostras no *dataset* também aumenta (ALOM *et al.*, 2019; MAHESWARI, 2018), o que é retratado na Figura 24.

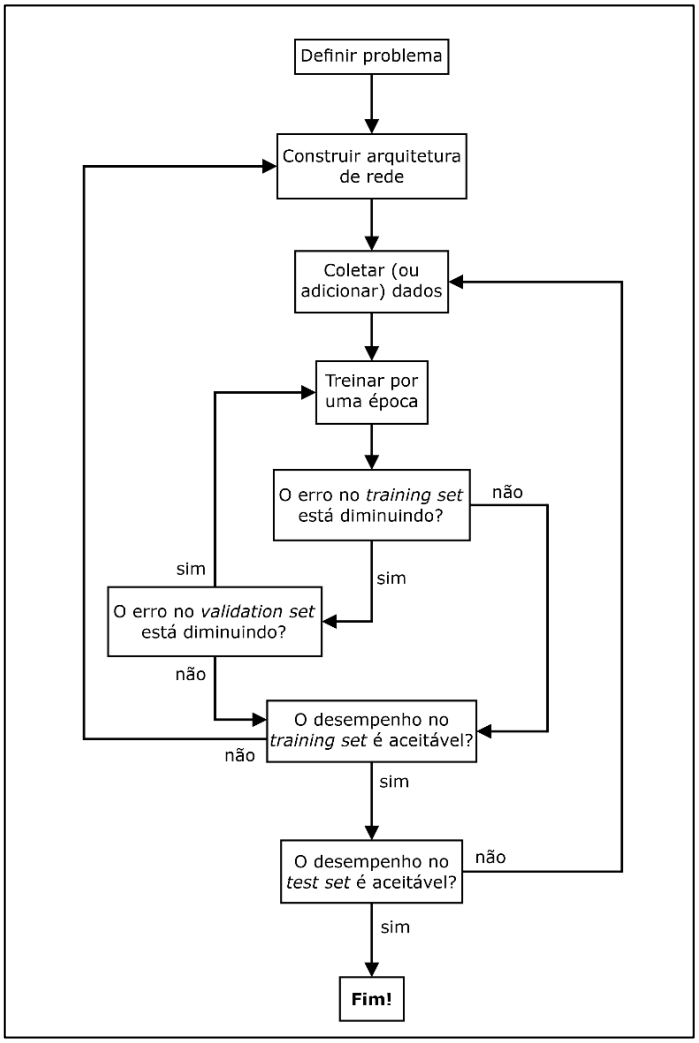
Figura 24 – Gráfico de desempenho da aprendizagem profunda (*Deep Learning*) *versus* quantidade de amostras



Fonte: Alom *et al.* (2019, p. 7)

O *training set* são as amostras que são processadas no treinamento para ajustar os pesos sinápticos, com o auxílio de um otimizador e do *backpropagation* (Seção 4.4.1), efetivamente contribuindo para a aprendizagem. O *test set* é a amostra utilizada no teste do desempenho da rede neural. O *validation set*, por sua vez, são as amostras utilizadas durante o treinamento para detectar eventuais ocorrências de sobreajuste (*overfitting*), servindo como ferramenta de diagnóstico da generalização. Em outras palavras, serve para verificar o progresso do desempenho da *1D-CNN* em outras amostras, que não pertençam ao *training set*. A atuação dos subconjuntos supracitados é ilustrada na Figura 25 a seguir.

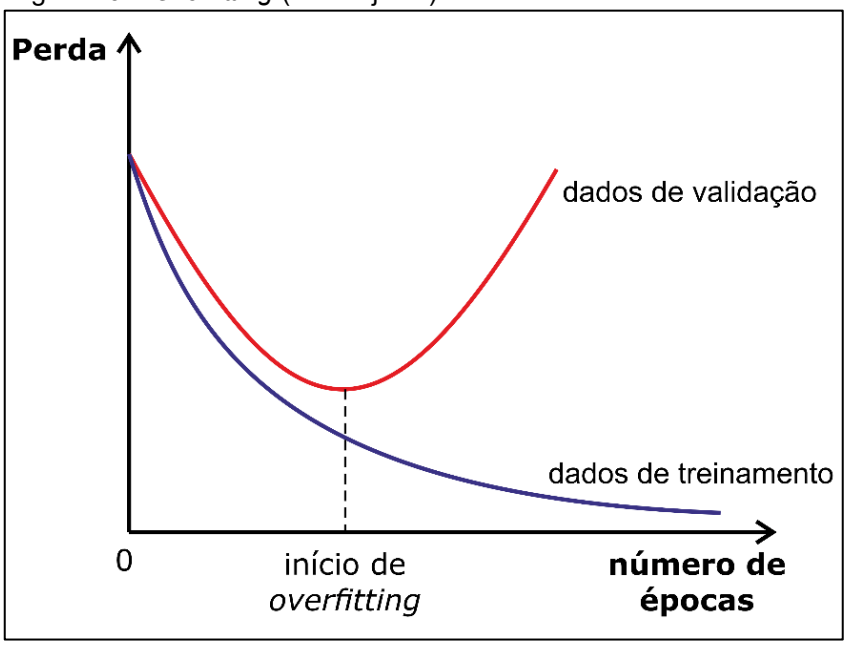
Figura 25 – Treinamento, validação e teste de uma rede neural artificial



Fonte: adaptado de Buduma e Lacascio (2017, p. 33)

O *overfitting* ocorre quando a rede neural ajusta exageradamente bem seus pesos sinápticos às amostras do *training set*, mas apresenta uma performance drasticamente inferior quando avaliada com outras amostras (GOODFELLOW; BENGIO; COURVILLE, 2016), sendo portanto um fenômeno bastante indesejado. Ele é detectado quando a perda no *training set* continuar a diminuir ao longo do treinamento, ao passo que a perda no *validation set* começa a aumentar, como mostra o gráfico da Figura 26. A perda representa o erro entre valores reais (rótulos) e valores estimados com a rede (predições), sendo essa grandeza é melhor definida na Subseção 4.4.1. Em geral, "quanto mais profunda e internamente conectada for uma RNA, maior será o risco de ela exibir *overfitting*", de acordo com (BUDUMA; LACASCIO, p. 30). Por outro lado, quando uma rede apresenta um bom desempenho em ambos subconjuntos, afirma-se que ela possui a capacidade de generalização.

Figura 26 – *Overfitting* (sobreajuste)



Fonte: adaptado de Nabi (2018, p. 11)

A presença dos rótulos nas amostras revela que redes neurais são sistemas de aprendizagem supervisionada. Isso significa que os rótulos servem de ensinamento, na fase de treinamento, para que seja indicado à rede o resultado correto que se busca obter na predição da amostra correspondente. Em outras

palavras, o modelo neural tem como finalidade estimar uma saída $\hat{y}$ que seja o mais aproximada possível do rótulo y.

A forma como redes neurais efetuam a aprendizagem é baseada no emprego de otimizadores e do *backpropagation*. Esse assunto é descrito a seguir.

4.4.1 Backpropagation e otimizadores

Durante o treinamento, é efetuado um algoritmo de retropropagação de erro, conhecido como *backpropagation*. Essencialmente, esse mecanismo faz com que a rede compute saídas estimadas $\hat{y}$ e as compara com os rótulos correspondentes y, constituindo uma função de erro J, também chamada de função de perda (*loss function*) ou ainda função de custo (*cost function*). Em seguida, a informação sobre o erro é propagado pela rede no sentido inverso, ou seja, da saída em direção à entrada. À medida que a retropropagação acontece, os pesos de cada neurônio são ajustados, e o processo se repete até todo o *training set* ser processado em um número pré-estabelecido de épocas (*epochs*). A literatura menciona várias funções de perda, porém a utilizada nesta monografia é o erro quadrático médio (*MSE – Mean Squared Error*), especificado na Equação 17. Conforme o relatado na Seção 6.1, $M = 8000$.

$$J(\boldsymbol{w}) = MSE = \frac{1}{M}\sum_{n=1}^{M}(y_n - \widehat{y_n})^2 \tag{17}$$

em que:

- J é a função de perda (*loss*);
- $\boldsymbol{w}$ é o vetor de pesos sinápticos;
- $\widehat{y_n}$ é a saída estimada da n-ésima amostra da RNA;
- y_n é o rótulo (*label*) da n-ésima amostra da RNA; e
- M é a quantidade de amostras de treinamento.

Para implementar o *backpropagation*, é necessário utilizar algum algoritmo de otimização, sendo o mais tradicional chamado de Descida do Gradiente (*GD – Gradient Descent*). Porém, à medida que a complexidade da rede aumenta, mais pesos sinápticos são criados, o que eleva o número de dimensões do espaço em que a superfície de perda é analisada. Com isso, essa superfície se torna altamente

irregular, não convexa e abstrata, aumentando regiões de mínimos locais e por vezes dificultando a obtenção de um valor ótimo de perda. Portanto, neste trabalho, recorreu-se ao otimizador *Adam* (*Adaptive Moment Estimation*), que é apropriado quando a rede neural em questão possui uma quantidade numerosa de dados e de pesos sinápticos, sendo particularmente útil em superfícies de perda não convexas (KINGMA; BA, 2015).

Durante o treinamento da *1D-CNN*, foi percebido um obstáculo em relação à função de ativação escolhida. Essa questão é abordada logo em seguida.

4.4.2 Dying ReLU

A eficiência e a velocidade do cálculo dos gradientes são fortemente influenciadas pelas funções de ativação, uma vez que suas derivadas são calculadas a partir da aplicação da regra da cadeia nos gradientes. Por isso, a escolha da função se torna relevante; ela deve ser apropriada para a tarefa e preferencialmente de pouca complexidade. Nesse contexto, a *ReLU* surge como uma alternativa atraente, devido às vantagens explicitadas na Subseção 4.1.2. Ela apresenta, portanto, derivadas de valor constante unitário quando os argumentos são positivos e de valor nulo quando os argumentos são negativos (Equação 18).

$$\frac{df}{dz} = \frac{d}{dz}[max(0, z)] = \begin{cases} 0, & se\ z \leq 0 \\ 1, & se\ z > 0 \end{cases} \tag{18}$$

A derivada nula resulta em um neurônio desativado e, por conseguinte, em um gradiente nulo, impossibilitando que os pesos do neurônio sejam atualizados. Ocasionalmente, quanto maior for a profundidade de uma rede neural, maior será a chance de alguns neurônios com ativação *ReLU* receberem repetidamente argumentos negativos, de modo que permanecem desativados (ou “mortos”) indefinidamente, o que caracteriza o problema de “*Dying ReLU*” (LU *et al.*, 2019; GOODFELLOW; BENGIO; COURVILLE, 2016). Como consequência da “morte” dos neurônios, parte da rede passa a não mais contribuir para a aprendizagem, prejudicando sua capacidade de solucionar a tarefa. Esse problema surgiu no início dos experimentos, porquanto a *1D-CNN*, até então, apresentava *ReLU* em suas camadas internas. Essa adversidade pôde ser mitigada com a redução da taxa de aprendizagem (*learning rate*) lr e a substituição da *ReLU* pela ativação *Leaky ReLU*

(LIU, 2017), garantindo que os gradientes não se tornassem nulos e ainda mantendo a não linearidade.

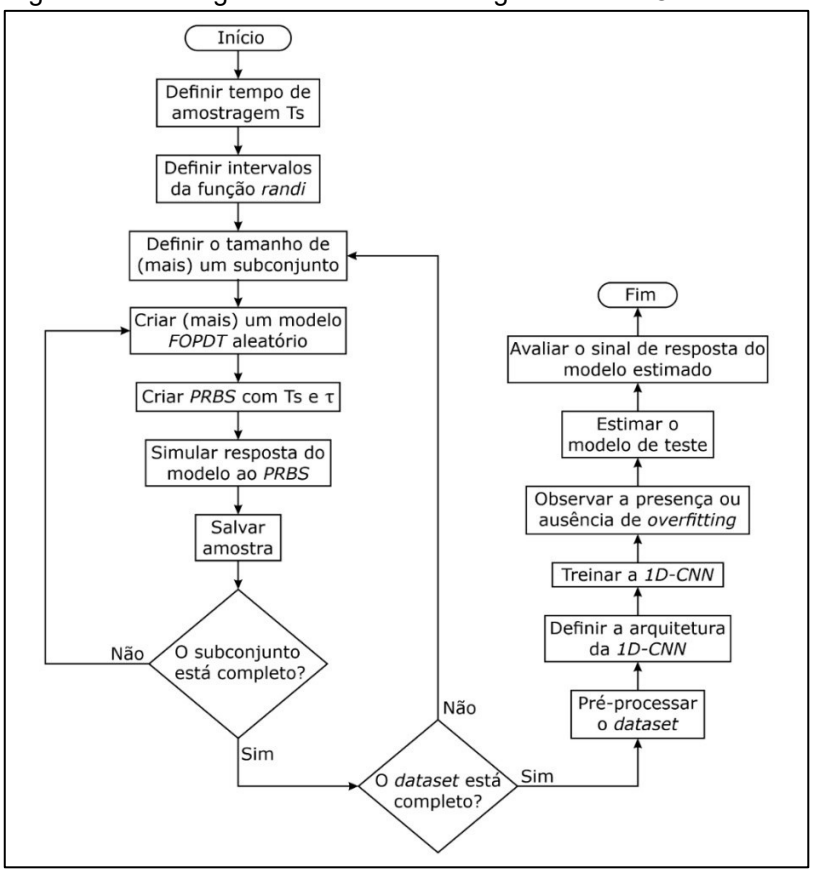
O próximo capítulo explica o procedimento para a estimação do modelo dinâmico, aplicando os fundamentos que foram descritos desde o primeiro capítulo.

5 METODOLOGIA

Para conseguir uma *1D-CNN* completamente treinada, são executadas simulações em dois ambientes de desenvolvimento integrado (*IDE – Integrated Development Environment*): *MATLAB*® e *Spyder*®. Este último é um software escrito em linguagem *Python* e está incorporado na distribuição de código aberto *Anaconda*®.

Os *softwares* são usados para executar os códigos-fontes do Apêndice Apêndice B (*MATLAB*®), Apêndice C (*MATLAB*®) e Apêndice D (*Spyder*®), e a função de todos eles pode ser resumida no fluxograma da Figura 27. Com os dois primeiros códigos-fontes são produzidos o *training set* e o *validation set*. Com o código-fonte *Python* todo o *dataset* (incluindo a amostra de teste, obtida no ensaio da incubadora) é pré-processado. Em seguida, a rede *1D-CNN* é estruturada, treinada e validada. Por último, os parâmetros do modelo dinâmico da incubadora são estimados.

Figura 27 – Fluxograma-resumo dos códigos *MATLAB*®



Fonte: próprio autor

Neste capítulo são abordados: a criação de modelos dinâmicos lineares; a obtenção de seus sinais de resposta, para formar o *dataset*; as instruções do ensaio da incubadora; o pré-processamento das amostras; e as funcionalidades e bibliotecas empregadas no treinamento, na validação e no teste da *1D-CNN*.

5.1 Ambiente *MATLAB*®

Nesta monografia, como exposto na Tabela 1 e na Tabela 2, as amostras apresentam 3 (três) atributos de entrada, cada um correspondendo a um tipo distinto de série temporal (sinal), e 3 (três) rótulos, cada um correspondendo a um parâmetro de modelos *FOPDT* (K, τ, θ). Enfatiza-se, contudo, uma particularidade: especificamente a amostra de teste (originada da incubadora) não possui rótulo, que é justamente o fato que motiva a sua identificação. Além disso, a análise de desempenho é enfocada nos sinais de resposta resultantes dos modelos, e não exatamente nos valores numéricos do rótulo e da saída estimada. Essa questão também é mencionada na Seção 6.4.

Tabela 1 – Formato do atributo de entrada

1º sinal	2º sinal	3º sinal
Tempo discreto	Sinal *PRBS*	Sinal de resposta

Fonte: próprio autor

Tabela 2 – Formato do rótulo (valor real de saída)

1º sub-rótulo	2º sub-rótulo	3º sub-rótulo
Ganho estático ($\boldsymbol{K}$)	Constante de tempo ($\boldsymbol{\tau}$)	Atraso ($\boldsymbol{\theta}$)

Fonte: próprio autor

Neste momento, o *training set* e o *validation set* são construídos por simulação no *software MATLAB*®. Vários modelos *FOPDT* são criados, fazendo com que cada parâmetro assuma aleatoriamente valores dentro de intervalos distintos, o que é retratado no código-fonte do Apêndice B. Eles são quantificados com início (i), fim (f) e passo (p), como mostrado a seguir.

$$v_K = [i_K, i_K + p_K, i_K + 2.p_K, i_K + 3.p_K, \dots, f_K]$$
$$v_\tau = [i_\tau, i_\tau + p_\tau, i_\tau + 2.p_\tau, i_\tau + 3.p_\tau, \dots, f_\tau] \quad (19)$$
$$v_\theta = [i_\theta, i_\theta + \theta, i_\theta + 2.p_\theta, i_\theta + 3.p_\theta, \dots, f_\theta]$$

Em seguida, é realizada a simulação dos modelos dinâmicos lineares, em que são aplicados a eles sinais de entrada, e deles obtidos sinais de resposta.

Com isso, as amostras a serem submetidas à *1D-CNN* são formadas pela concatenação de 3 (três) sequências temporais de cada modelo, da maneira ilustrada na Tabela 1, e os rótulos correspondentes são agrupados como na Tabela 2. Após a formação de cada amostra, ela é salva em um arquivo de extensão .csv (*Comma-Separated Values*), de acordo com o código-fonte do Apêndice C.

Por fim, as amostras geradas são divididas entre os subconjuntos de treinamento (*training set*) e validação (*validation set*), tal qual sugere o procedimento da Seção 4.4. Conforme o que foi supracitado nesta subseção, a aprendizagem da rede convolucional é feita baseando-se nas amostras do *training set*, e a estimação na amostra do *test set*. Em razão disso, e em consonância com a explicação da Seção 3.3, a validação não é empregada da maneira descrita naquela seção. A aquisição de um outro sinal de resposta medido na incubadora ou seu fracionamento se mostrariam redundantes pelos seguintes motivos:

1) O próprio funcionamento das RNAs abrange a precaução referente aos dados de teste. De acordo com explicações anteriormente expostas, o sinal de resposta da incubadora não é usado para construir seu modelo dinâmico, e sim apenas para avaliá-lo. O modelo dinâmico, na verdade, é construído por meio do aprendizado de uma rede neural que foi treinada com as respostas de 8.000 modelos dinâmicos lineares oriundos de simulação em *software*. Além disso, (AGUIRRE, 2019b) afirma que “o importante da validação do modelo dinâmico é testá-lo com dados que nunca foram analisados pelo modelo”, o que já é feito com o *validation set*.
2) Utilizar os dados advindos da incubadora neonatal no treinamento não seria sequer possível, devido à ausência de rótulos e, consequentemente, de gradientes de erro (Subseção 4.4.1). Ademais, ainda que fosse possível, a adição de apenas 1 (uma) amostra no treinamento teria uma

variação insignificante no desempenho. Uma melhora perceptível seria eventualmente alcançada com a adição de centenas ou milhares de amostras medidas, sendo inviável devido aos recursos e tempo despendidos nos ensaios de laboratório e à limitação da capacidade de processar um número muito maior de amostras, tendo em vista as especificações do PC utilizado (Quadro 1).

Com efeito, os tamanhos dos subconjuntos e as respectivas pastas de destino para armazenamento são previamente definidos no código-fonte do Apêndice C, por razões de melhor organização.

Os dados criados até o momento são originados das simulações de resposta no *software*. Doravante, passa-se a adquirir a amostra de teste. Essa amostra é aquela obtida pela experimentação com a incubadora neonatal.

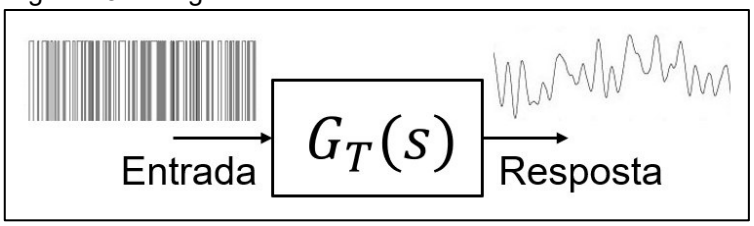
5.2 Ensaio da Incubadora Neonatal

Este estágio se refere à obtenção e preparação de dados no formato da Tabela 1, porém oriundos das medições feitas na incubadora.

Inicialmente, implementa-se a identificação por otimização, discutida na Seção 3.4, com o propósito de extrair um valor aproximado da constante de tempo τ, para usar na Equação 6 e assim projetar o sinal de entrada *PRBS*. A ordem dos sinais n e o comprimento N tem seus valores definidos na Seção 6.1 e no código-fonte do Apêndice B.

Com o sinal construído, são permitidas a execução do ensaio e, consequentemente, a aquisição dos dados de resposta desejados, da maneira ilustrada na Figura 28.

Figura 28 – Diagrama do ensaio da incubadora



Fonte: adaptado de Borjas e Garcia (2012, p. 212)

De posse de todos os dados necessários, o ambiente de treinamento da rede neural começa a ser preparado.

5.3 Ambiente *Spyder*®

Esta seção é dividida em 2 (duas) partes: a primeira apresenta o pré-processamento; e a segunda, a configuração dos elementos da rede convolucional e o processamento do *dataset*.

O *software* de linguagem *Python*, *Spyder*®, é destinado a propriamente construir a arquitetura da *1D-CNN*, treiná-la e gerar suas predições.

Inicialmente, uma preparação dos dados é realizada precedentemente, a qual é conhecida como pré-processamento. Ele é necessário para compatibilizar o *dataset* em termos de agrupamento, disposição e padronização de suas amostras para a alimentação adequada da rede.

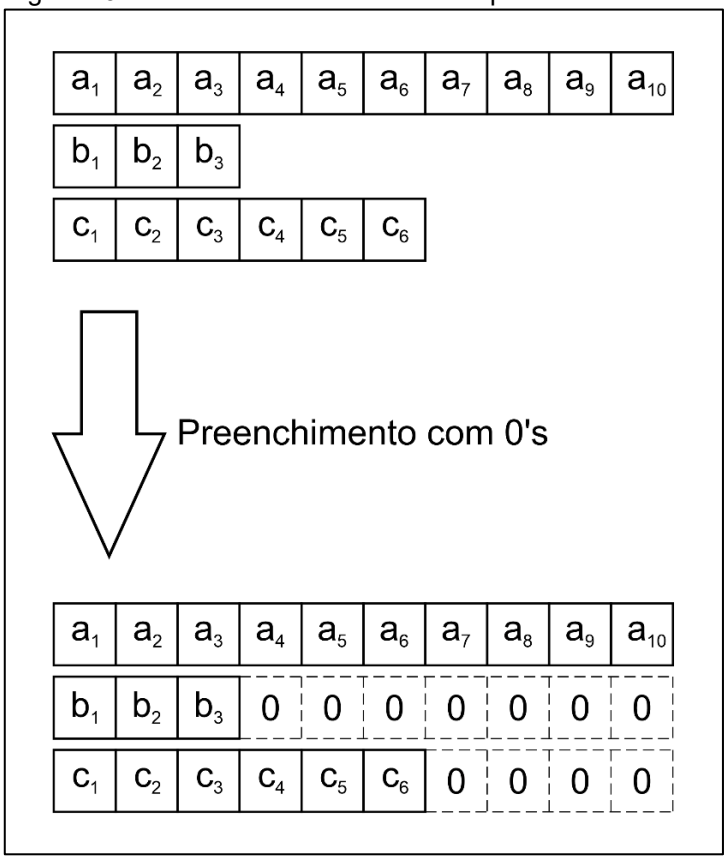
5.3.1 Pré-processamento

São aplicados três procedimentos ao *dataset*: o preenchimento (*padding*), o redimensionamento e o *feature scaling*.

5.3.1.1 Padding

Devido ao uso de um período fixo de amostragem T_S e à grande variedade de constantes de tempo, as durações dos sinais se tornam bastante diversificadas, o que causa diferenças significativas entre os tamanhos dos sinais de entrada do *dataset*. Isso se torna inconveniente, uma vez que os atributos de entrada dos subconjuntos precisam ser posteriormente agrupados em tabelas e matrizes no *software Spyder*®, o que não é possível diante de sequências com comprimentos diferentes. A fim de resolver esse empecilho, realiza-se o preenchimento dos dados com zeros. Primeiro, descobre-se qual o maior comprimento de todo o *dataset* para que ele seja utilizado como referência. Então, são adicionados aos dados restantes uma quantidade suficiente de zeros, de modo que isso resulte em um conjunto de dados de tamanhos idênticos. A Figura 29 ilustra a operação.

Figura 29 – Preenchimento de dados sequenciais



Fonte: Próprio autor

5.3.1.2 Redimensionamento

Efetivamente, essa técnica é aplicada no momento em que os dados estão devidamente agrupados em uma estrutura composta. O intuito é organizar as estruturas matriciais correspondentes aos subconjuntos em três dimensões, da seguinte maneira:

$$formato = (n° \; de \; amostras; \; n° \; de \; avanços \; de \; tempo \; discreto; \; n° \; de \; canais)$$

No ambiente *Spyder*®, a estrutura é denominada da seguinte forma (KERAS, 2020):

$$shape = (samples, timesteps, feature)$$

Vale ressaltar que o número de avanços corresponde ao comprimento da amostra que tem a maior duração, uma vez que, nesse momento, o *padding* é feito na etapa anterior. Ademais, o número de canais se refere à quantidade de sinais agrupados em cada uma amostra. Nesta aplicação, conforme a Tabela 1, foram 3

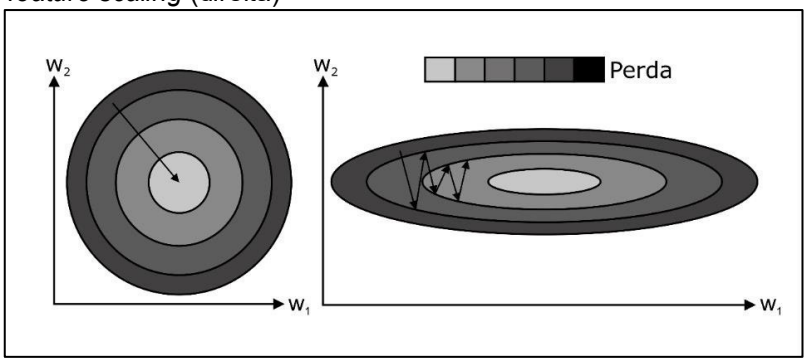
sinais. Com efeito, os canais representam os *features*, visto que cada um expressa uma certa característica da amostra a ser analisada.

Esse procedimento é obrigatório ao se trabalhar com séries temporais na *Keras*, porque muitos modelos montados com a biblioteca requerem que os dados tenham o formato supracitado para que seu processamento ocorra.

5.3.1.3 Feature Scaling

Desde a obtenção do *dataset*, estão presentes discrepâncias de escala entre as três sequências temporais e entre os três valores dos rótulos, o que é compreensível, já que cada sequência e parâmetro manifesta um sentido físico próprio. Contudo, essas diferenças de magnitude, a depender do seu grau, pode dificultar ou até mesmo impossibilitar o atingimento de uma boa performance nas estimações (GÉRON, 2017). A existência de *features* com escalas diferentes provoca a deformação da superfície de perda de tal modo a impedir os gradientes de apontarem para as regiões de valores mínimos de perda, fazendo com que eles sequer sejam alcançados ou causando o aumento do tempo necessário para o treinamento. Para contornar esse problema, transformam-se os dados com o intuito de colocá-los em uma escala unificada. Essa é uma das técnicas mais importantes e mais necessárias, sendo denominada de *feature scaling* (GÉRON, 2017). A seguir, na Figura 30, pode ser feita uma comparação entre as curvas de nível das superfícies quando se dispõe de dados transformados (à esquerda) e não transformados (à direita).

Figura 30 – Curvas de nível de redes com (esquerda) e sem *feature scaling* (direita)



Fonte: adaptado de Géron (2017, p. 113)

O procedimento é realizado para cada tipo de sequência que houver, cada um correspondendo a um canal. Conforme a Tabela 1, os tipos são: avanço de tempo discretizado, sinal *PRBS* e sinal de resposta a *PRBS*. Assim, o procedimento é realizado 3 (três) vezes. Inicialmente, tomam-se todas as sequências temporais referentes apenas a um dos canais. Então, para cada posição (*timestep*), os valores são transformados para obedecer a uma determinada escala. Existem duas maneiras de fazer essa modificação:

1) **Normalização (*Normalization*):** Consiste em aplicar a Equação 20, de modo a deixar os valores no intervalo [0, 1]. No *scikit-learn*, é implementado pela classe *MinMaxScaler*.

$$q^* = \frac{q - q_{min}}{q_{max} - q_{min}} \tag{20}$$

2) **Padronização (S*tandardization*):** Consiste em calcular a média (μ) e o desvio padrão (σ) dos valores considerados, aplicando a Equação 21. No *scikit-learn*, é implementado pela classe *StandardScaler*.

$$q^* = \frac{q - \mu}{\sigma} \tag{21}$$

É importante enfatizar que, a partir desse estágio, os dados já não são mais inteligíveis, podendo inclusive assumir valores negativos, quando alterados pela Equação 21. Apesar disso, a rede encontra benefícios valiosos em analisar dados com essas propriedades, como já explicitado. Além disso, o método também é realizado separadamente para os rótulos, visto que os três parâmetros possuem magnitudes diferentes. Assim, em uma regressão de múltiplas saídas, em que estas não se submeteram a um *feature scaling*, a otimização da perda ocorreria de modo mais priorizado em relação a um dos parâmetros, em detrimento dos outros.

Foi escolhida a padronização (*StandardScaler*) para a consecução dos resultados, porque, durante algumas simulações, a normalização (*MinMaxScaler*) equivocadamente transformou valores significativos dos sinais em zeros. Isso foi observado especialmente ao selecionar-se aleatoriamente, no código-fonte da Apêndice B, valores muito pequenos para o ganho estático K nos rótulos.

5.3.2 Configurações e processamento

Posteriormente, a *1D-CNN* é criada, com a definição de sua arquitetura e seus hiperparâmetros (otimizador, função de perda, taxa de aprendizagem, número de filtros, coeficiente angular da *Leaky ReLU*, entre outros). Após, ocorre o treinamento, etapa mais demorada do processo, na qual os pesos são reiteradamente ajustados a fim de atingir os valores mínimos, relativos a um ponto ótimo, da superfície multidimensional de perda. É também estabelecido previamente o número de épocas (*epochs*) pelas quais o treinamento acontece. Além disso, é construído um gráfico de valor de perda no *training set* e no *validation set* ao longo das épocas, tal qual o da Figura 26, a fim de diagnosticar principalmente a ocorrência de *overfitting*. Por fim, a rede gera as estimações de todo o *dataset*, e seu desempenho pode então ser verificado.

Para a cumprimento de tais atividades, é fundamental o uso de funções de algumas extensões, sendo elas:

- ***Numpy*:** pacote que fornece muitas operações eficientes de computação numérica para estruturas compostas, como *arrays* e matrizes multidimensionais (NUMPY, 2020).
- ***Pandas*:** pacote que reúne artifícios para análise de dados e manipulação de estruturas compostas (PANDAS, 2020).
- ***Matplotlib*:** biblioteca orientada para a construção de gráficos bidimensionais a partir de estruturas compostas (MATLPLOTLIB, 2020).
- ***Keras*:** biblioteca destinada a elaborar e treinar modelos de *deep learning*, com suporte padrão para redes convolucionais e recorrentes (CHOLLET, 2018). É dela que provém a maioria das funcionalidades presentes no trabalho. Ademais, utiliza a plataforma *TensorFlow* como *backend*.
- ***TensorFlow:*** plataforma de código aberto criado pela *Google* voltada para *machine learning* e *deep learning* (TENSORFLOW, 2020).
- ***scikit-learn*:** módulo usado para aplicações em *machine learning* (SCIKIT-LEARN, 2020).

Keras, da mesma maneira que outras bibliotecas de *deep learning*, como é o caso da *Caffe* (CAFFE, 2020), possibilita a execução de seus códigos com a unidade de processamento gráfico (*GPU – Graphics Processing Unit*) de um

computador, em vez da tradicional unidade central de processamento (*CPU – Central Processing Unit*). Essa é uma grande vantagem, já que *GPUs* lidam especificamente com processamento paralelo de estruturas de dados compostas (AMORIM; ARAÚJO, 2011). Na prática, a execução do treinamento e de atividades correlatas é muito mais veloz. Todavia, até a data da conclusão deste trabalho, *Keras* somente oferece suporte para placas gráficas da NVIDIA®. Detalhes sobre a funcionalidade de *GPUs* e a razão de sua melhor performance em determinadas atividades podem ser encontrados, por exemplo, em (PACHECO, 2019) e (CAULFIELD, 2009). O processo de habilitação da *GPU* é concomitante à instalação do *TensorFlow* e foi seguido conforme as instruções em (TENSORFLOW, 2020).

O procedimento metodológico foi completamente especificado. No capítulo seguinte, é abordada a execução dos passos descritos e são analisados os resultados.

6 EXPERIMENTOS E RESULTADOS

Neste momento, o roteiro do Capítulo 5 é implementado. Isto é, as amostras são geradas por simulações e pelo ensaio de resposta da incubadora ao *PRBS*, sendo os sinais do *dataset* posteriormente pré-processados. Depois, a construção da arquitetura da *1D-CNN* e seu treinamento são efetuados, e os resultados são obtidos, avaliados e discutidos.

Todo o experimento, com exceção da Seção 6.2, é implementado por meio de códigos-fontes nos *softwares MATLAB*® e *Spyder*® (Apêndice B, Apêndice C e Apêndice D), podendo ser resumido com o fluxograma da Figura 27. Ademais, do mesmo modo do Capítulo 5, este capítulo se inicia com a criação das amostras no *software MATLAB*®.

6.1 Produção do *Dataset*

O processo de formação do *dataset* teve início com a definição e produção das amostras (Seção 5.1). O tempo de amostragem T_S escolhido nos experimentos foi de 12 (doze) segundos, por ser um processo consideravelmente lento. Além disso, o padrão adotado para os sinais de entrada foi o *PRBS*, descrito na Seção 3.5. Os sinais *PRBS* de entrada de cada modelo dinâmico simulado assumiram uma ordem $n = 7$, resultando em um comprimento $N = 127$ (Equação 5), e tiveram que ser projetados de acordo com o valor de sua constante de tempo τ correspondente, com a qual foi possível escolher um T_b (Equação 6). Efetivamente, os valores de T_b usados são a média dos limites inferior e superior daquela desigualdade, consoante a Equação 22. Essa rotina pode ser vista no código-fonte do Apêndice B.

$$T_b = \frac{1}{2}\left(\frac{\tau}{10} + \frac{\tau}{3}\right) \quad \rightarrow \quad T_b = \frac{13}{60}\tau \tag{22}$$

Essas definições são de suma importância para o funcionamento correto da *1D-CNN*, pois qualquer amostra que for gerada externamente ao *software*, como é o caso da amostra medida na incubadora, também deve obedecer a elas. Ou seja, se a uma rede devidamente treinada for submetida uma amostra cujo sinal de entrada não for uma sequência *PRBS* ou possuir um T_b em desconformidade com a Equação 22, a estimação pode ser comprometida.

Depois de estabelecidas essas diretrizes, o *dataset* começou a ser construído (Apêndice C). O *dataset* foi composto de 9.001 amostras, sendo o *training set* formado por 8.000, o *validation set* por 1.000 e o *test set* por apenas 1 (uma) amostra, que é coletada no ensaio da incubadora a *PRBS* (Seção 6.2). Primeiramente foram criadas as inúmeras funções *FOPDT* com a escolha aleatória de valores dos parâmetros dentro dos intervalos (espaço amostral) exibidos na Tabela 3. Os parâmetros selecionados compõem o rótulo do modelo *FOPDT* correspondente (Tabela 2)

Tabela 3 – Espaço amostral dos parâmetros

Parâmetro	Limite inferior	Limite superior	Passo
K (adim.)	0,01	5	0,01
τ (segundos)	120	1200	0,1
θ (segundos)	0	90	0,1

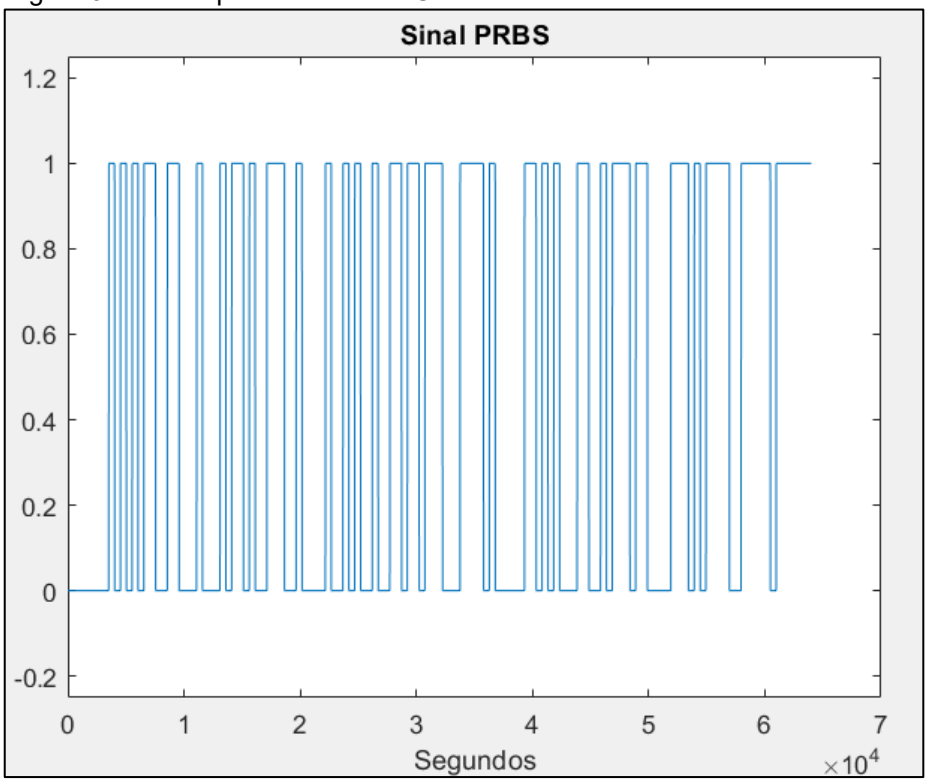
Fonte: próprio autor

Como observado nos códigos-fontes constantes do Apêndice B e do Apêndice C, a cada iteração, um sistema dinâmico foi criado, seu sinal *PRBS* foi construído com base no valor correspondente de τ, o tempo foi discretizado, a resposta do sistema ao *PRBS* foi determinada e os atributos de entrada (sinais) e os rótulos são armazenadas em arquivos *.csv* . Até esse momento do processo, o *dataset* contava com 9.000 amostras, satisfazendo a seguinte proporção:

- *Training set:* 8.000 amostras (88,89% do *dataset*)
- *Validation set:* 1.000 amostras (11,11% do *dataset*)

Além do *dataset*, também é salvo um arquivo *.csv* intitulado “n_amostras” que, contendo os tamanhos de cada subconjunto, é lido posteriormente com o código do Apêndice D. Exemplos de sinal *PRBS* e de sinal de resposta que compõem o *dataset* se encontram, respectivamente, na Figura 31 e na Figura 32.

Figura 31 – Exemplo de sinal *PRBS*



Fonte: próprio autor

Figura 32 – Exemplo de sinal de resposta a *PRBS*

Fonte: próprio autor

Os limites do espaço amostral da Tabela 3, assim como a arquitetura e os hiperparâmetros da *1D-CNN* (Seção 6.3), se justificam pela necessidade de conciliar 3 (três) aspectos da experimentação: o desempenho da estimação, o tamanho do *dataset* e o tempo necessário de treinamento. Em suma, como já mencionado, modelos de *deep learning* tendem a se tornar mais precisos à medida que processam mais amostras durante seu treinamento (ALOM *et al.*, 2019; MAHESWARI, 2018). Contudo, um *dataset* com uma ordem de centenas de milhares ou mesmo de milhões de amostras a ser treinado em um PC com as especificações listadas no Quadro 1 inviabiliza essa operação pela excessiva demora de sua conclusão.

Quadro 1 – Especificações técnicas do PC utilizado nas simulações

Ficha técnica	
Fabricante/Série/Modelo	ASUS X555LF
Ano	2015
Processador	Intel® Core i5 5200U
Memória RAM	8 GB
Placa Gráfica (*GPU*)	NVIDIA® GeForce 940m

Fonte: adaptado de ASUS (2020)

Ademais, o limite inferior escolhido para o intervalo de τ é escolhido levando em consideração o tempo de amostragem T_S , já que valores de constante de tempo τ muito pequenos podem prejudicar a captação da resposta transitória dos sistemas dinâmicos. Para evitar esse cenário, impõe-se que os modelos dinâmicos atendam o requisito da Equação 23.

$$T_S \leq 10.\tau \tag{23}$$

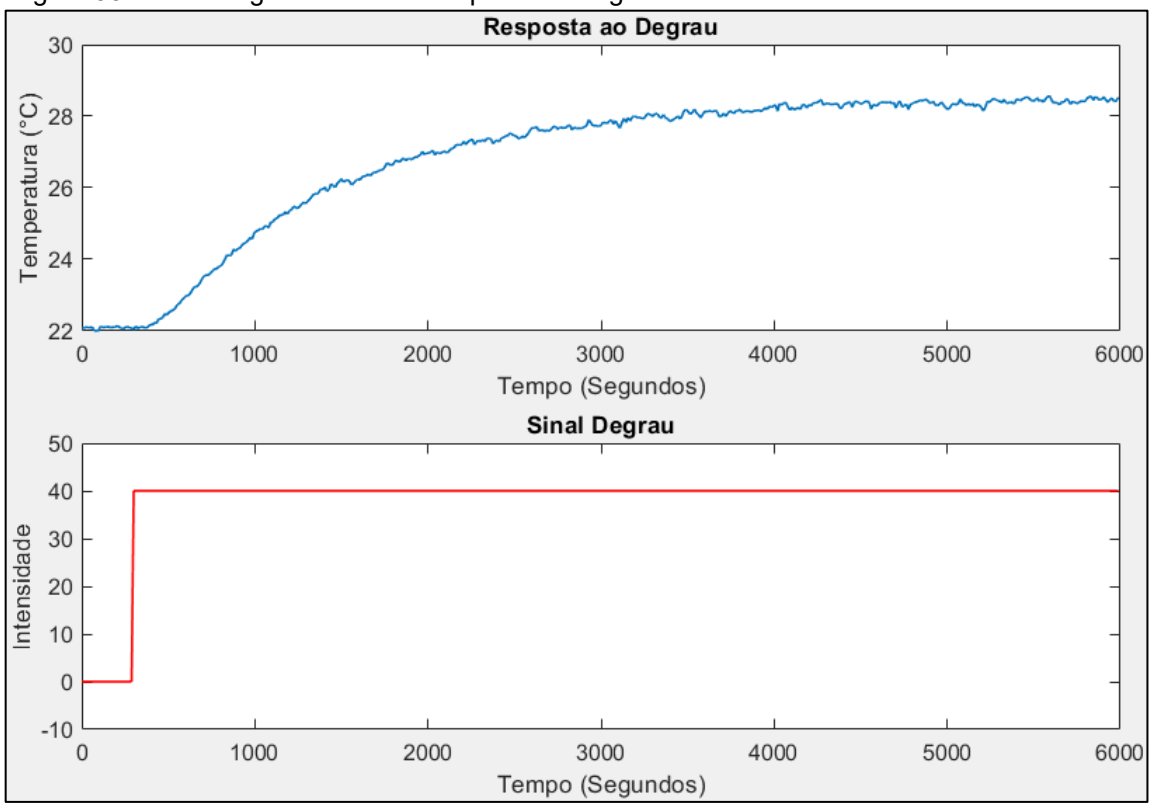
Para completar o *dataset*, passou-se à etapa de aquisição da amostra de teste, medida no sistema físico.

6.2 Produção da Amostra de Teste

Nesta aplicação, para testar o modelo-*CNN*, a amostra de teste é processada, e a resposta desse modelo dinâmico é comparado à resposta dessa amostra (Seção 6.2). Foi previamente escolhido o processo *SISO* de temperatura da incubadora para

a formação dos dados. Em seguida, iniciou-se a obtenção do modelo-*PRBS* com a identificação por otimização, empregando um sinal de degrau como entrada do sistema. Conforme a Figura 33, o degrau teve amplitude de 40% de *PWM*, e o sistema físico começou a operar em 22,06 °C e se estabilizou ao final em 28,52 °C.

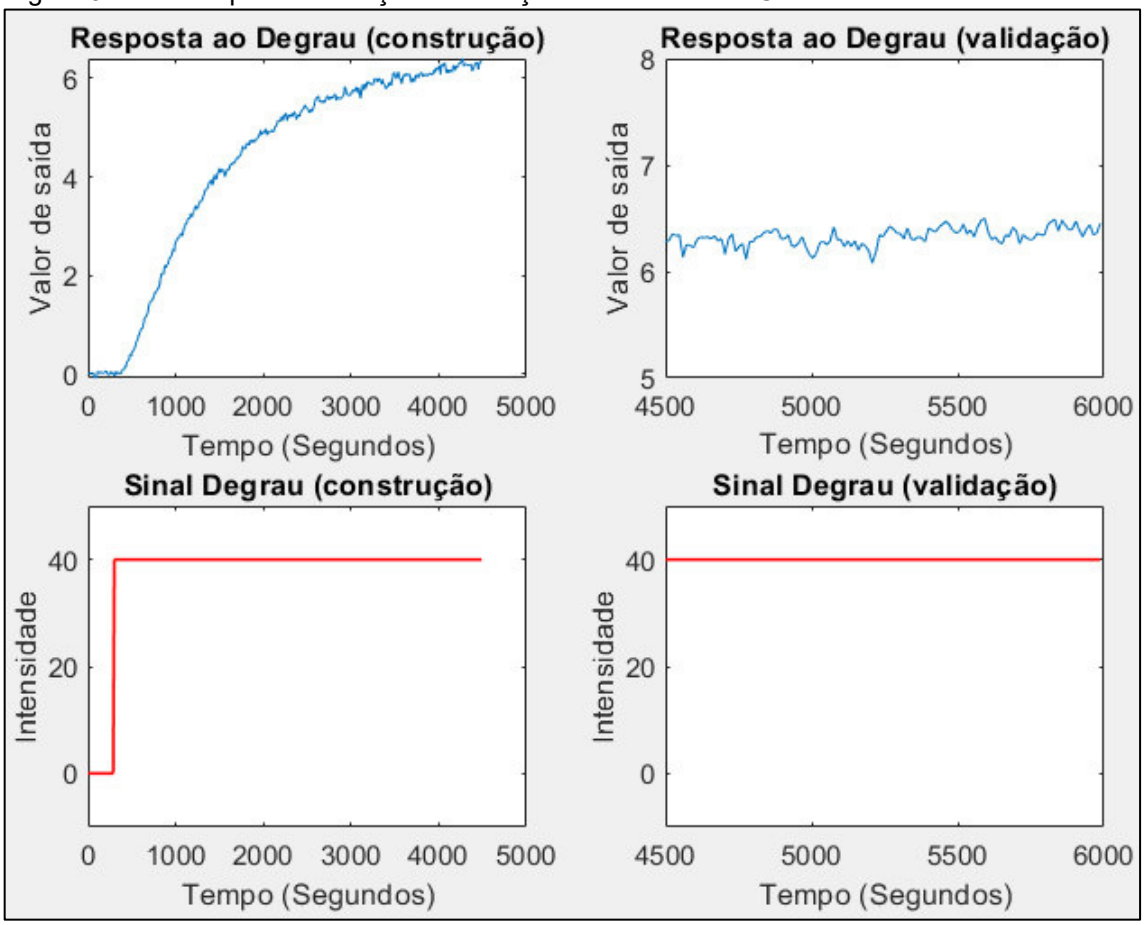
Figura 33 – Sinal degrau e sinal de resposta ao degrau



Fonte: próprio autor

Os sinais, então, foram segmentados em duas partes, conforme dita a Seção 3.3; uma parte (75% do total) é usada para construir o modelo-*PRBS*; e a outra (25% do total), para validá-lo (Figura 34).

Figura 34 – Sinais para construção e validação do modelo-*PRBS*



Fonte: próprio autor

Dessa forma, foi estimado o modelo-*PRBS*, cujos parâmetros estão indicados na Tabela 4 e na Equação 24, empregando a função *fmincon* no *System Identification Toolbox* do *MATLAB*®. A constante de tempo encontrada foi $\tau = 1083{,}0$. Os gráficos de resposta são retratados na Figura 35 e na Figura 36. A curva azul se refere ao modelo-*PRBS*; e a curva preta, à incubadora.

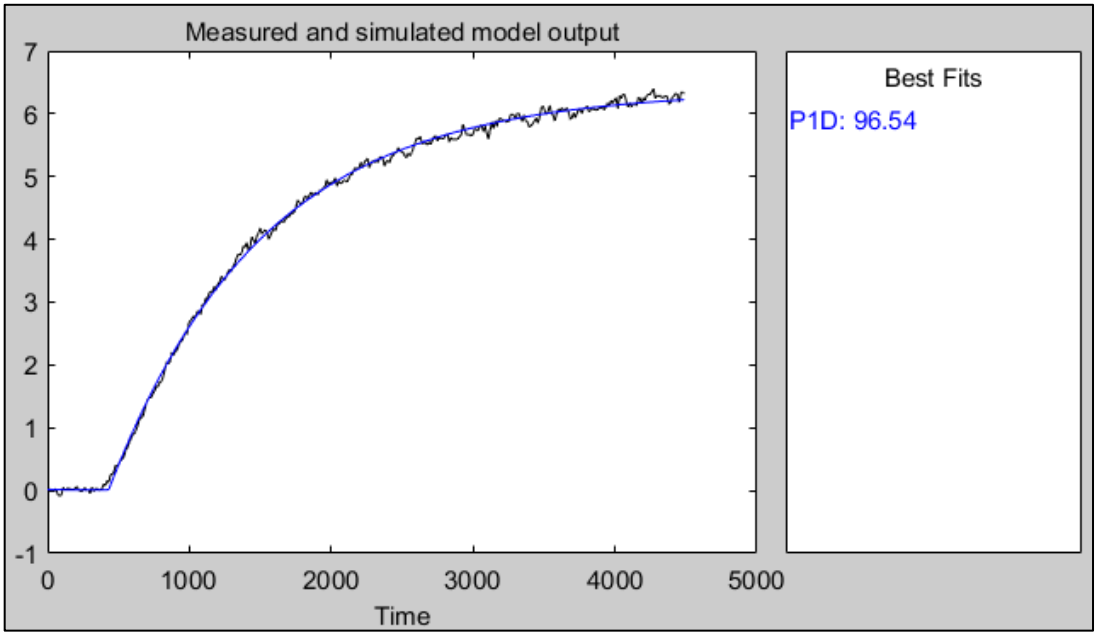
Tabela 4 – Parâmetros do modelo-*PRBS*

Ganho estático (K)	**Constante de tempo (τ)**	**Atraso (θ)**
0,159	1083,0 segundos	130 segundos

Fonte: próprio autor

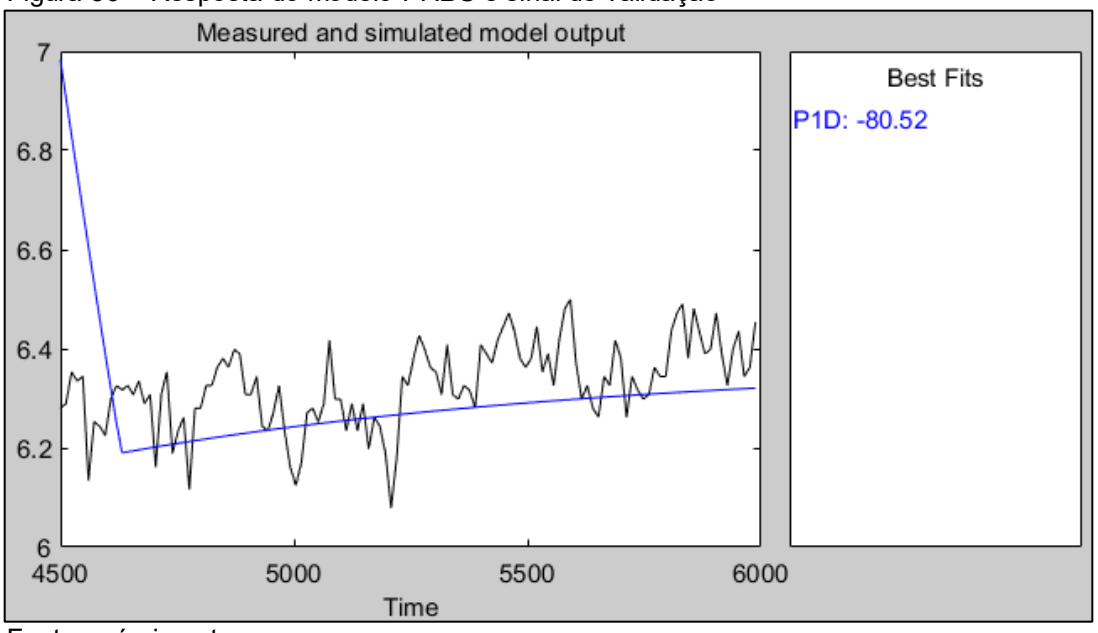
$$G_{PRBS}(s) = \frac{0{,}159}{1083s + 1} e^{-130s} \tag{24}$$

Figura 35 – Resposta do modelo-*PRBS* e sinal de construção



Fonte: próprio autor

Figura 36 – Resposta do modelo-*PRBS* e sinal de validação



Fonte: próprio autor

É importante esclarecer que a reta azul descendente, no início do gráfico da Figura 36, se deu em razão de configurações das condições iniciais da simulação, portanto não fazendo efetivamente parte da saída do modelo-*PRBS*. De fato, esse

modelo leva um tempo igual a 130 segundos (o atraso de transporte θ) para produzir a saída, que começa a ser gerada somente no tempo $t = 4630\ s$.

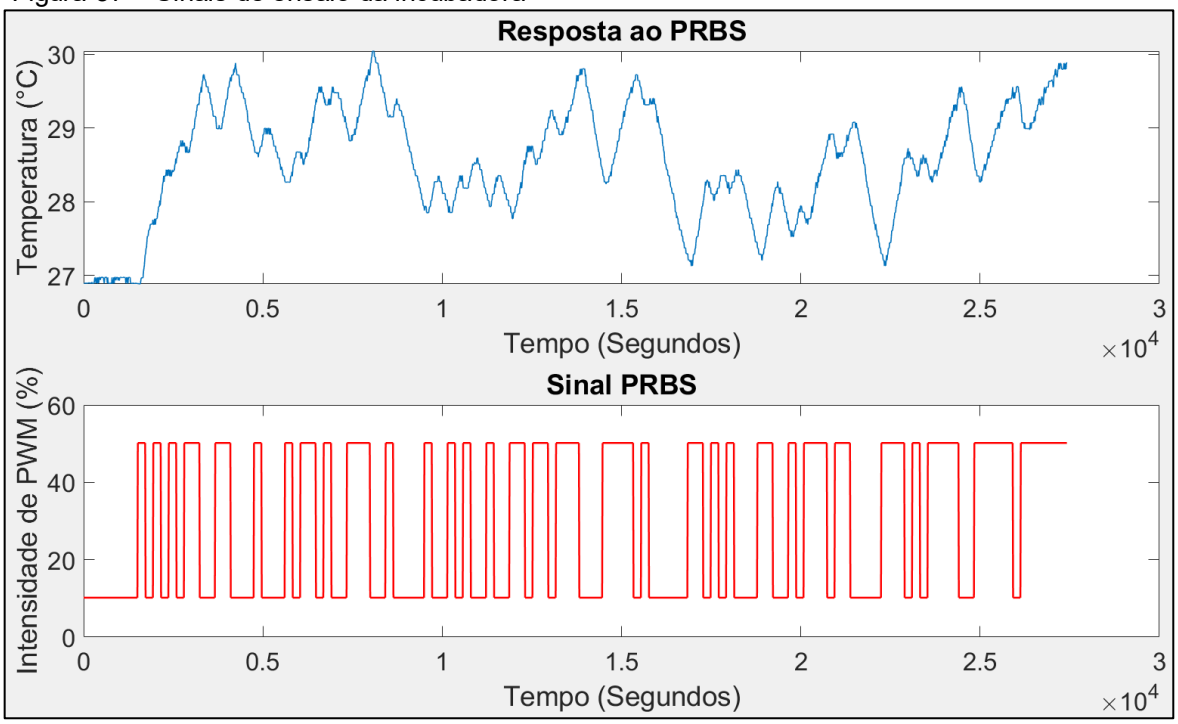
Explicada essa questão, observa-se que a curva do modelo-*PRBS* se ajusta ao sinal medido, sendo considerado um modelo válido para o ponto de operação considerado.

Encontrada a constante de tempo τ da incubadora, o sinal *PRBS* (Figura 37) pôde ser projetado conforme a Equação 6 e a Equação 22; e o ensaio da resposta da incubadora, executado.

O sinal *PRBS* aplicado ao sistema teve um nível superior de 50% de *PWM* e um inferior de 10% de *PWM*, tendo uma amplitude de 40 unidades de medida (Figura 37). Essa informação é levada em conta, no momento da estimação do modelo *FOPDT*, simplesmente dividindo o ganho estático K do modelo estimado com a rede por um fator igual a 40.

Ao todo, o procedimento levou cerca de 8,5 horas, resultando nos sinais mostrados na Figura 37.

Figura 37 – Sinais de ensaio da incubadora

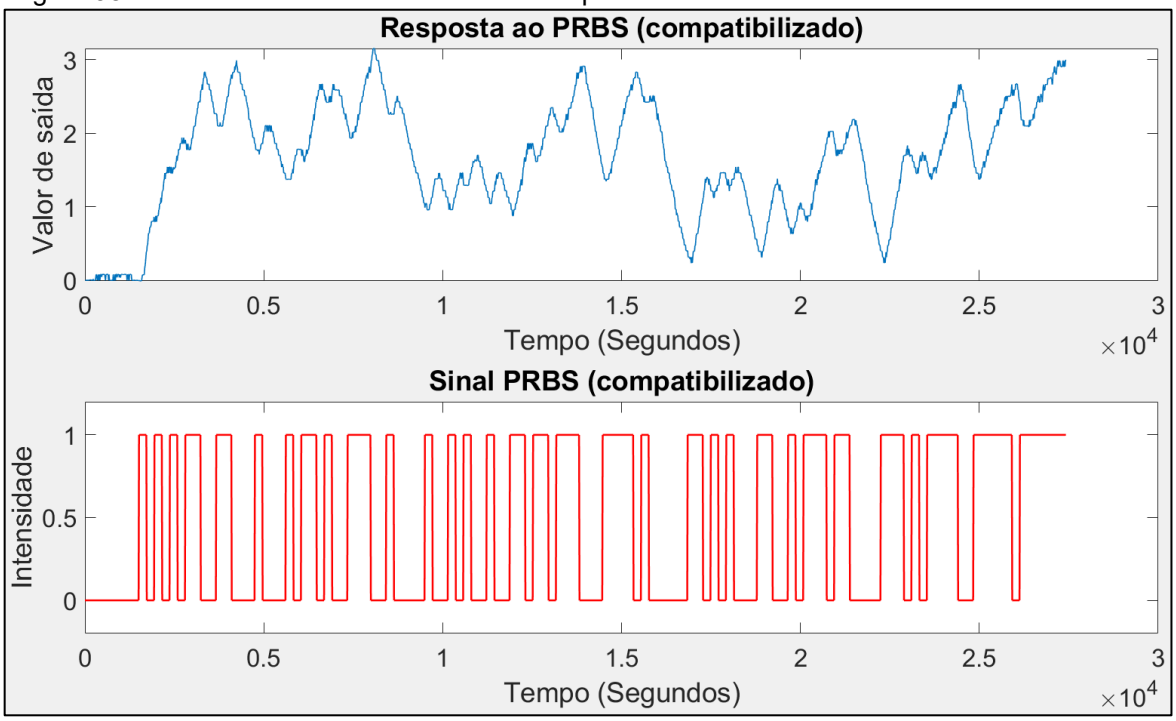


Fonte: Próprio autor

Ressalta-se que o ponto de operação da incubadora em $t = 0$ se encontrou em 26,9 °C, além do que, conforme mencionado, o sinal *PRBS* teve amplitude não unitária. Isso desperta a necessidade de adequar esses sinais ao resto do *dataset*. Dessa forma, deve ser subtraído o valor do ponto de operação inicial de ambos os sinais, tal qual sugere a Equação 25. A finalidade é deslocar o sinal para a origem do gráfico, porquanto todas as amostras do *dataset* apresentam ponto de operação nulo. Tais ajustes resultaram nos sinais da Figura 38.

$$x_a = x - x_0 \tag{25}$$

Figura 38 – Sinais do ensaio da incubadora compatibilizados



Fonte: Próprio autor

Devido à amplitude de 40 unidades do sinal *PRBS* aplicado no ensaio, após a *1D-CNN* ter estimado os parâmetros (Seção 6.4), o valor de K precisou ser dividido pelo valor dessa amplitude.

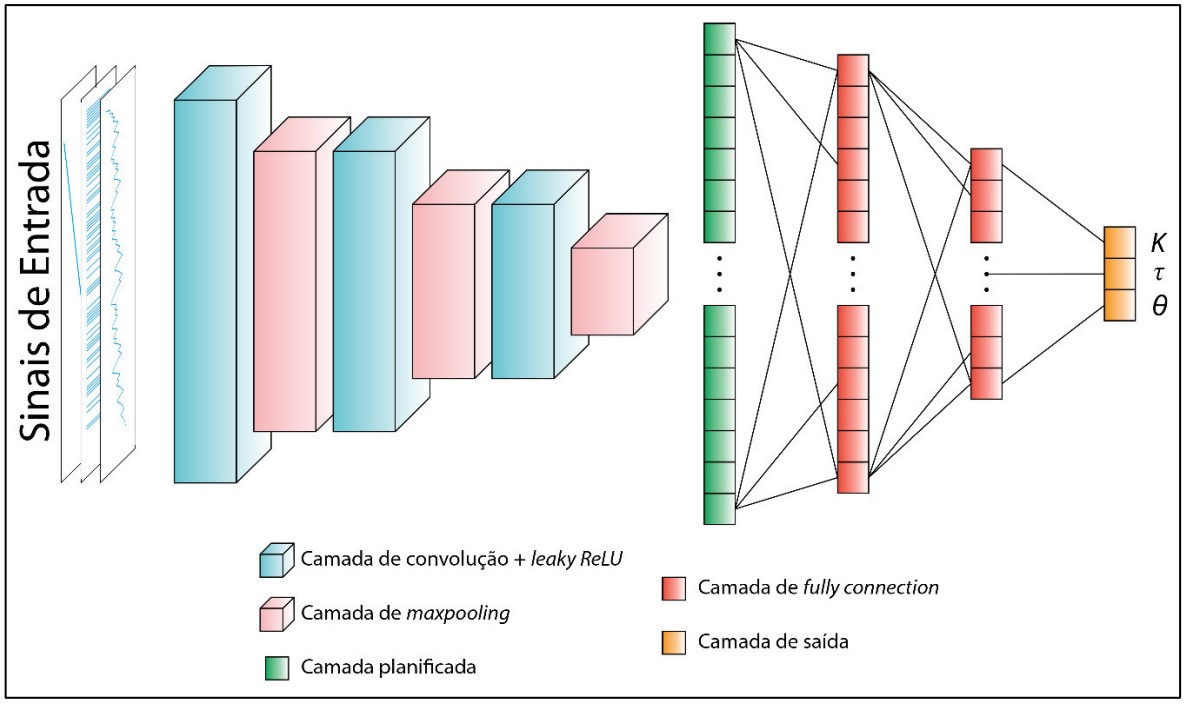
Após o armazenamento das amostras do ensaio e a sua incorporação ao *dataset*, a próxima etapa foi a de preparação prévia ao treinamento, ou seja, o *dataset* precisou ser manipulado, submetendo-se ao já citado pré-processamento

(Subseção 5.3.1). Com os dados pré-processados, encaminha-se para a etapa mais relevante da experimentação, a de treinamento.

6.3 Treinamento

Em um momento inicial, a arquitetura e os hiperparâmetros da *1D-CNN* foram estabelecidos. O funcionamento da rede neural deste trabalho, uma *1D-CNN*, é resumido na Figura 39.

Figura 39 – Arquitetura da *1D-CNN*



Fonte: Próprio autor

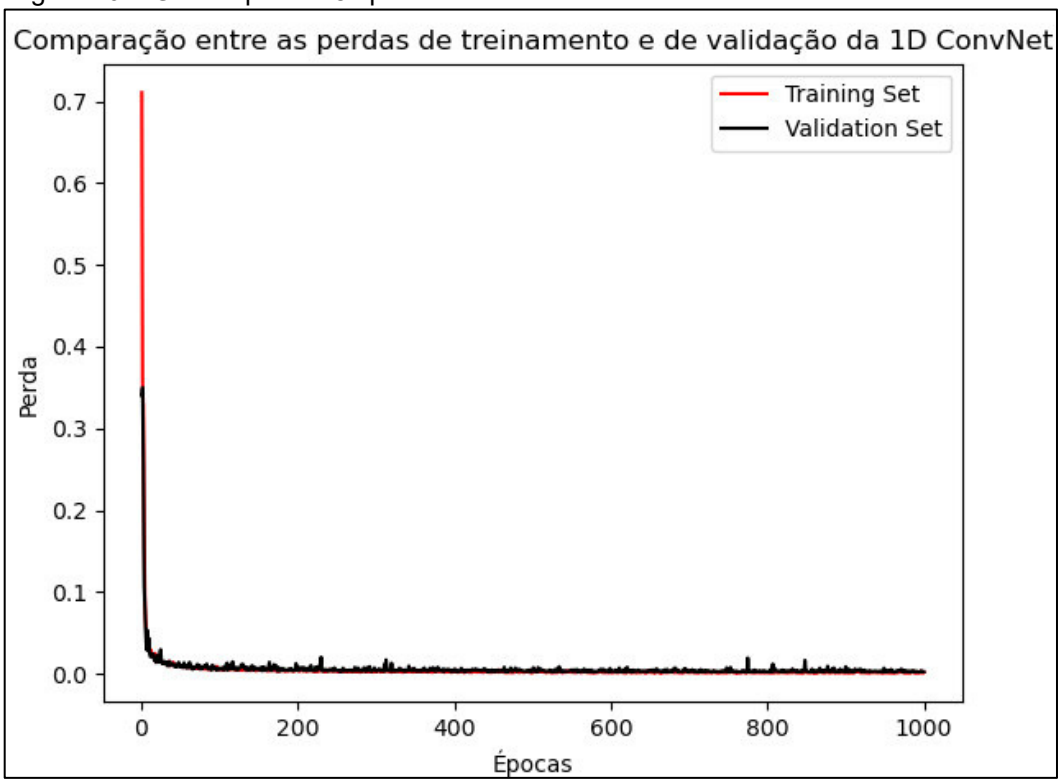
A arquitetura também é especificada no Apêndice D e no Apêndice E. A essa rede foram atribuídas 3 camadas de convolução; 3 camadas de *maxpooling*; 1 camada de *flattening*; 2 camadas de *full connection*, havendo 400 neurônios na primeira e 200 na segunda; e uma camada de saída com 3 neurônios, cada um correspondendo a um dos parâmetros de um modelo *FOPDT*, como sugere a Figura 39. Ademais, cada camada de convolução possuiu 64 filtros, cada um com comprimento de 100 unidades, e deslocamento (*stride*) de 2 unidades. As camadas de *maxpooling* também tiveram deslocamento e comprimento, ambos, de 2 unidades. Conforme pontuado na Seção 4.1 e na Subseção 4.4.2, a rede apresentou a ativação identidade (Equação 11 e Figura 13) na camada de saída e a ativação

Leaky ReLU (Equação 13 e Figura 15) nas demais camadas, adotando um coeficiente angular de $\alpha = 10^{-6}$. Já o otimizador selecionado, mencionado na Subseção 4.4.1, foi o *Adam*, com taxa de aprendizagem $lr = 10^{-3}$, e a função de perda empregada foi o *MSE*. Com isso, a rede convolucional resultante apresentou, ao todo, 10 camadas de profundidade e mais de 1,2 milhão de pesos sinápticos.

Antes de prosseguir com o treinamento, foram previamente definidos o número de épocas $epochs = 1000$. De acordo com o especificado na Subseção 5.3.2 e no Quadro 1, o treinamento foi realizado com uma *GPU* NVIDIA® GeForce 940m. Ela proporcionou um aumento de velocidade de processamento em aproximadamente 5 (cinco) vezes em relação à velocidade de processamento na *CPU*. Mesmo nessa circunstância, a *1D-CNN* levou cerca de 10,28 horas para completar seu treinamento final (Apêndice E), tendo sido ainda realizadas várias simulações ao longo de sua concepção, em algumas das quais chegando a durar 1,49 dia. Isso foi decorrência principalmente do uso de um *hardware* desatualizado de *GPU*, datado de março de 2015 (NVIDIA, 2020).

O progresso do treinamento e a validação podem ser verificados com os gráficos da Figura 40 até a Figura 43 e com o conteúdo do console *IPython*, no Apêndice E. As figuras revelam que não ocorreu *overfitting*, isto é, o desempenho da *1D-CNN* com amostras de treinamento é próximo ao desempenho com outras amostras que não foram por ela processadas. O esboço dos gráficos citados, assim como outras funcionalidades do procedimento, foi realizado mediante algumas rotinas escritas em *Python* propostas por (CARREMANS, 2018) e previamente adaptadas.

Figura 40 – Gráfico perda *vs* épocas

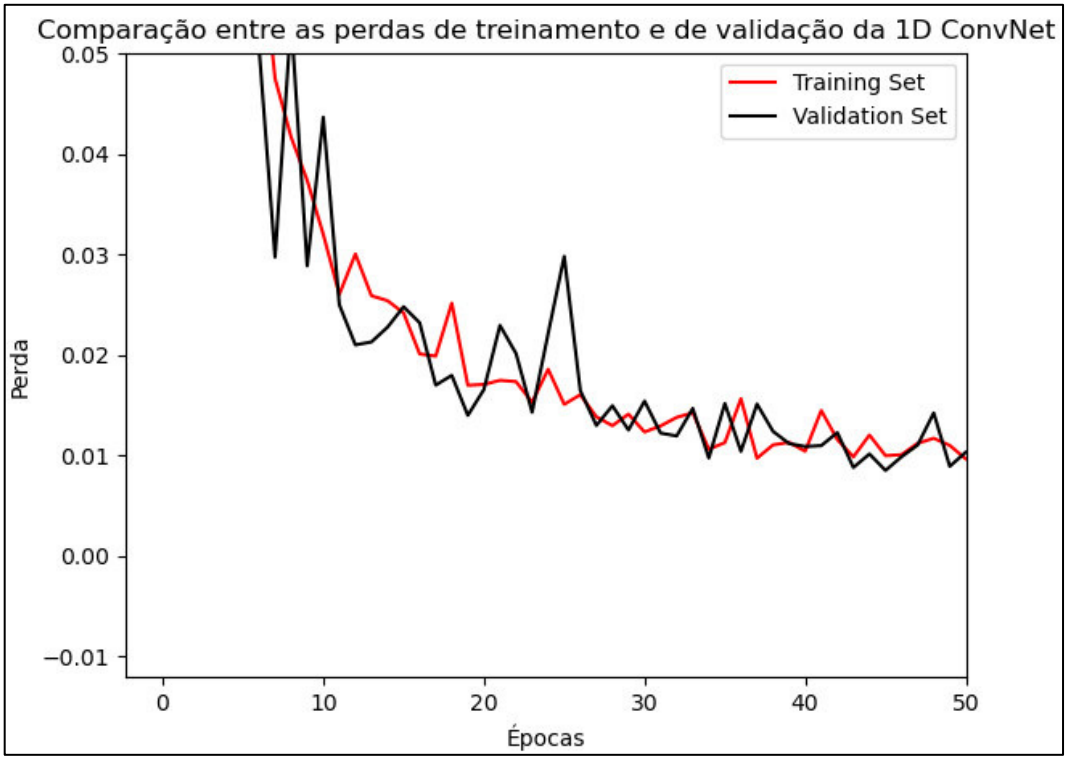


Fonte: próprio autor

Figura 41 – Gráfico perda *vs* épocas (início do treinamento)

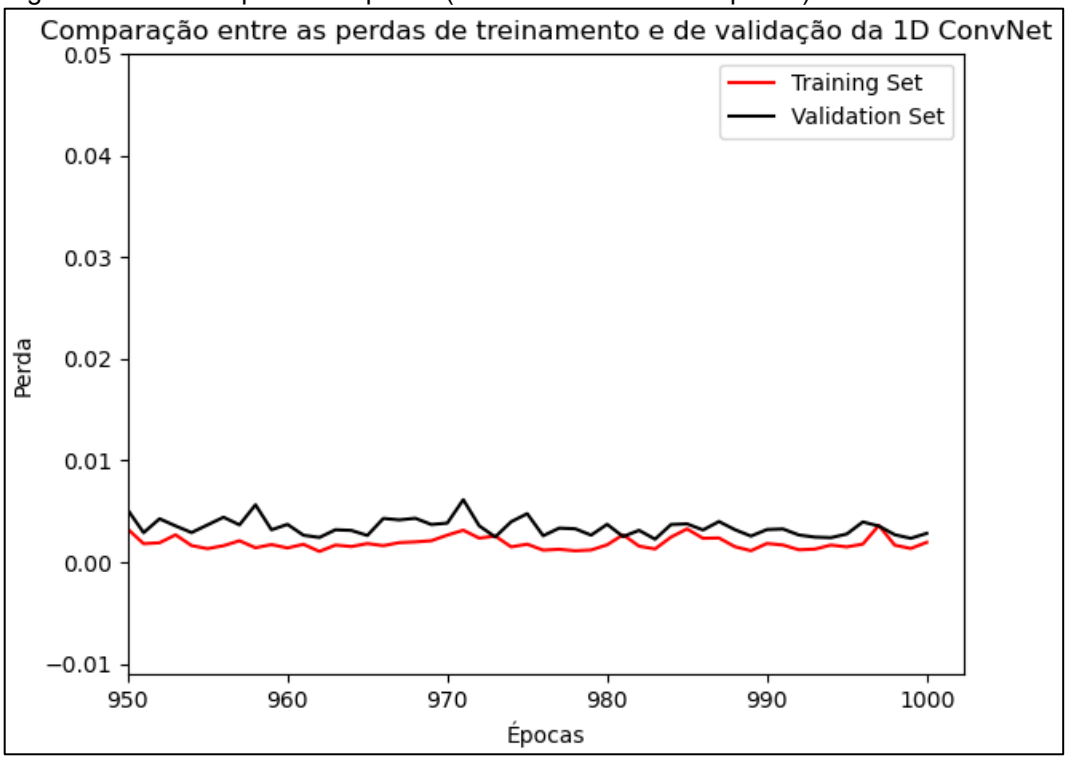
Fonte: próprio autor

Figura 42 – Gráfico perda *vs* épocas (início do treinamento ampliado)



Fonte: próprio autor

Figura 43 – Gráfico perda *vs* épocas (final do treinamento ampliado)



Fonte: próprio autor

Primeiramente, vale recordar que a medida de perda no eixo Y tem origem em números padronizados (com *feature scaling*) (Subseção 5.3.1), se tratando de valores meramente proporcionais ao *MSE*. A despeito disso, os gráficos fornecem uma boa ideia do que acontece durante o aprendizado do modelo neural. Esses valores de *MSE* entre parâmetro estimado e rótulo (parâmetro real), enquanto efetivam um papel importante na execução do *backpropagation* (Subseção 4.4.1), não realizam um papel tão relevante na avaliação do desempenho do modelo estimado, algo que é esclarecido na Seção 6.4. Em segundo lugar, observa-se pela comparação entre as perdas de treinamento (curva vermelha) e de validação (curva preta) que não houve *overfitting* durante o treinamento (Figura 26). Um dos principais motivos da ausência desse fenômeno foi o número considerável de amostras no *training set*, o que reforça a máxima de que quanto mais amostras forem empregadas na aprendizagem de um modelo de *deep learning*, melhor tenderá a ser o resultado por ele produzido. E em terceiro lugar, as curvas de perda da Figura 40 são notoriamente diferente das curvas da Figura 26. Esse fato decorre da acentuada estocasticidade existente no treinamento de redes mais complexas e da grande irregularidade de suas superfícies de perda. A consequência é o surgimento de oscilações dos valores ao longo das épocas, fazendo com que as curvas de perda não sejam estritamente decrescentes.

Ademais, os *callbacks ModelCheckpoint* e *EarlyStopping* proporcionaram grande auxílio durante o procedimento. O *ModelCheckpoint* permite salvar, ao longo de todas as épocas, os pesos sinápticos que resultaram no menor valor de perda, dentro do *validation set*. Isto é, a rede treinada não é obtida necessariamente na época nº 1000, e sim na época de mínima perda de validação; no caso deste estudo, como indica o Apêndice E, a rede atingiu um estado ótimo na época nº 947. Por sua vez, o *EarlyStopping* é capaz de interromper o treinamento quando identifica eventuais indícios de *overfitting*, como o da Figura 26. Para evitar uma interrupção equivocada em razão das oscilações de perda já mencionadas, esse *callback* possui um argumento chamado *patience*, para tolerar um determinado número de épocas nas quais não há evolução na aprendizagem. Durante o treinamento, o *EarlyStopping* não foi ativado, sinalizando novamente a ausência de *overfitting*.

Superado o treinamento, os resultados passam a ser estruturados e explorados.

6.4 Obtenção e Avaliação do Modelo

Após o treinamento, a validação e a estimação do *dataset* (sinais da incubadora e dos sistemas simulados), o desempenho referente a essas estimações foi avaliado.

A validação da *1D-CNN* foi comprovada com as curvas de perda observadas desde a Figura 40 até Figura 43, na última seção (Seção 6.3), e reforçada com a análise das estimações e dos erros que constam no Apêndice A. Observa-se que os *MSEs* do *validation set* são, em média, da mesma ordem que aqueles do *training set*, o que valida o aprendizado da rede convolucional.

Consoante a explicação da Subseção 4.4.1, o desempenho de RNAs, em tarefas de regressão, é mensurado pelo valor de perda (*loss*), calculados a partir da saída da rede. No entanto, pelo fato de esta aplicação ser identificação de sistemas, o que se avalia é a semelhança entre os sinais de resposta real e estimado, e não propriamente a aproximação entre os parâmetros reais (rótulo) $y = [K, \tau, \theta]$ e as saídas estimadas $\hat{y} = [\widehat{K}, \hat{\tau}, \hat{\theta}]$. Diante disso, a performance da *1D-CNN* foi realizada cumulativamente de 3 (três) formas:

a) Pela comparação gráfica entre a curva gerada pela incubadora e a curva gerada pelo modelo-*CNN*.

b) Pelo erro quadrático médio (*MSE*) dos pontos da curva estimada em relação à real; e

c) Pelo desvio padrão (σ) dos pontos da curva estimada em relação à real.

Em relação a estes dois últimos cálculos, eles foram realizados tanto para o modelo-*PRBS* (discutido na Seção 3.4 e na Seção 6.2) quanto para o modelo-*CNN* e constam na Tabela 5, exibida a seguir.

Tabela 5 – Estimações do modelo dinâmico da incubadora neonatal

Amostras	Parâmetros estimados (predições)			Erro Quadrático Médio (*MSE*) entre as Curvas de Resposta	Desvio Padrão (σ) do Erro entre as Curvas de Resposta
	Ganho estático K (adim.)	Constante de tempo τ (s)	Atraso θ (s)		
Modelo-*PRBS*	0,159	1083,0	130	2,0916	0,7272
Modelo-*CNN*	0,100	901,75	124	0,2463	0,2565

Fonte: próprio autor

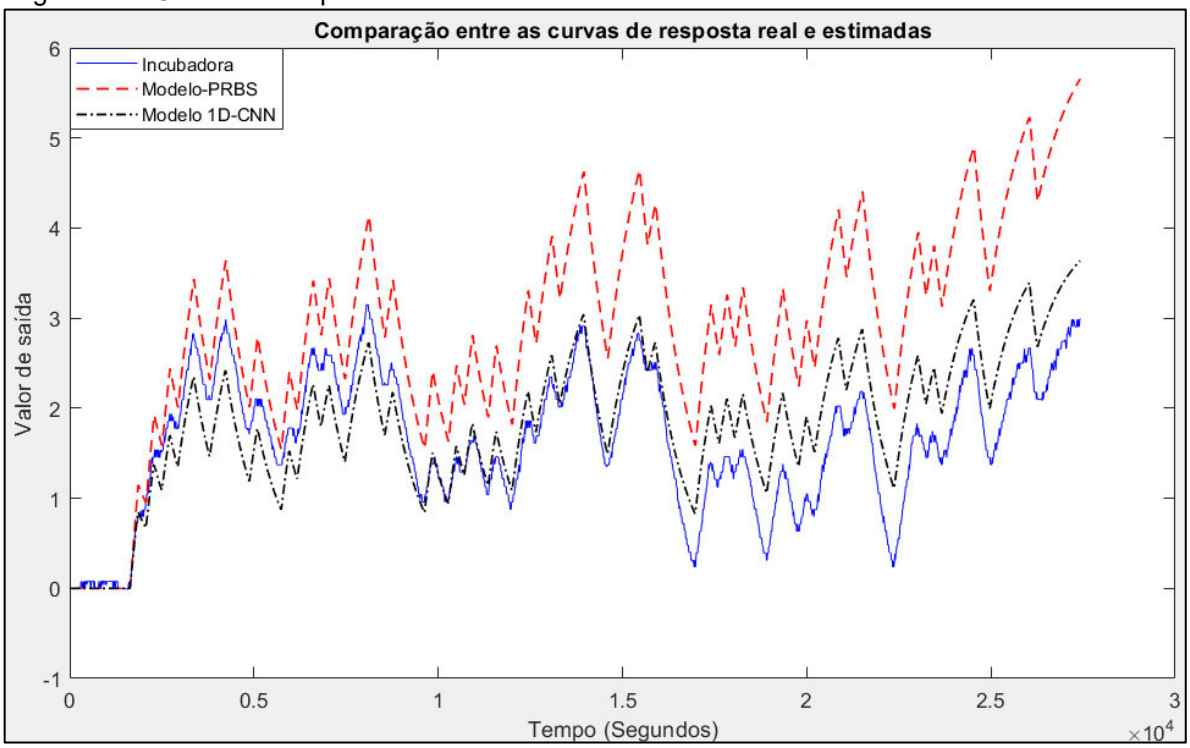
Portanto, o modelo dinâmico indicado pela *1D-CNN* tem a seguinte forma:

$$G_{CNN}(s) = \frac{0{,}100}{901{,}75s + 1} e^{-124s} \tag{26}$$

Ressalta-se que a predição original do ganho estático era $K = 4{,}004$, mas o valor deve ser dividido por 40, tendo em vista a amplitude do sinal *PRBS* aplicado na incubadora neonatal, ponto discutido na Seção 6.2. Essa adaptação resultou no valor $K = 0{,}100$.

Já no que concerne à comparação de gráficos, a Figura 44 ilustra o comportamento dinâmico do modelo-*CNN* (curva preta tracejada e pontilhada) e do sistema físico (curva azul contínua). Também é retratada, na mesma figura, a resposta do modelo-*PRBS* (curva vermelha tracejada).

Figura 44 – Curvas de resposta estimadas e real



Fonte: próprio autor

Tais resultados contrastam com os gráficos da Figura 35 e da Figura 36, que mostram um melhor ajuste do modelo-*PRBS* à resposta ao degrau medida na incubadora. Diante disso, conclui-se que o comportamento desse sistema físico não é igual para todos os pontos de operação. A escolha desses pontos implica diretamente no desempenho dos modelos estimados, tendo em vista a faixa de operação na Figura 33, que se distingue da faixa de operação da Figura 37, a qual inicia em $T_0 = 26{,}89\ °C$ e atinge o limite superior de $T_0 = 30{,}05\ °C$. Isso significa que a incubadora apresentou um comportamento significativamente não linear na faixa de operação considerada.

Esse grau de não linearidade também esclarece o desempenho do modelo-*CNN*; ao estimá-lo, a *1D-CNN* se baseou em sinais produzidos por sistemas lineares invariantes no tempo, criados por simulação no *software MATLAB*®. Em razão disso, a rede convolucional sofre certa penalidade em termos de performance, se for utilizada para identificar sistemas físicos predominantemente não lineares. Isso pode ser confirmado comparando os resultados da estimação da incubadora com as dos sistemas simulados, que figuram no Apêndice A.

Como consequência, o modelo-*CNN* pode ser utilizado no projeto de controladores, somente se tal projeto admitir as especificações de erro da Tabela 5. *A contrario sensu*, se o controlador necessitar de um erro menor que $MSE = 0{,}2463$ ou de um desvio padrão menor que $\sigma = 0{,}2565$, então o modelo-*CNN* não atenderá aos requisitos desse projeto.

Contudo, um ponto altamente relevante é o seguinte: desde que foi treinada e validada, a rede convolucional está apta a identificar vários outros sistemas físicos *FOPDT* eminentemente lineares que respeitem os limites da Tabela 3, já que a rede foi treinada com modelos dinâmicos com essas características.

Tendo sido discutidos os resultados, o próximo capítulo conclui o trabalho.

7 CONCLUSÃO

Esta monografia consiste em obter um modelo dinâmico de uma incubadora neonatal, tendo como ferramenta uma rede neural convolucional unidimensional (*1D-CNN*). O propósito foi achar uma descrição do comportamento dinâmico da temperatura no interior do dispositivo, estimando os parâmetros de sua função de transferência, o que caracterizou uma tarefa de regressão não linear de múltiplas saídas, ou seja, um mapeamento entre os sinais da incubadora e os seus parâmetros. Conforme exposto na Seção 4.1, isso difere de muitas aplicações envolvendo redes convolucionais, porque estas são mais frequentemente empregadas em tarefas de classificação de imagens. A *1D-CNN* efetuou a estimação a partir dos sinais do sistema físico, após ter sido treinada e validada com as amostras, respectivamente, do *training set* e do *validation set*, pondo em prática os conceitos de convolução, *backpropagation*, *overfitting*, otimizadores de descida de gradiente, entre outros.

Os resultados apontam que a dinâmica da incubadora na faixa de operação da Figura 37 foi predominantemente não linear, pelas razões discutidas na Seção 6.4, o que resultou nos erros da Tabela 5. Essa rede consegue identificar outros sistemas *FOPDT* predominantemente lineares, em virtude do treinamento realizado com as amostras geradas de modelos lineares simulados no *MATLAB*®.

Para obter um modelo de performance aprimorada, algumas alternativas podem ser vislumbradas, dentre elas:

a) Compor o *dataset* somente com sinais oriundos do sistema físico, algo inviável no caso da incubadora, por conta da duração de ensaio excessivamente longa e da necessidade de executar milhares de ensaios. Nesse caso, a *1D-CNN* perderia a capacidade de identificar outros sistemas dinâmicos físicos; ou

b) Substituir os sinais *PRBS* por sinais degrau.

Também é oportuno frisar a necessidade do cumprimento de certos requisitos, que permitiram à *1D-CNN* a habilidade de estimar modelos dinâmicos; dentre eles, estão o conhecimento prévio do tipo de função de transferência (*FOPDT*), a seleção do tipo de sinal de entrada (*PRBS*) e o pré-processamento do *dataset*. Além disso, como explicado na Seção 6.1, a profundidade da arquitetura e o

tamanho do *dataset* não puderam ser aumentados devido à capacidade limitada do computador utilizado nos experimentos, diante do esforço computacional da tarefa; mesmo utilizando uma placa gráfica para acelerar a programação no *software Spyder*®, foram despendidas mais de 10 horas para concluir o treinamento. Do contrário, arquiteturas mais robustas seriam eventualmente construídas; e melhores resultados, alcançados.

Não obstante, se os índices atingidos pela *1D-CNN*, que constam na Tabela 5, forem suficientes para o projeto de determinado controlador, então seu modelo dinâmico poderá ser usado, permitindo, dessa forma, o controle de um ambiente termoneutro para bebês prematuros.

Isso revela que essa rede convolucional unidimensional foi uma ferramenta alternativa para a identificação de sistemas, uma tarefa que implica a manipulação de dados sequenciais, o que reforça as conclusões de (BAI; KOLTER; KOLTUN, 2018). Apesar disso, é relevante investigar ainda mais esse modelo neural, a fim de aprimorá-lo e, assim, produzir modelos dinâmicos mais eficazes. A partir desse contexto, são apontadas algumas ideias na seção abaixo.

7.1 Trabalhos Futuros

Tendo em vista a oportunidade de continuar as investigações e de aprimorar as técnicas e os resultados aqui apresentados, são sugeridos os seguintes pontos como objetos de trabalhos futuros:

- Empregar outros tipos de sinais de entrada (por exemplo, sinais degrau);
- Projetar um sistema de controle de temperatura da incubadora neonatal com base no modelo-*CNN*;
- Replicar os experimentos em um modelo comercial de incubadora neonatal;
- Adaptar a *1D-CNN* para identificar sistemas físicos sem ser previamente conhecida a estrutura de suas funções de transferência (1ª ordem com atraso, 2ª ordem sem atraso, etc.);
- Testar outras arquiteturas de rede ou outros tipos de modelos de *deep learning*; e
- Selecionar técnicas tradicionais de identificação para fins de comparação de desempenho.

REFERÊNCIAS

AGOSTINI, N. **Sistema computadorizado para verificação da funcionalidade em incubadoras neonatais**. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Santa Catarina. Florianópolis, p. 79. 2003. il. color.

AGUIRRE, L. A. **Introdução à Identificação de Sistemas. Técnicas Lineares e não Lineares Aplicadas a Sistemas**. 3. ed. Belo Horizonte: UFMG, 2007.

AGUIRRE, L. A. Modelagem e Identificação - Introdução. **Youtube**, 22 ago. 2019a. Disponivel em: <https://www.youtube.com/watch?v=TWdgSG0sMIQ&list=PLALrL4i0Pz6DrrCkkJ-k-_S3qi1bFzUUu&index=1>. Acesso em: 16 abr. 2020. 8:56.

AGUIRRE, L. A. Modelagem e Identificação - Validação: Escolha de Dados. **Youtube**, 7 nov. 2019b. Disponivel em: <https://www.youtube.com/watch?v=AJeMx4q3VAo&list=PLALrL4i0Pz6DrrCkkJ-k-_S3qi1bFzUUu&index=56>. Acesso em: 27 fev. 2021. 12:04.

ALBUQUERQUE, A. A. M. D. **Sistema de Controle de uma Incubadora Neonatal segundo a Norma NBR IEC 60.601-2/19: Aspectos de Avaliação, Identificação Dinâmica e Novas Propostas**. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Ceará. Fortaleza, p. 87. 2012.

ALOM, M. Z. et al. A State-of-the-Art Survey on Deep Learning Theory and Architectures. **Multidisciplinary Digital Publishing Institute (MDPI) Electronics**, v. 8, n. 292, 5 mar. 2019. Disponivel em: <https://www.mdpi.com/2079-9292/8/3/292>. Acesso em: 20 fev. 2021. il. color.

AMORIM, P. H. J.; ARAÚJO, F. R. D. S. GPU: A matriz de processadores, 25 nov. 2011. Disponivel em: <https://ic.unicamp.br/~ducatte/mo401/1s2011/T2/Artigos/G02-121498-095431-t2.pdf>. Acesso em: 18 set. 2020.

APICELLA, A. et al. A Survey on Modern Trainable Activation Functions, 25 fev. 2021. Disponivel em: <https://arxiv.org/abs/2005.00817>. Acesso em: 04 mar. 2021.

ASUS. ASUS X555 Tech Specs. **ASUS**, 2020. Disponivel em: <https://www.asus.com/Laptops/For-Home/Everyday-use/ASUS-X555/techspec/>. Acesso em: 14 out. 2020.

BAI, S.; KOLTER, J. Z.; KOLTUN, V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks, 19 abr. 2018. Disponivel em: <https://arxiv.org/abs/1803.01271>. Acesso em: 07 mar. 2020.

BEZERRA CORREIA, W.; CLAURE TORRICO, B.; OLÍMPIO PEREIRA, R. D. Optimal control of MIMO dead-time linear systems with dead-time compensation

structure, DYNA v. 84, n. 200, mar. 2017. p. 62-71. Disponivel em: <http://www.repositorio.ufc.br/handle/riufc/40385>. Acesso em: 20 fev. 2021.

BEZERRA, S. G. T. D. A. **Reservoir Computing com Hierarquia para Previsão de Vazões Médias Diárias**. Dissertação (Mestrado em Engenharia da Computação) - Universidade de Pernambuco. Recife, p. 70. 2016. il. color.

BORJAS, S. D. M.; GARCIA, C. Identificação Determinística por Subespaços. **Trends in Computational and Applied Mathematics**, v. 13, n. 3, 02 set. 2012. p. 207-218. Disponivel em: <https://tema.sbmac.org.br/tema/article/view/535>. Acesso em: 15 fev. 2021. il. p&b.

BUDUMA, N.; LACASCIO, N. **Fundamentals of Deep Learning**. 1. ed. [S.l.]: O'Reilly, 2017.

CAFFE. Caffe. **Caffe**, 2020. Disponivel em: <https://caffe.berkeleyvision.org/>. Acesso em: 28 jun. 2020.

CARREMANS, B. Handling overfitting in deep learning models. **Towards Data Science**, 23 ago. 2018. Disponivel em: <https://towardsdatascience.com/handling-overfitting-in-deep-learning-models-c760ee047c6e>. Acesso em: 01 jul. 2020.

CAULFIELD, B. What's the Difference Between a CPU and a GPU? **NVIDIA**, 16 dez. 2009. Disponivel em: <https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/>. Acesso em: 18 set. 2020.

CHOLLET, F. **Deep Learning with Python**. 1. ed. Shelter Island: Manning, 2018.

COELHO, A. A. R.; COELHO, L. D. S. **Identificação de Sistemas Dinâmicos Lineares**. Florianópolis: UFSC, 2004.

FACELI, K. et al. **Inteligência Artificial. Uma Abordagem de Aprendizado de Máquina**. Rio de Janeiro: LTC, 2011.

FILHO, J. R. V.; PÁSSARI, I. A.; NIVEIROS, S. I. Gestão de Custos Hospitalares: um Estudo de Caso no Hospital Santa Casa de Misericórdia e Maternidade de Rondonópolis – MT. **XXIV Congresso Brasileiro de Custos**, Florianópolis, n. 27, 15-17 nov. 2017. Disponivel em: <https://anaiscbc.emnuvens.com.br/anais/article/view/4362>. Acesso em: 16 fev. 2021.

GÉRON, A. **Hands-on Machine Learning with Scikit-Learn & TensorFlow**. 1. ed. Sebastopol: O'Reilly, 2017. Acesso em: 30 jun. 2020.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. Disponivel em: <https://www.deeplearningbook.org>. Acesso em: 05 jun. 2020.

GPAR. Controle de Incubadora Neonatal. **Grupo de Pesquisa em Automação, Controle e Robótica**, 2020. Disponivel em: <https://gpar.ufc.br/?page_id=981>. Acesso em: 05 jun. 2020.

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

IFF/FIOCRUZ. Principais questões sobre Prevenção de Hipotermia: da sala de parto à admissão na UTI neonatal. **Portal de Boas Práticas em Saúde da Mulher, da Criança e do Adolescente**, 01 fev. 2019. Disponivel em: <https://portaldeboaspraticas.iff.fiocruz.br/atencao-recem-nascido/principais-questoes-sobre-prevencao-de-hipotermia-da-sala-de-parto-a-admissao-na-uti-neonatal/>. Acesso em: 13 set. 2020.

INCE, T. et al. Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks. **IEEE Transactions on Industrial Electronics**, v. 63, n. 11, 28 jun. 2016. p. 7067-7075. Disponivel em: <https://ieeexplore.ieee.org/document/7501527>. Acesso em: 13 mar. 2021.

KARLIK, B.; OLGAC, A. V. Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks. **International Journal of Artificial Intelligence And Expert Systems**, v. 1, n. 4, fev. 2011. Disponivel em: <https://www.researchgate.net/publication/228813985_Performance_Analysis_of_Various_Activation_Functions_in_Generalized_MLP_Architectures_of_Neural_Networks>. Acesso em: 30 mar. 2020.

KEESMAN, K. J. **System Identification. An Introduction**. Wageningen: Springer, 2011.

KERAS. Masking layer. **Keras Documentation**, 2020. Disponivel em: <https://keras.io/api/layers/core_layers/masking/>. Acesso em: 30 jun. 2020.

KINGMA, D. P.; BA, J. L. Adam: A Method for Stochastic Optimization. **3rd International Conference for Learning Representations**, San Diego, 7-9 mai. 2015. Disponivel em: <https://arxiv.org/abs/1412.6980>. Acesso em: 23 jun. 2020.

KIRANYAZ, S. et al. Convolutional Neural Networks for Patient-Specific ECG Classification. **IEEE Engineering in Medicine and Biology Society Conference**, n. 37, ago. 2015. Disponivel em: <https://www.researchgate.net/publication/285493884_Convolutional_Neural_Networks_for_Patient-Specific_ECG_Classification>. Acesso em: 13 mar. 2021.

LAI, W. H.; KEK, S. L.; GAIK, T. K. Solving Nonlinear Least Squares Problem Using Gauss-Newton Method. **IJISET - International Journal of Innovative Science, Engineering & Technology**, v. 4, n. 1, 1 jan. 2017. p. 258-262. Disponivel em: <https://www.researchgate.net/publication/317036822_Solving_Nonlinear_Least_Squares_Problem_Using_Gauss-Newton_Method>. Acesso em: 18 mar. 2021.

LECUN, Y. et al. Gradient-Based Learning Applied to Document Recognition. **Proceedings of the IEEE**, v. 86, 1998. p. 2278-2324.

LIU, D. A Practical Guide to ReLU. **Medium**, 30 nov. 2017. Disponivel em: <https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7>. Acesso em: 05 jun. 2020.

LJUNG, L. **System Identification. Theory for the User**. 2. ed. Linköping: Prentice-Hall, 1999.

LU, L. et al. Dying ReLU and Initialization: Theory and Numerical Examples, 15 mar. 2019. Disponivel em: <https://arxiv.org/abs/1903.06733>. Acesso em: 25 jun. 2020.

MACHADO, V. C. **Planejamento de Perturbações para aldentificação de Modelos Dinâmicos**. Dissertação (Mestrado em Engenharia Química) - Universidade Federal do Rio Grande do Sul. Porto Alegre, p. 110. 2004.

MAHESWARI, J. P. Breaking the curse of small datasets in Machine Learning: Part 1. **Towards Data Science**, 21 dez. 2018. Disponivel em: <https://towardsdatascience.com/breaking-the-curse-of-small-datasets-in-machine-learning-part-1-36f28b0c044d>. Acesso em: 29 jun 2020.

MATHWORKS. MATLAB Documentation: Solver-Based Nonlinear Optimization. **Mathworks**, 2021. Disponivel em: <https://www.mathworks.com/help/optim/ug/fmincon.html>. Acesso em: 04 mar. 2021.

MATLPLOTLIB. User's guide: History. **Matlplotlib**, 2020. Disponivel em: <https://matplotlib.org/stable/users/history.html>. Acesso em: 28 jun. 2020.

MINISTÉRIO DA SAÚDE. Juntos para os bebês nascidos muito cedo, cuidando do futuro: 17/11 – Dia Mundial da Prematuridade. **Biblioteca Virtual em Saúde**, 16 nov. 2020. Disponivel em: <http://bvsms.saude.gov.br/ultimas-noticias/3358-juntos-para-os-bebes-nascidos-muito-cedo-cuidando-do-futuro-17-11-dia-mundial-da-prematuridade>. Acesso em: 16 fev. 2021.

NABI, J. Machine Learning - Fundamentals: Basic theory underlying the field of Machine Learning. **Towards Data Science**, 15 ago. 2018. Disponivel em: <https://towardsdatascience.com/machine-learning-basics-part-1-a36d38c7916>. Acesso em: 25 jun. 2020. il. p&b.

NAKAMURA, W. T. **Utilização das Redes Neurais Artificiais no Planejamento Estratégico de Negócios: uma Análise do Software Zaitun Time Series**. Dissertação (Mestrado em Engenharia de Software). Belo Horizonte, p. 66. 2013. il. p&b.

NIELSEN, M. Chapter 1: Using neural nets to recognize hardwritten digits. **Neural Networks and Deep Learning**, 2015. Disponivel em:

<http://neuralnetworksanddeeplearning.com/chap1.html>. Acesso em: 31 mar. 2020. il. p&b.

NUMPY. What is NumPy? **NumPy**, 2020. Disponivel em: <https://numpy.org/doc/stable/user/whatisnumpy.html#why-is-numpy-fast>. Acesso em: 28 jun. 2020.

NVIDIA. GeForce 940M Specifications, 2020. Disponivel em: <https://www.nvidia.com/en-us/geforce/gaming-laptops/geforce-940m/specifications/>. Acesso em: 14 out. 2020.

OLÍMPIO PEREIRA, R. D. **Auto-Tuning Control Techniques Applied to a Neonatal Incubator**. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Ceará. Fortaleza, p. 116. 2016.

OLIVEIRA, M. A. D. **Sistema de Ensaio de Desempenho de Incubadora Neonatal**. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Santa Catarina. Florianópolis, p. 95. 2007.

OLIVEIRA, M. A.; TAVARES, M. C.; MORAES, R. Sistema para Ensaio de Funcionalidade de Incubadoras Neonatais. **IV Latin American Congress on Biomedical Engineering 2007**, jan. 2007. p. 528-532. Disponivel em: <https://www.researchgate.net/publication/292206596_Sistema_para_Ensaio_de_Funcionalidade_de_Incubadoras_Neonatais>. Acesso em: 04 set. 2020.

ORENSTEIN, L. P. **Procedimento para Identificação de Sistemas Dinâmicos em Ambiente Industrial**. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Rio de Janeiro. Rio de Janeiro, p. 108. 2013.

PACHECO, A. G. C. **Multiplicação de matrizes: uma comparação entre as abordagens sequencial (CPU) e paralela (GPU)**. Relatório Técnico - Universidade Federal do Espírito Santo. Vitória, p. 10. 2019.

PANDAS. Package Overview. **Pandas**, 2020. Disponivel em: <https://pandas.pydata.org/docs/getting_started/overview.html>. Acesso em: 28 jun. 2020.

PONTI, M. A.; COSTA G. B. P. D. Capítulo 3: Como funciona o Deep Learning. **Tópicos em Gerenciamento de Dados e Informações 2017**, Uberlândia, n. 32, 02 out. 2017. p. 63-93. Disponível em: <https://sbbd.org.br/2017/wp-content/uploads/sites/3/2017/10/topicos-em-gerenciamento-de-dados-e-informacoes-2017.pdf>. Acesso em: 01 abr. 2020.

REDE NACIONAL PRIMEIRA INFÂNCIA. O custo da prematuridade para a saúde pública ultrapassa R$ 8 bilhões por ano no país. **RNPI**, 21 ago. 2019. Disponivel em: <http://primeirainfancia.org.br/o-custo-da-prematuridade-para-a-saude-publica-ultrapassa-r-8-bilhoes-por-ano-no-pais/>. Acesso em: 16 fev. 2021.

RODRIGUES, G. G. **Identificação de Sistemas Dinâmicos Não-Lineares Utilizando Modelos NARMAX Polinomiais - Aplicação a Sistemas Reais**. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Minas Gerais. Belo Horizonte, p. 105. 1996.

ROSENBLATT, F. The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain. **Psychological Review**, Buffalo, 65, n. 6, 23 Abril 1958. p. 386-408. Disponivel em: <https://www.uni-bielefeld.de/lili/personen/eikmeyer/theprorez/frosenblatt.pdf>. Acesso em: 23 mar. 2020.

SAHA, S. Convolutional Neural Networks - the ELI5 way. **Towards Data Science**, 15 dez. 2018. Disponivel em: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Acesso em: 24 abr. 2020.

SASAKI, T.; PRATT, S. C. The Psychology of Superorganisms: Collective Decision Making by Insect Societies. **Annual Review of Entomology**, v. 63, jan. 2018. p. 259-275.

SCIKIT-LEARN. scikit-learn: Machine Learning in Python. **scikit-learn**, 2020. Disponivel em: <https://scikit-learn.org/stable/>. Acesso em: 28 jun. 2020.

SONI, D. Supervised vs. Unsupervised Learning. **Towards Data Science**, 22 mar. 2018. Disponivel em: <https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d>. Acesso em: 09 jun. 2020. il. color.

SUPERDATASCIENCE TEAM. The Ultimate Guide to Convolutional Neural Networks (CNN). **SuperDataScience**, 27 ago. 2018. Disponivel em: <https://www.superdatascience.com/blogs/the-ultimate-guide-to-convolutional-neural-networks-cnn>. Acesso em: 26 abr. 2020.

TENSORFLOW. Tutorial. **Tensorflow**, 2020. Disponivel em: <https://www.tensorflow.org/tutorials?hl=pt_br>. Acesso em: 18 set. 2020.

TREVISANUTO, D.; TESTONI, D.; ALMEIDA, M. F. B. D. Maintaining normothermia: Why and how? **Seminars in Fetal and Neonatal Medicine**, mar. 2018. p. 7. Disponivel em: <https://www.researchgate.net/publication/323909528_Maintaining_normothermia_Why_and_how>. Acesso em: 13 set. 2020.

VIANA, S. O pipeline de visão computacional. **Medium**, 15 fev. 2020. Disponivel em: <https://suzana-svm.medium.com/o-pipeline-de-visao-computacional-com-python-opencv-adc70112f5ee>. Acesso em: 17 mar. 2021. il. color.

WOODWORTH, K. R. et al. Birth and Infant Outcomes Following Laboratory-Confirmed SARS-CoV-2 Infection in Pregnancy — SET-NET, 16 Jurisdictions, March 29–October 14, 2020. **Morbidity and Mortality Weekly Report**, v. 44, n. 69, 6 nov. 2020. p. 1635-1640. Disponivel em:

<https://www.cdc.gov/mmwr/volumes/69/wr/mm6944e2.htm>. Acesso em: 14 mar. 2021.

WORLD HEALTH ORGANIZATION, MATERNAL AND NEWBORN HEALTH/SAFE MOTHERHOOD. **Therman Protection of the Newborn. A Practical Guide**. Genebra: World Health Organization, 1997. Disponivel em: <https://www.who.int/maternal_child_adolescent/documents/ws42097th/en/>. Acesso em: 13 set. 2020.

XU, B. et al. Empirical Evaluation of Rectified Activations in Convolution Network, 27 nov. 2015. Disponivel em: <https://arxiv.org/abs/1505.00853>. Acesso em: 04 mar. 2021.

APÊNDICE A – ESTIMAÇÕES DOS MODELOS DINÂMICOS CRIADOS NO *MATLAB*®

Aqui se encontram alguns resultados das estimações dos sistemas dinâmicos criados via simulação, isto é, do restante do *dataset* (*training set* e *validation set*). O resultado referente ao *test set* é apresentado na Tabela 5, na Seção 6.4, que é a amostra referente à incubadora. A Tabela 5 possui menos colunas que as deste apêndice porque aquela se refere a uma amostra sem rótulos.

A.1 *Training Set*:

N° da amostra	Parâmetros reais (rótulos)			Parâmetros estimados (predições)			Erro Quadrático Médio (*MSE*) entre as Curvas de Resposta	Desvio Padrão (σ) do Erro entre as Curvas de Resposta
	Ganho estático K (adim.)	Constante de tempo τ (s)	Atraso θ (s)	Ganho estático K (adim.)	Constante de tempo τ (s)	Atraso θ (s)		
1	4,57	803,00	8,70	4,52	798,74	8,38	5,958E-04	8,875E-03
2	1,40	710,60	86,2	1,39	708,78	87,2	5,400E-05	2,820E-03
3	4,83	290,20	87,4	4,80	294,59	86,6	3,209E-04	9,932E-03
4	4,79	644,20	72,1	4,76	647,43	71,8	3,064E-04	7,704E-03
5	0,71	575,50	82,5	0,75	559,70	84,0	4,680E-04	9,011E-03
6	3,97	1156,3	59,0	3,95	1155,3	59,3	1,530E-04	4,628E-03
7	0,18	1037,1	84,1	0,19	1053,9	81,5	3,876E-05	2,302E-03
8	3,40	938,40	66,9	3,41	924,04	66,5	8,316E-05	5,139E-03
9	1,97	827,90	15,4	2,00	830,28	15,6	2,408E-04	5,765E-03
10	3,54	154,30	24,9	3,55	154,81	24,5	3,269E-05	3,636E-03
...	...	...	...	...	...	...	...	...
8000	3,26	1084,1	40,5	3,23	1090,9	40,56021	2,481E-04	6,801E-03
						Média:	**1,923E-04**	**4,889E-03**

A.2 *Validation Set*:

N° da amostra	Parâmetros reais (rótulos)			Parâmetros estimados (predições)			Erro Quadrático Médio (*MSE*) entre as Curvas de Resposta	Desvio Padrão (σ) do Erro entre as Curvas de Resposta
	Ganho estático K (adim.)	Constante de tempo τ (s)	Atraso θ (s)	Ganho estático K (adim.)	Constante de tempo τ (s)	Atraso θ (s)		
1	4,82	736,20	18,4	4,80	738,09	18,8	1,164E-04	4,671E-03
2	1,32	218,00	61,1	1,32	222,73	62,4	2,521E-05	2,344E-03
3	4,71	285,40	60,1	4,68	287,19	59,8	2,107E-04	6,899E-03
4	3,91	1093,3	45,8	3,85	1099,1	46,2	9,562E-04	1,250E-02
5	1,92	932,00	40,8	1,93	934,60	40,8	1,845E-05	1,545E-03
6	2,86	469,30	24,1	2,87	464,41	25,0	5,556E-05	4,107E-03
7	3,66	127,80	58,4	3,70	136,35	58,7	4,879E-04	1,407E-02
8	1,03	826,90	24,9	1,10	819,91	25,6	1,380E-03	1,444E-02
9	4,30	542,50	65,3	4,28	542,21	65,2	1,187E-04	4,160E-03
10	1,42	543,10	25,2	1,44	546,79	24,4	9,513E-05	3,586E-03
...	...	...	...	...	...	...	...	...
1000	4,13	218,00	18,5	4,13	216,57	18,1	2,328E-05	2,412E-03
						Média:	**3,014E-04**	**5,665E-03**

APÊNDICE B – CÓDIGO-FONTE *MATLAB* DA SIMULAÇÃO DE RESPOSTA DE MODELOS DINÂMICOS LINEARES A SINAIS *PRBS*

```
%% 1) Escolha de valores aleatórios para K (adimensional), tau (em SEGUNDOS) e delay (em SEGUNDOS):

Ts = 12;        % Ts fixo (em segundos)

valores_K = 10^-2 : 10^-2 : 5;      % entre 0,01 e 5

valores_tau = 120 : 10^-1 : 1200;   % entre 2 e 20 mins

valores_delay = 0 : 10^-1 : 90;     % entre 0 e 90 segs, em intervalos de 12 segs (Ts)


len_K = length(valores_K);

len_tau = length(valores_tau);

len_delay = length(valores_delay);


K = valores_K(randi(len_K));

tau = valores_tau(randi(len_tau));

delay = valores_delay(randi(len_delay));


%% 2) Construção da Função de Transferência

num = [K];

den = [tau 1];

G = tf(num, den, 'InputDelay', delay)


%% 3) Intervalo p/ Tb proposto por (AGUIRRE, 2007):

Tb_min = tau/10;

Tb_max = tau/3;

Tb = 1/2*(Tb_max + Tb_min); %Valor atribuído: ponto no meio do intervalo
```

```
%% 4) Rotina de Construção do PRBS criada por René Descartes e Guilherme Carvalho, com adaptações:

rbs = round(Tb/Ts);

Band = [0 1];

Range = [0 1]; % amplitude do PRBS no eixo Y

Par = [127,1,1];

u1 = idinput(Par,'prbs',Band,Range);

u_prbs = [];

for a = 1:length(u1)
    for b = 1:rbs
        u_prbs = [u_prbs;u1(a)];
    end
end

tfinal = Par(1)*Par(3)*rbs*Ts -Ts;

t2 = 0:Ts:tfinal;


save('prbs.mat','t2','u_prbs');


%% 5) Simulação da Resposta do Sistema Linear ao Sinal PRBS:

lsim(G, u_prbs, t2);
```

```
%% 6) Extração dos dados resultantes:
h = findobj(gca,'Type','line');
x = h(3).XData;
x = x';
y = h(3).YData;
y = y';
sinais = [x u_prbs y];          % Formação de um atributo de entrada para a 1D-CNN
parametros = [K; tau; delay];   % Rótulo correspondente
```

APÊNDICE C – CÓDIGO-FONTE *MATLAB* DA CRIAÇÃO DO *DATASET*

```
clear all; close all; clc;
%% 1) Construção do Training Set:
n_train = 8000;     % tamanho do Training Set
for i = 1:n_train
   fprintf('Trein.: iteracao nº %i/%i', i, n_train)
   run SimulacaoComPRBS    % Construção dos sinais e dos rótulos
   % Salvando sinais no dado de entrada Train_X_(i).csv:
   dlmwrite(['C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/'...
   'Dataset/Training Set/Train_X_' num2str(i) '.csv'], ...
   sinais, 'delimiter', ',', 'precision', 12);
   % Salvando rótulos no arquivo Train_Y_(i).csv:
   dlmwrite(['C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/'...
   'Dataset/Training Set/Train_Y_' num2str(i) '.csv'], ...
   parametros, 'delimiter', ',', 'precision', 12);
end

%% 2) Construção do Validation Set
n_valid = 1000;     % tamanho do Validation Set
for i = 1:n_valid
   fprintf('Valid.: iteracao nº %i/%i', i, n_valid)
   run SimulacaoComPRBS    % Construção dos sinais e dos rótulos
   % Salvando sinais no dado de entrada Valid_X_(i).csv:
```

```
        dlmwrite(['C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive
Spyder/'...
        'Dataset/Validation Set/Valid_X_' num2str(i) '.csv'], ...
        sinais, 'delimiter', ',', 'precision', 12);
        % Salvando rótulos no arquivo Valid_Y_(i).csv:
        dlmwrite(['C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive
Spyder/'...
        'Dataset/Validation Set/Valid_Y_' num2str(i) '.csv'], ...
        parametros, 'delimiter', ',', 'precision', 12);
    end


    %% 3) Número de amostras dos Subconjuntos:
    qtde = [n_train; n_valid];
    dlmwrite(['C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive
Spyder/'...
    'Dataset/n_amostras.csv'], qtde, 'delimiter', ',');
```

APÊNDICE D – CÓDIGO-FONTE *PYTHON*

```
# 1) DEFINIÇÃO DE ROTINAS SUGERIDAS POR (BERT CARREMANS, 2018) COM
ADAPTAÇÕES:

def train_model(model, X_train, y_train, X_val, y_val,
                epochs, batch_size, callbacks=None):
    '''
    Function to train a multi-class model. The number of epochs and
    batch_size are set by the constants at the top of the
    notebook.

    Parameters:
        model : model with the chosen architecture
        X_train : training features
        y_train : training target
        X_val : validation features
        Y_val : validation target

    Output:
        model training history
    '''

    history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
                        verbose=1, callbacks=callbacks,
                        validation_data=(X_val, y_val), shuffle=True)
    return history

def optimal_epoch(model_hist):
    '''
    Function to return the epoch number where the validation loss is
    at its minimum

    Parameters:
        model_hist : training history of model

    Output:
        epoch number with minimum validation loss
    '''

    min_epoch = np.argmin(model_hist.history['val_loss']) + 1
    min_val_loss = np.array(model_hist.history['val_loss']).min()
    print(f'''Minimum validation loss of value: {min_val_loss}
reached in epoch: {min_epoch}''')
    return min_epoch

def eval_metric(model, history, metric_name):
    '''
    Function to evaluate a trained model on a chosen metric.
    Training and validation metric are plotted in a
    line chart for each epoch.

    Parameters:
        history : model training history
        metric_name : loss or accuracy
    Output:
        line chart with epochs of x-axis and metric on
        y-axis
```

```
    '''

    metric = history.history[metric_name]
    val_metric = history.history['val_' + metric_name]

    offset = 0
    for vez in range(0, offset):
        metric[vez] = 0
        val_metric[vez] = 0

    e = range(1, len(model_history.epoch) + 1)

    plt.plot(e, metric, 'b', label='Train ' + metric_name)
    plt.plot(e, val_metric, 'r', label='Validation ' + metric_name)
    plt.xlabel('Epoch number')
    plt.ylabel(metric_name)
    plt.title('Comparing training and validation ' + metric_name + ' for '
              + model.name)
    plt.legend()
    plt.show()


# 2) IMPORTAÇÃO DE BIBLIOTECAS E PACOTES:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Flatten, LeakyReLU
from keras.layers.convolutional import Conv1D, MaxPooling1D
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.optimizers import Adam
from keras.regularizers import l2


# 3) IMPORTAÇÃO E PREPROCESSAMENTO DO DATASET:
channels = 3  # Quantidade de sinais: [tempo (s), prbs (0 or 1), resposta a prbs]

samples = pd.read_csv(filepath_or_buffer='C:/Users/ASUS/OneDrive/' +
                      'TCC - OneDrive/TCC – OneDrive Spyder/Dataset/' +
                      'n_amostras.csv', header=None, squeeze=True)

training_samples = int(samples.values[0])     # Tamanho do Training Set
valid_samples = int(samples.values[1])          # Tamanho do Validation Set

# Qual é o comprimento da sequência temporal mais longa

# do training set?
maximo_train = 0
minimo_train = 0
for numero in range(0, training_samples):
    comprimento_train = pd.read_csv('C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
    'TCC - OneDrive Spyder/Dataset/Training Set/Train_X_' +
    str(numero + 1) + '.csv',
    header=None).to_numpy().shape[0]    # Conversão de pd.DF para np.array
    if numero == 0:
        maximo_train = minimo_train = comprimento_train
        indice_do_max_train = indice_do_min_train = numero
    if maximo_train < comprimento_train:
```

```
        maximo_train = comprimento_train
        indice_do_max_train = numero
    if minimo_train > comprimento_train:
        minimo_train = comprimento_train
        indice_do_min_train = numero
print(f"'maior tamanho do training set: {maximo_train}
index do máx nº: {indice_do_max_train}
menor tamanho do training set: {minimo_train}
index do mín nº: {indice_do_min_train}'")


# e do validation set?
maximo_valid = 0
minimo_valid = 0
for numero in range(0, valid_samples):
    comprimento_valid = pd.read_csv('C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
    'TCC - OneDrive Spyder/Dataset/Validation Set/Valid_X_' +
    str(numero + 1) + '.csv',
    header=None).to_numpy().shape[0]    # Conversão de pd.DF para np.array
    if numero == 0:
        maximo_valid = minimo_valid = comprimento_valid
        indice_do_max_valid = indice_do_min_valid = numero
    if maximo_valid < comprimento_valid:
        maximo_valid = comprimento_valid
        indice_do_max_valid = numero
    if minimo_valid > comprimento_valid:
        minimo_valid = comprimento_valid
        indice_do_min_valid = numero
print(f"'maior tamanho do validation set: {maximo_valid}
index do máx nº: {indice_do_max_valid}
menor tamanho do validation set: {minimo_valid}
index do mín nº: {indice_do_min_valid}'")

# de todo o dataset?
maximo = max(maximo_train, maximo_valid)
minimo = min(minimo_train, minimo_valid)
timesteps = maximo

nb_param = 3    # Nº de parâmetros da FT = 3 ([K, tau, delay])

# 3.1) Training Set:
X_train = np.empty((training_samples, timesteps, channels), dtype=np.float64)
y_train = np.empty((training_samples, nb_param), dtype=np.float64)

for numero in range(0, training_samples):
    # X_train:
    X_aux = pd.read_csv('C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
    'TCC - OneDrive Spyder/Dataset/Training Set/Train_X_' +
    str(numero + 1) + '.csv',
    header=None).to_numpy()     # Conversão de pd.DF para np.array
    # Preenchimento com zeros:
    X_aux = X_aux.transpose()
    X_aux = pad_sequences(X_aux, maxlen=timesteps, dtype='float64',
                          padding='post', truncating='post',
                          value=0)
    X_aux = X_aux.transpose()
    # Salvando em X_train:
    X_train[numero, :, :] = X_aux
```

```
    # y_train:
    y_aux = pd.read_csv('C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
    'TCC - OneDrive Spyder/Dataset/Training Set/Train_Y_' +
    str(numero + 1) + '.csv', header=None,
    squeeze=True).to_numpy()    # Conversão de pd.DF para np.array
    y_train[numero, :] = y_aux

# 3.2) Validation Set:
X_valid = np.empty((valid_samples, timesteps, channels), dtype=np.float64)
y_valid = np.empty((valid_samples, nb_param), dtype=np.float64)

for numero in range(0, valid_samples):
    # X_valid:
    X_aux = pd.read_csv('C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
    'TCC - OneDrive Spyder/Dataset/Validation Set/Valid_X_' +
    str(numero + 1) + '.csv',
    header=None).to_numpy()     # Conversão de pd.DF para np.array
    # Preenchimento com zeros:
    X_aux = X_aux.transpose()
    X_aux = pad_sequences(X_aux, maxlen=timesteps, dtype='float64',
                          padding='post', truncating='post',
                          value=0)
    X_aux = X_aux.transpose()
    # Salvando em X_valid:
    X_valid[numero, :, :] = X_aux
    # y_valid:
    y_aux = pd.read_csv('C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
    'TCC - OneDrive Spyder/Dataset/Validation Set/Valid_Y_' +
    str(numero + 1) + '.csv', header=None,
    squeeze=True).to_numpy()    # Conversão de pd.DF para np.array
    y_valid[numero, :] = y_aux


# 4) FEATURE SCALING:
from sklearn.preprocessing import StandardScaler

# 4.1) Padronização dos canais:
scalers_channel = {}
for i in range(X_train.shape[2]):
    scalers_channel[i] = StandardScaler()
    X_train[:, :, i] = scalers_channel[i].fit_transform(X_train[:, :, i])

for i in range(X_valid.shape[2]):
    X_valid[:, :, i] = scalers_channel[i].transform(X_valid[:, :, i])

# 4.2) Padronização dos rótulos (target variables - y):
scaler_output = StandardScaler()
scaler_output.fit(y_train)    # Fitting the scale to the training labels
y_train = scaler_output.transform(y_train)    # Transforming the training labels
y_valid = scaler_output.transform(y_valid)    # Transforming the validation labels


# 5) DEFINIÇÃO DA ARQUITETURA DA 1D-CNN:
model = Sequential()
model.add(Conv1D(filters=64,
          kernel_size=100,
          strides=2,
          kernel_initializer='he_uniform',
```

```
        use_bias=False,
        input_shape=(timesteps, channels),
        kernel_regularizer=l2(l=0)))
model.add(LeakyReLU(alpha=1e-6))
model.add(MaxPooling1D(pool_size=2, strides=2))
model.add(Conv1D(filters=64,
        kernel_size=100,
        strides=2,
        kernel_initializer='he_uniform',
        use_bias=False,
        kernel_regularizer=l2(l=0)))
model.add(LeakyReLU(alpha=1e-6))
model.add(MaxPooling1D(pool_size=2, strides=2))
model.add(Conv1D(filters=64,
        kernel_size=100,
        strides=2,
        kernel_initializer='he_uniform',
        use_bias=False,
        kernel_regularizer=l2(l=0)))
model.add(LeakyReLU(alpha=1e-6))
model.add(MaxPooling1D(pool_size=2, strides=2))
model.add(Flatten())
model.add(Dense(units=400, kernel_initializer='he_uniform',
            use_bias=False, kernel_regularizer=l2(l=0)))
model.add(LeakyReLU(alpha=1e-6))
model.add(Dense(units=200, kernel_initializer='he_uniform',
            use_bias=False, kernel_regularizer=l2(l=0)))
model.add(LeakyReLU(alpha=1e-6))
model.add(Dense(units=nb_param, kernel_initializer='he_uniform', use_bias=False))
model.compile(optimizer=Adam(learning_rate=1e-3, amsgrad=True),
          loss='mse')
model.name = '1D ConvNet'
model.summary()


# 6) CHAMADA DE CALLBACKS, TREINAMENTO DA 1D-CNN E CONSTRUÇÃO DO GRÁFICO
# PERDA vs ÉPOCAS:
model_checkpoint = ModelCheckpoint(filepath='C:/Users/ASUS/OneDrive/' +
                           'TCC - OneDrive/TCC - OneDrive Spyder/' +
                           'best_model_10.h5',
                           monitor='val_loss', verbose=1,
                           save_best_only=True)
early_stop = EarlyStopping(monitor='loss', patience=500, verbose=1)
ep = 1000
bs = 32
model_history = train_model(model, X_train, y_train, X_valid, y_valid,
                      epochs=ep, batch_size=bs,
                      callbacks=[model_checkpoint, early_stop])
model_min = optimal_epoch(model_history)
eval_metric(model, model_history, 'loss')


# 7) REALIZAÇÃO DAS PREDIÇÕES E ARMAZENAMENTO EM ARQUIVO .CSV:

# 7.1) Predições do Training Set:
prediction_train = model.predict(X_train, verbose=1)
prediction_train = scaler_output.inverse_transform(prediction_train)
y_train = scaler_output.inverse_transform(y_train)
```

```
aux = pd.DataFrame(data=prediction_train)
aux.to_csv(path_or_buf='C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
        'TCC - OneDrive Spyder/Dataset/Dataset Predictions/' +
        'best_model_10_predictions_train.csv', sep=';', header=False,
        index=False)
aux = pd.DataFrame(data=y_train)
aux.to_csv(path_or_buf='C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
        'TCC - OneDrive Spyder/Dataset/Dataset Predictions/' +
        'label_train.csv', sep=';', header=False, index=False)

# 7.2) Predições do Validation Set:
prediction_valid = model.predict(X_valid, verbose=1)
prediction_valid = scaler_output.inverse_transform(prediction_valid)
y_valid = scaler_output.inverse_transform(y_valid)

aux = pd.DataFrame(data=prediction_valid)
aux.to_csv(path_or_buf='C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
        'TCC - OneDrive Spyder/Dataset/Dataset Predictions/' +
        'best_model_10_predictions_valid.csv', sep=';', header=False,
        index=False)
aux = pd.DataFrame(data=y_valid)
aux.to_csv(path_or_buf='C:/Users/ASUS/OneDrive/TCC - OneDrive/' +
        'TCC - OneDrive Spyder/Dataset/Dataset Predictions/' +
        'label_valid.csv', sep=';', header=False, index=False)


# 8) ARMAZENAMENTO DO HISTÓRICO DE TREINAMENTO:
from pickle import dump as pickle_dump
with open('C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/' +
        'BestModelTrainingHistory_10.p', 'wb') as file_pi:
    pickle_dump(model_history, file_pi
```

APÊNDICE E – CONSOLE *IPYTHON*

O console *IPython* exibe o conteúdo abaixo quando o código do Apêndice D é executado. Aqui é possível observar inicialmente a definição da arquitetura da *1D-CNN*, o tipo de cada camada e o número de pesos sinápticos contidos em cada uma delas. Também é informado que essa rede apresenta um número total de 1.200.600 pesos sinápticos.

Todavia, a maior parte do conteúdo desse terminal é o progresso do treinamento. Observa-se, como já mencionado na Seção 6.1 e na Seção 6.3, que a rede neural treina por 1.000 épocas com 8.000 amostras e valida a aprendizagem com outras 1.000 amostras. O tempo de processamento por época foi, em média, 37 segundos, o que levou a um tempo total de treinamento próximo a 10,28 horas. Também é possível notar, conforme os relatos da Seção 6.3, que o aprendizado (ajuste de pesos) na época nº 947 resultou no menor valor de val_loss de todo o treinamento, e, por isso, foram salvas as configurações dessa época.

Os valores de $loss$ são as perdas calculadas no *training set*; e val_loss, as calculadas no *validation set*. Destaca-se que ambas as variáveis são calculadas em amostras que passaram por *feature scaling* e, portanto, não expressam o valor real de perda calculado. Porém, conforme esclarecimentos na Seção 6.3, a partir dessas variáveis ($loss$ e val_loss), é possível monitorar o treinamento, possibilitando a verificação de convergência a um valor mínimo, a detecção de eventual *overfitting*, entre outras atividades.

```
Python 3.7.6 | packaged by conda-forge | (default, Mar 23 2020, 22:22:21) [MSC v.1916
64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.15.0 -- An enhanced Interactive Python.
In [1]: runfile('C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/1DCNN.py',
wdir='C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder')
Using TensorFlow backend.
maior tamanho do training set: 2794
index do máx nº: 118
menor tamanho do training set: 254
index do mín nº: 149
```

maior tamanho do validation set: 2794
index do máx nº: 39
menor tamanho do validation set: 254
index do mín nº: 6
Model: "1D ConvNet"

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 1348, 64)	19200
leaky_re_lu_1 (LeakyReLU)	(None, 1348, 64)	0
max_pooling1d_1 (MaxPooling1	(None, 674, 64)	0
conv1d_2 (Conv1D)	(None, 288, 64)	409600
leaky_re_lu_2 (LeakyReLU)	(None, 288, 64)	0
max_pooling1d_2 (MaxPooling1	(None, 144, 64)	0
conv1d_3 (Conv1D)	(None, 23, 64)	409600
leaky_re_lu_3 (LeakyReLU)	(None, 23, 64)	0
max_pooling1d_3 (MaxPooling1	(None, 11, 64)	0
flatten_1 (Flatten)	(None, 704)	0
dense_1 (Dense)	(None, 400)	281600
leaky_re_lu_4 (LeakyReLU)	(None, 400)	0
dense_2 (Dense)	(None, 200)	80000
leaky_re_lu_5 (LeakyReLU)	(None, 200)	0
dense_3 (Dense)	(None, 3)	600

```
Total params: 1,200,600
Trainable params: 1,200,600
Non-trainable params: 0
_________________________________________________________________
Train on 8000 samples, validate on 1000 samples
Epoch 1/1000
8000/8000 [==============================] - 55s 7ms/step - loss: 0.7105 - val_loss: 0.3406

Epoch 00001: val_loss improved from inf to 0.34061, saving model to C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/best_model_10.h5
Epoch 2/1000
8000/8000 [==============================] - 34s 4ms/step - loss: 0.3476 - val_loss: 0.3501

Epoch 00002: val_loss did not improve from 0.34061
Epoch 3/1000
8000/8000 [==============================] - 35s 4ms/step - loss: 0.3253 - val_loss: 0.2575

Epoch 00003: val_loss improved from 0.34061 to 0.25749, saving model to C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/best_model_10.h5
Epoch 4/1000
8000/8000 [==============================] - 35s 4ms/step - loss: 0.1664 - val_loss: 0.1081

Epoch 00004: val_loss improved from 0.25749 to 0.10813, saving model to C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/best_model_10.h5
Epoch 5/1000
8000/8000 [==============================] - 35s 4ms/step - loss: 0.0846 - val_loss: 0.0828

Epoch 00005: val_loss improved from 0.10813 to 0.08284, saving model to C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/best_model_10.h5
Epoch 6/1000
8000/8000 [==============================] - 35s 4ms/step - loss: 0.0631 - val_loss: 0.0506

Epoch 00006: val_loss improved from 0.08284 to 0.05065, saving model to C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/best_model_10.h5
Epoch 7/1000
8000/8000 [==============================] - 35s 4ms/step - loss: 0.0476 - val_loss: 0.0298
```

```
Epoch 00007: val_loss improved from 0.05065 to 0.02977, saving model to
C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/best_model_10.h5
Epoch 8/1000
8000/8000 [==============================] - 35s 4ms/step - loss: 0.0417 - val_loss: 0.0535

Epoch 00008: val_loss did not improve from 0.02977
Epoch 9/1000
8000/8000 [==============================] - 35s 4ms/step - loss: 0.0374 - val_loss: 0.0289

Epoch 00009: val_loss improved from 0.02977 to 0.02891, saving model to
C:/Users/ASUS/OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/best_model_10.h5
Epoch 10/1000
8000/8000 [==============================] - 36s 4ms/step - loss: 0.0320 - val_loss: 0.0437

Epoch 00010: val_loss did not improve from 0.02891

                                    .
                                    .
                                    .


Epoch 00945: val_loss did not improve from 0.00221
Epoch 946/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0015 - val_loss: 0.0032

Epoch 00946: val_loss did not improve from 0.00221
Epoch 947/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0013 - val_loss: 0.0022

Epoch 00947: val_loss improved from 0.00221 to 0.00220, saving model to C:/Users/ASUS/
OneDrive/TCC - OneDrive/TCC - OneDrive Spyder/best_model_10.h5
Epoch 948/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0011 - val_loss: 0.0030

Epoch 00948: val_loss did not improve from 0.00220
Epoch 949/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0028 - val_loss: 0.0072
```

```
Epoch 00949: val_loss did not improve from 0.00220
Epoch 950/1000
8000/8000 [==============================] - 38s 5ms/step - loss: 0.0033 - val_loss: 0.0052

Epoch 00950: val_loss did not improve from 0.00220

Epoch 951/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0018 - val_loss: 0.0029

Epoch 00951: val_loss did not improve from 0.00220
Epoch 952/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0019 - val_loss: 0.0043

Epoch 00952: val_loss did not improve from 0.00220
Epoch 953/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0027 - val_loss: 0.0036

Epoch 00953: val_loss did not improve from 0.00220
Epoch 954/1000
8000/8000 [==============================] - 38s 5ms/step - loss: 0.0016 - val_loss: 0.0029

Epoch 00954: val_loss did not improve from 0.00220
Epoch 955/1000
8000/8000 [==============================] - 38s 5ms/step - loss: 0.0013 - val_loss: 0.0037

Epoch 00955: val_loss did not improve from 0.00220
Epoch 956/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0016 - val_loss: 0.0044

Epoch 00956: val_loss did not improve from 0.00220
Epoch 957/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0021 - val_loss: 0.0037

Epoch 00957: val_loss did not improve from 0.00220
Epoch 958/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0014 - val_loss: 0.0056
```

```
Epoch 00958: val_loss did not improve from 0.00220
Epoch 959/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0017 - val_loss: 0.0032

Epoch 00959: val_loss did not improve from 0.00220
Epoch 960/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0014 - val_loss: 0.0037

Epoch 00960: val_loss did not improve from 0.00220
Epoch 961/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0018 - val_loss: 0.0027

Epoch 00961: val_loss did not improve from 0.00220
Epoch 962/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0011 - val_loss: 0.0024

Epoch 00962: val_loss did not improve from 0.00220
Epoch 963/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0017 - val_loss: 0.0032

Epoch 00963: val_loss did not improve from 0.00220
Epoch 964/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0015 - val_loss: 0.0031

Epoch 00964: val_loss did not improve from 0.00220
Epoch 965/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0018 - val_loss: 0.0026

Epoch 00965: val_loss did not improve from 0.00220
Epoch 966/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0016 - val_loss: 0.0043

Epoch 00966: val_loss did not improve from 0.00220
Epoch 967/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0019 - val_loss: 0.0042

Epoch 00967: val_loss did not improve from 0.00220
Epoch 968/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0020 - val_loss: 0.0043
```

```
Epoch 00968: val_loss did not improve from 0.00220
Epoch 969/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0021 - val_loss: 0.0037

Epoch 00969: val_loss did not improve from 0.00220
Epoch 970/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0027 - val_loss: 0.0038


Epoch 00970: val_loss did not improve from 0.00220
Epoch 971/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0031 - val_loss: 0.0062

Epoch 00971: val_loss did not improve from 0.00220
Epoch 972/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0024 - val_loss: 0.0036

Epoch 00972: val_loss did not improve from 0.00220
Epoch 973/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0026 - val_loss: 0.0025

Epoch 00973: val_loss did not improve from 0.00220
Epoch 974/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0015 - val_loss: 0.0040

Epoch 00974: val_loss did not improve from 0.00220
Epoch 975/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0018 - val_loss: 0.0048

Epoch 00975: val_loss did not improve from 0.00220
Epoch 976/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0012 - val_loss: 0.0026

Epoch 00976: val_loss did not improve from 0.00220
Epoch 977/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0013 - val_loss: 0.0034
```

```
Epoch 00977: val_loss did not improve from 0.00220
Epoch 978/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0011 - val_loss: 0.0033

Epoch 00978: val_loss did not improve from 0.00220
Epoch 979/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0012 - val_loss: 0.0027

Epoch 00979: val_loss did not improve from 0.00220
Epoch 980/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0017 - val_loss: 0.0037

Epoch 00980: val_loss did not improve from 0.00220
Epoch 981/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0027 - val_loss: 0.0025

Epoch 00981: val_loss did not improve from 0.00220
Epoch 982/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0016 - val_loss: 0.0031

Epoch 00982: val_loss did not improve from 0.00220
Epoch 983/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0013 - val_loss: 0.0023

Epoch 00983: val_loss did not improve from 0.00220
Epoch 984/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0025 - val_loss: 0.0037

Epoch 00984: val_loss did not improve from 0.00220
Epoch 985/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0033 - val_loss: 0.0038

Epoch 00985: val_loss did not improve from 0.00220
Epoch 986/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0024 - val_loss: 0.0032

Epoch 00986: val_loss did not improve from 0.00220
Epoch 987/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0024 - val_loss: 0.0040
```

```
Epoch 00987: val_loss did not improve from 0.00220
Epoch 988/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0015 - val_loss: 0.0032

Epoch 00988: val_loss did not improve from 0.00220
Epoch 989/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0011 - val_loss: 0.0026

Epoch 00989: val_loss did not improve from 0.00220
Epoch 990/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0018 - val_loss: 0.0032

Epoch 00990: val_loss did not improve from 0.00220
Epoch 991/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0017 - val_loss: 0.0033

Epoch 00991: val_loss did not improve from 0.00220
Epoch 992/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0012 - val_loss: 0.0027

Epoch 00992: val_loss did not improve from 0.00220
Epoch 993/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0013 - val_loss: 0.0025

Epoch 00993: val_loss did not improve from 0.00220
Epoch 994/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0017 - val_loss: 0.0024

Epoch 00994: val_loss did not improve from 0.00220
Epoch 995/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0015 - val_loss: 0.0028

Epoch 00995: val_loss did not improve from 0.00220
Epoch 996/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0018 - val_loss: 0.0040

Epoch 00996: val_loss did not improve from 0.00220
Epoch 997/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0036 - val_loss: 0.0036
```

```
Epoch 00997: val_loss did not improve from 0.00220
Epoch 998/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0017 - val_loss: 0.0027

Epoch 00998: val_loss did not improve from 0.00220
Epoch 999/1000
8000/8000 [==============================] - 37s 5ms/step - loss: 0.0013 - val_loss: 0.0023

Epoch 00999: val_loss did not improve from 0.00220
Epoch 1000/1000
8000/8000 [==============================] - 36s 5ms/step - loss: 0.0020 - val_loss: 0.0028

Epoch 01000: val_loss did not improve from 0.00220
Minimum validation loss of value: 0.0021970244902186097
reached in epoch: 947
8000/8000 [==============================] - 11s 1ms/step
1000/1000 [==============================] - 1s 1ms/step
```

In [2]: