UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ENGENHARIA ELETRÔNICA
BACHARELADO EM ENGENHARIA ELÉTRICA

EDUARDO PACHECO CARREIRO BRAGA
MATHEUS HENRIQUE DO AMARAL PRATES

DESENVOLVIMENTO DE UM *FRAMEWORK* DE REDES NEURAIS COM INTERFACE GRÁFICA

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2020

EDUARDO PACHECO CARREIRO BRAGA
MATHEUS HENRIQUE DO AMARAL PRATES

DESENVOLVIMENTO DE UM *FRAMEWORK* DE REDES NEURAIS COM INTERFACE GRÁFICA

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica, do Departamento Acadêmico de Engenharia Eletrônica, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Hugo Valadares Siqueira

PONTA GROSSA
2020

Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEP. ACADEMICO DE ELETRONICA-PG

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO - TCC

DESENVOLVIMENTO DE UM FRAMEWORK DE REDES NEURAIS COM INTERFACE GRÁFICA

Por

Eduardo Pacheco Carreiro Braga e Matheus Henrique do Amaral Prates

Monografia apresentada às 16 horas do dia 03 de dezembro de 2020 como requisito parcial, para conclusão do Curso de Engenharia Elétrica da Universidade Tecnológica Federal do Paraná, Câmpus Ponta Grossa. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação e conferidas, bem como achadas conforme, as alterações indicadas pela Banca Examinadora, o trabalho de conclusão de curso foi considerado APROVADO.

Banca examinadora:

Profª. Marcella Scoczynski Ribeiro Martins	Membro
Profª. Yara de Souza Tadano	Membro
Prof. Hugo Valadares Siqueira	Orientador
Prof. Josmar Ivanqui	Professor responsável TCC-II

Documento assinado eletronicamente por (Document electronically signed by) **MARCELLA SCOCZYNSKI RIBEIRO MARTINS**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em (at) 03/12/2020, às 17:40, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por (Document electronically signed by) **HUGO VALADARES SIQUEIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em (at) 03/12/2020, às 17:40, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por (Document electronically signed by) **YARA DE SOUZA TADANO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em (at) 03/12/2020, às 17:42, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por (Document electronically signed by) **JOSMAR IVANQUI**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em (at) 04/12/2020, às 18:26, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site (The authenticity of this document can be checked on the website) https://sei.utfpr.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador (informing the verification code) **1780400** e o código CRC (and the CRC code) **D2CD80B3**.

Referência: Processo nº 23064.021957/2020-01 SEI nº 1780400

Criado por josmar, versão 3 por josmar em 03/12/2020 07:43:57.

Dedico este trabalho a família e aos meus amigos, pelos momentos de ausência.

AGRADECIMENTOS

Este trabalho não poderia ser terminado sem a ajuda de diversas pessoas e a esta instituição às quais presta-se homenagem. Certamente esses parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase. Portanto, desde já pedimos desculpas àquelas que não estão presentes entre estas palavras, mas que elas estejam certas de que fazem parte dos nossos pensamentos e de nossa gratidão.

As nossas famílias, pelo carinho, incentivo e total apoio em todos os momentos.

Ao nosso orientador, que nos mostrou os caminhos a serem seguidos e pela confiança depositada.

Aos professores e colegas do departamento, que ajudaram de forma direta e indireta na conclusão deste trabalho.

Enfim, a todos os que de alguma forma contribuíram para a realização deste trabalho.

Primeira Lei: Um robô não pode ferir um ser humano ou, por omissão, permitir que um ser humano sofra algum mal. Segunda Lei: Um robô deve obedecer as ordens que lhe sejam dadas por seres humanos, exceto nos casos em que tais ordens contrariem a Primeira Lei. Terceira Lei: Um robô deve proteger sua própria existência desde que tal proteção não entre em conflito com a Primeira e Segunda Leis (ASIMOV, Isaac, 1950).

RESUMO

BRAGA, Eduardo Pacheco Carreiro; PRATES, Matheus Henrique do Amaral. **Desenvolvimento de um *Framework* de Redes Neurais com Interface Gráfica**. 2020. 69 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2020.

Impulsionada pelos avanços em *hardwares* dos dispositivos e por equipamentos como nas tecnologias da Internet das Coisas, a quantidade de informações geradas e a interação com computadores cresceram fortemente nas últimas décadas. Com isso, o interesse por Aprendizado de Máquina (AM) tornou-se cada vez maior. Logo, o espaço para o desenvolvimento de ferramentas que facilitam a aplicações de redes neurais artificiais se torna cada vez maior. Porém, as ferramentas já existentes muitas vezes não apresentam uma interface agradável e simples de usar, principalmente para os usuários menos experientes. Para isso, este trabalho tem como objetivo apresentar o desenvolvimento de uma ferramenta gráfica denominada *framework* para modelagem da rede neural artificial Perceptron de Múltiplas Camadas, desenvolvido na linguagem de programação *Python*. Este *software* visa proporcionar ao usuário uma experiencia intuitiva, com fácil extração de informações das bases de dados compatíveis como *Microsoft Excel* (.xlsx), *Matlab* (.mat), arquivo de texto (.txt) e Valores Separados por Vírgulas (.csv). Os dados uma vez inseridos poderão ser pré tratados através dos métodos de normalização, dessazonalização e codificação distribuída. Desta forma, usuários com pouca experiência em redes neurais artificiais poderão desenvolver modelos treinados e visualizar os resultados de maneira gráfica.

Palavras-chave: Rede Neural. Aprendizado de Maquina. Python. Interface Gráfica. Framework.

ABSTRACT

BRAGA, Eduardo Pacheco Carreiro; PRATES, Matheus Henrique do Amaral. **Development of a Neural Network Framework With a Graphical Interface**. 2020. 69 p. Final Coursework (Bachelor's Degree in Electrical Engineering) – Federal University of Technology – Paraná. Ponta Grossa, 2020.

Driven by advances in device hardware and equipment such as Internet of Things technologies, the amount of information generated and the interaction with computers have grown strongly in recent decades. As a result, interest in AM became increasingly greater. Therefore, the space for the development of tools that facilitate the application of artificial neural networks becomes increasingly larger. However, existing tools often do not have a pleasant and simple to use interface, especially for less experienced users. For that, this work aims to present the development of a graphical tool called framework for modeling the artificial neural network Multilayer Perceptron, developed in the programming language python. This software aims to provide the user with an intuitive experience, with easy extraction of information from compatible databases such as Microsoft Excel (.xlsx), Matlab (.mat), Text file (.txt) and Values Separated by Commas (.csv). The data once entered may be pre-treated using the normalization, seasonally adjusted and distributed coding methods. In this way, users with little experience in artificial neural networks can develop trained models and visualize the results graphically.

Keywords: Neural Network. Machine Learnig. Python. Graphic Interface. Framework.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

SIGLAS

AM	Aprendizado de Máquina
CEPEA	*Centro de Estudos Avançados em Economia Aplicada*
ELU	*Exponential Linear Unit*
GUI	*Graphical User Interface*
LSM	*Least Mean Square*
MLP	*Multilayer Perceptron*
MSE	*Mean Squared Error*
ReLU	*Rectified Linear Unit*
RNAs	Redes Neurais Artificiais

SUMÁRIO

1 INTRODUÇÃO

Impulsionada pelos avanços em *hardwares* dos dispositivos e por equipamentos como nas tecnologias da Internet das Coisas, a quantidade de informações geradas e a interação com computadores cresceram fortemente nas últimas décadas (JUNIOR, 2018). Com isso, o interesse por AM tornou-se cada vez maior.

As Redes Neurais Artificiais (RNAs), como o próprio nome sugere, foram inspiradas na rede neural biológica, em que um neurônio quando recebe uma informação é ativado e transmite-a de forma modulada para os próximos neurônios, até que a tarefa seja concluída. Como é sabido, o cérebro humano é capaz de processar uma quantidade incontável de tarefas em pouco tempo (SCHWARTZ; KANDEL, 2014). Por exemplo, um goleiro, que ao ver uma bola em sua direção com grande velocidade, dispara todas as funções necessárias para saltar em defesa em uma fração de segundos. Assim também são as RNAs, executando diversas ações para chegar em um resultado (CARVALHO, 2020).

Partindo-se da observação do cérebro humano, Warren McCulloch e Walter Pitts, em 1943, modelaram o primeiro neurônio artificial, de maneira simplificada (HAYKIN, 2003). Outra contribuição importante foi a de Rosenblatt (1958), que propôs o *perceptron* junto com um algoritmo de treinamento para determinação dos pesos. Seguindo os avanços no campo das RNAs, Widrow e Hoff (1960) apresentaram o algoritmo *Least Mean Square* (LSM) e o utilizaram para desenvolver o Adaline (*Adaptive Linear Element*), que se baseava nesta regra de aprendizagem. A diferença principal entre o perceptron e o Adaline é o modelo de treinamento (SIQUEIRA, 2013).

Já a grande contribuição que impulsionou as aplicações de RNAs foi a publicação apresentada por Rumelhart, Hinton e Williams (1986), que propôs a aplicação do *backpropagation* de Werbos (1974): o algoritmo de retropropagação do erro. Esta proposta ainda se mostra relevante ao ponto de este ser o algoritmo mais comumente aplicado no treinamento da arquitetura Perceptron de Múltiplas Camadas - MLP (SIQUEIRA, 2013).

Hoje, as RNAs podem ser aplicadas em praticamente todas as áreas de conhecimento e não somente em ciências exatas (SILVA DANILO HERNAME SPATTI, 2010), podendo ser utilizadas para classificação de dados, identificação de pessoas em imagens ou vídeos, reconhecimento de padrões, dentre outras (GERON, 2019). Contudo, a etapa de implementação destes algoritmos envolve diversos passos, em termos de conhecimentos de ferramentas matemáticas avançadas, como cálculo diferencial, geometria analítica e estatística. Já no campo da ciência da computação, são necessários conhecimentos em programação e manipulação de dados.

Com isso, o espaço para o desenvolvimento de ferramentas que facilitam a

aplicações de redes neurais artificiais se torna cada vez maior. Então, com este trabalho, pretende-se apresentar um *framework* amigável, para que pesquisadores e estudantes de diversas áreas do conhecimento possam utilizar corretamente as RNAs, além de desfrutar de etapas de pré-processamento, seleção de entradas e visualização gráfica dos dados de entrada ou dos resultados.

1.1 OBJETIVOS

O objetivo principal deste trabalho é desenvolver um *Framework* em linguagem *Python* que possibilite usuários com pouca experiencia em AM ou RNAs navegar por uma interface gráfica intuitiva de modo que, a partir da inserção do conjunto de dados, seja possível obter uma rede neural ajustada.

O objetivo geral será atingido após desenvolvidas as etapas de revisão bibliográfica e implementação computacional.

Então, para atingir os objetivos específicos, deve-se desenvolver um *Framework* que possua:

- Uma interface gráfica
- Aquisição e visualização de dados
- Opções de pré-processamento
- Opção de execução do modelo

1.2 JUSTIFICATIVA

Como principal motivação e impacto, espera-se ao apresentar este trabalho, trazer a possibilidade de aplicação de redes neurais para usuários menos experientes na área de aprendizado de máquina. Dessa forma, pretende-se permitir que os usuários das mais diversas áreas do conhecimento possam utilizar esta ferramenta, e assim garantir uma análise mais bem fundamentada de seus dados.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido em 5 capítulos, sendo o número 2 uma síntese sobre técnicas de pre-processamento, a história das RNAs, e elementos fundamentais para o funcionamento das redes neurais. No capítulo 3 são apresentadas a descrição das etapas desenvolvidas do *framework* e suas respectivas telas. Para o 4º capítulo está reservada a apresentação prática do *framework*, demonstrando a utilização para

duas aplicações diferentes. No último capítulo estarão as considerações finais e conclusões do trabalho.

2 INTRODUÇÃO À APRENDIZADO DE MÁQUINA E REDES NEURAIS

Neste capítulo são abordados os conceitos de pré-processamento de dados, história das redes neurais artificiais e o que são elas, tipos de AM, RNAs simples e profundas. Também são introduzidas as ideias sobre as funções de ativação, divisão de amostras do *dataset*, os conceitos de regressão e classificação, gradiente descendente e validação cruzada para redes Perceptron de múltiplas camadas.

2.1 PRÉ-PROCESSAMENTO

Antes de entender o que são AM e RNAs, é necessário compreender a importância dos cuidados com os dados que serão processados pela rede. Muitas das vezes, os dados que são coletados não estão ajustados e preparados para alimentar uma rede neural artificial. Por exemplo, uma base de dados pode conter uma coluna com valores entre 0 a 100 e outra entre 10000 a 1000000, ou seja, escalas bem diferentes. Para esse caso, é interessante aplicar algum método de pré-processamento, transformando os dados para a mesma escala. Esta seção mostrará uma breve introdução aos métodos de pré-processamento implementados neste trabalho.

2.1.1 *Min Max Scaler* - Normalização

O *Min Max Scaler* é um método de normalização que dimensiona e traduz cada característica (dimensão) das amostras individualmente, de modo que estas estejam na faixa especificada no conjunto de treinamento, por exemplo, entre 0 e 1, ou entre -1 e 1. Um exemplo desse método é apresentado na Figura 1, em que a tabela a esquerda mostra os dados antes de aplicar a normalização, enquanto na tabela da direita são os dados pós normalização.

Figura 1 – Base de dados Wine

	alcohol	malic_acid	ash	alcalinity_of_ash	proline	class_0	class_1	class_2
0	14.23	1.71	2.43	15.6	1065.0	1.0	0.0	0.0
1	13.20	1.78	2.14	11.2	1050.0	1.0	0.0	0.0
2	13.16	2.36	2.67	18.6	1185.0	1.0	0.0	0.0
3	14.37	1.95	2.50	16.8	1480.0	1.0	0.0	0.0
4	13.24	2.59	2.87	21.0	735.0	1.0	0.0	0.0

	alcohol	malic_acid	ash	alcalinity_of_ash	proline	class_0	class_1	class_2
0	0.842105	0.191700	0.572193	0.257732	0.561341	1.0	0.0	0.0
1	0.571053	0.205534	0.417112	0.030928	0.550642	1.0	0.0	0.0
2	0.560526	0.320158	0.700535	0.412371	0.646933	1.0	0.0	0.0
3	0.878947	0.239130	0.609626	0.319588	0.857347	1.0	0.0	0.0
4	0.581579	0.365613	0.807487	0.536082	0.325963	1.0	0.0	0.0

Fonte: Autoria própria

2.1.2 *One Hot Encoder* - Codificação distribuída

Um dos problemas ao utilizar RNAs é a impossibilidade de lidar com rótulos não numéricos, para isso é necessário utilizar a codificação distribuída. Muitas das aplicações exigem a classificação dos dados em grupos específicos, como no caso da base IRIS (LEARN, 2020), em que a saída da rede determinará o tipo de flor a partir das características da sépala. A Figura 2, ilustra, na tabela a esquerda a saída (*Target*) em classes, enquanto a direita estão os dados tratados com o método de codificação distribuída.

Figura 2 – Base de dados IRIS

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	1.0	0.0	0.0
1	4.9	3.0	1.4	0.2	1.0	0.0	0.0
2	4.7	3.2	1.3	0.2	1.0	0.0	0.0
3	4.6	3.1	1.5	0.2	1.0	0.0	0.0
4	5.0	3.6	1.4	0.2	1.0	0.0	0.0

Fonte: autoria própria

Então, pode-se definir a codificação distribuída em uma representação de variáveis categóricas como vetores binários. Isso primeiro requer que os valores categóricos sejam mapeados para valores inteiros e, em seguida, cada valor inteiro é transformado em um vetor binário que contém todos os valores zero, exceto o índice da classe.

2.1.3 *Z-Score* - Dessazonalização

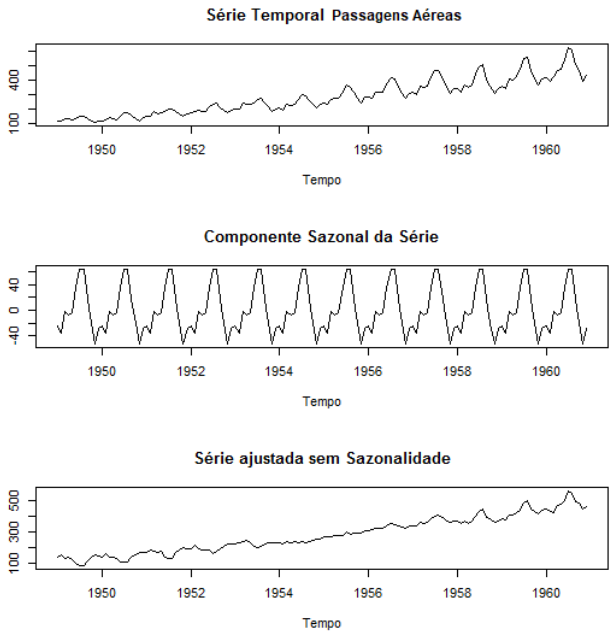
Em alguns casos práticos, os conjuntos de dados podem apresentar sazonalidade, especificamente quando há relação temporal entre as amostras. Isso ocorre quando há variações no padrão da série em intervalos regulares específicos. Nesse caso, a dessazonalização é necessária para deixar as dados aproximadamente estacionários, facilitando seu processo de previsão. O método que retira a parte sazonal da série é chamado de *Z-Score* ou *standard score*.

Um exemplo da ocorrência desse efeito é o aumento no preço das passagens de avião, como apresentado na Figura 3.

2.2 BREVE HISTÓRIA DAS REDES NEURAIS ARTIFICIAIS

A quantidade de informações que o cérebro humano pode armazenar, as quais são utilizadas no processo de aprendizagem, ainda não puderam ser quantificadas, segundo Izquierdo (2002). Porém, sabe-se que existem células responsáveis pelo transporte e processamento das informações que são recebidas, os neurônios.

Figura 3 – Preço de passagens aéreas



Fonte: Action (2020)

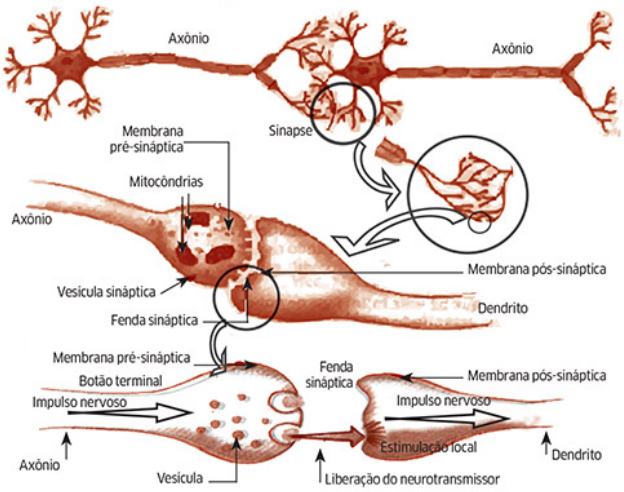
O sistema nervoso é como uma rede de bilhões de neurônios e, a partir das trocas de informações entre os eles, o cérebro é capaz de desenvolver regras próprias para aprender com experiências do passado. Os neurônios são formados por dendritos, um conjunto de terminais de entrada, pelo corpo central e, pelos axônios que são longos terminais de saída. As trocas de informações entre neurônios é chamada de sinapse, ou a região onde dois neurônios entram em contato e através da qual os impulsos nervosos são transmitidos entre eles.

Um neurônio A recebe os impulsos que, em um determinado momento, são processados, quando atingem o limiar de ação, o neurônio A dispara, produzindo uma substância neurotransmissora que fluirá do corpo celular para o axônio, que está conectado a um dendrito de um outro neurônio B. O neurotransmissor tem o poder de diminuir ou aumentar a polaridade da membrana pós-sináptica, que resulta inibindo ou excitando a geração de pulsos no neurônio B. Este processo depende de vários fatores, por exemplo a geometria da sinapse e o tipo de neurotransmissor (CARVALHO, 2020).

A informação transmitida pelos neurônios, na realidade, são impulsos elétricos, que propagam um estímulo ao longo dos neurônios. Estes podem ser qualquer sinal captado pelos receptores nervosos, como é mostrado na Figura 4.

Uma forma de modelar o funcionamento dos neurônios considera que os sinais elétricos gerados em sensores como retina ocular e papilas gustativas são recebidos pelos axônios. Se estes sinais forem superiores a um limiar de disparo (*threshold*), seguem pelo axônio, caso contrário, são bloqueados. Um neurônio recebe sinais atra-

Figura 4 – Troca de informação entre neurônios



Fonte: Paula (2015)

vés de inúmeros dendritos, onde são ponderados e enviados para o axônio da próxima célula. Na passagem por um neurônio, o sinal pode ser amplificado ou atenuado, dependendo do dendrito de origem. Isto se deve à famosa plasticidade sináptica, ou à capacidade da célula nervosa modular o sinal que recebe (ACADEMY, 2020).

Cada região do cérebro é especializada em uma dada função, por exemplo, processamento de sinais auditivos e sonoros, de pensamentos e sentimentos. Tal processamento ocorre através das ligações particulares entre as redes de neurônios, realizado em paralelo. Cada região do cérebro possui um arranjo distinto, variando o número de neurônios, sinapses por neurônio, etc.

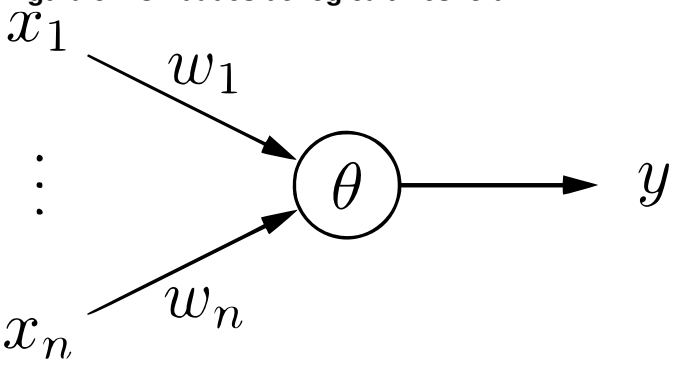
Quando pesquisadores tentaram entender e replicar o comportamento dos neurônios biológicos, o neurofisiologista Warren McCulloch e o matemático Walter Pitts em 1943 interpretaram o comportamento das redes neurais biológicas através de circuitos elétricos (MCCULLOCH; PITTS, 1943). Neste trabalho, os autores mostraram seu modelo matemático e algoritmo denominados lógica de limiar (*threshold logic*), emulando o que ocorre com os neurônios biológicos. Assim, para que a informação se propague, é necessário estímulo suficiente para ativá-lo.

Neste modelo, os impulsos elétricos provenientes de outros neurônios são representados pelos chamados sinais de entrada (x_n), que nada mais são do que os dados que alimentam a rede neural artificial. Dentre os vários estímulos recebidos, alguns excitarão mais e outros menos, o neurônio receptor. Quanto maior o valor do peso w_i, mais excitatório é o estímulo.

A soma ou corpo da célula θ é representada por uma composição de dois módulos: o primeiro é a junção aditiva, onde ocorre o somatório dos sinais de entrada multiplicados pelo seu peso e, posteriormente uma função de ativação, que definirá as entradas e pesos sinápticos, a qual será a saída do neurônio. O axônio é aqui

representado pela saída y obtida pela aplicação da função de ativação. Assim como no modelo biológico, o estímulo pode ser excitatório ou inibitório, representado pelo peso sináptico positivo ou negativo, respectivamente. Essa estrutura é apresentada por Borgelt (2017), na Figura 5.

Figura 5 – Unidades de lógica *threshold*



Fonte: Borgelt (2017).

O modelo proposto em 1943 possui uma natureza binária, pois tanto os sinais de entrada quanto os de saída, admitem apenas dois valores. McCulloch (1943) acreditava que o funcionamento do sistema nervoso central possuía um caráter binário, ou seja, um neurônio influencia ou não o outro de maneira discreta.

Hoje, o neurônio matemático mais utilizado na literatura recebe um ou mais sinais de entrada, binários ou não, e devolve um único sinal de saída, que pode ser distribuído como sinal de saída da rede, ou como sinal de entrada para um ou vários outros neurônios da camada seguinte que formam a rede neural artificial.

Os dendritos e axônios são representados matematicamente apenas pelas sinapses, a intensidade da ligação é representada por uma grandeza denominada peso sináptico, simbolizada pela letra w_n. Quando as entradas x_n, são apresentadas ao neurônio, multiplicadas pelos pesos sinápticos correspondentes, geram as entradas ponderadas. Então, x_1 que multiplica w_1, e assim por diante. A multiplicação de matrizes descreve uma das bases matemáticas do funcionamento de uma rede neural artificial como é mostrado na Equação 1.

$$\begin{bmatrix} 1 & x_{11} & x_{12} & ... & x_{1d} \\ 1 & x_{21} & x_{22} & ... & x_{2d} \\ 1 & ... & ... & .... & ... \\ 1 & x_{n1} & x_{n2} & ... & x_{nd} \end{bmatrix} \times \begin{bmatrix} w_0 \\ w_1 \\ ... \\ w_d \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ ... \\ y_n \end{bmatrix} \tag{1}$$

O neurônio então totaliza todos os produtos, gerando um único resultado. Esse valor, é então apresentado a uma função de ativação ou função de transferência, que têm a finalidade de evitar o acréscimo progressivo dos valores de saída ao longo

das camadas da rede, visto que tais funções possuem valores máximos e mínimos contidos em intervalos determinados. O uso de funções de ativação não-lineares torna a rede neural uma ferramenta poderosa (ACADEMY, 2020).

2.3 TIPOS DE APRENDIZADO DE MÁQUINA

O aprendizado de máquina pode ser definido como a capacidade de um algoritmo em realizar uma tarefa a partir de uma experiência similar, buscando melhorar sua performance. Em outras palavras, um algoritmo pode aprender a atingir um objetivo a partir de um grande volume de dados, suas experiências. Como tipos de aprendizado existem dois grandes grupos:

- Aprendizado Supervisionado
- Aprendizado Não Supervisionado

O aprendizado supervisionado é baseado em dados que possuem marcações que relacionam a entrada e a saída. Para que uma rede possa distinguir entre uma imagem de um gato e um cachorro, por exemplo, antes ela deve ser exposta à figuras previamente rotuladas (RAMASUBRAMANIAN; MOOLAYIL, 2019). Dessa forma, a rede aprende os padrões nas imagens e consegue predizer a qual grupo a imagem pertence.

Já no cenário onde os dados não estão devidamente marcados, torna-se difícil obter saídas corretas, deixando os resultados meramente exploratórios ou para clusterização.

Entre as técnicas utilizadas para resolver problemas de aprendizado supervisionado estão redes neurais artificiais, regressão linear, regressão logística, máquina de suporte vetorial, etc. O aprendizado de máquina supervisionado é a área que concentra a maioria das aplicações bem sucedidas e bem definidas (HONDA; FACURE; YAOHAO, 2017).

Pode-se dizer que o aprendizado não supervisionado tenta, de certa forma simular o processo de aprendizado humano, pois esse método busca aprender de maneira implícita, sem explicação ou sem um guia que aponte a resposta correta, como no aprendizado supervisionado.

Dessa forma, o aprendizado não supervisionado busca nos dados, padrões e semelhanças que podem fornecer mais informações sobre o montante de dados, ou até separar os dados para serem marcados e utilizados com aprendizado supervisionado.

Já os exemplos de aplicações de aprendizado não supervisionados são sistemas de sugestão inteligente para filmes ou músicas, como os existentes em plata-

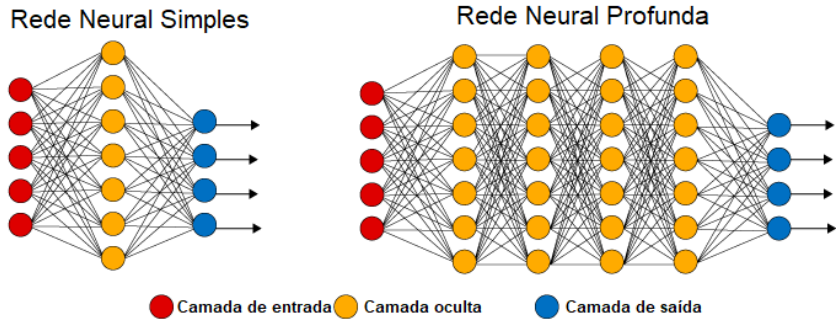
formas como *Netflix* e *Spotify*. Entre as técnicas utilizadas para resolver problemas de aprendizado não supervisionado estão redes neurais artificiais, clusterização k-médias, máquina de suporte vetorial, etc (HONDA; FACURE; YAOHAO, 2017).

2.4 REDES NEURAIS ARTIFICIAIS SIMPLES E PROFUNDAS

Aprendizagem Profunda ou *Deep Learning*, é uma sub-área do Aprendizado de Máquina, na qual se emprega modelos matemáticos para processamento de dados. Redes de Aprendizado Profundo utilizam várias camadas de neurônios artificiais para tratar problemas como reconhecimento de fala humana e de objetos. As informações são passadas através das camadas, a partir da saída da camada anterior. A primeira camada de uma rede neural artificial é chamada de entrada, já a última é a de saída. Todas as camadas entre as duas são chamadas de ocultas ou escondidas. Cada uma é tipicamente um algoritmo simples e uniforme contendo uma função de ativação.

Aprendizagem profunda é o termo usado para nomear os métodos de treinamento de RNAs que realizam o aprendizado de características de forma hierárquica, de tal forma que, características nos níveis mais altos sejam formadas pela combinação de características de mais baixo nível (BEZERRA, 2016). Porém, muitos dos problemas de hoje ainda podem ser resolvidos a partir de redes simples. A Figura 6 mostra um exemplo de uma rede simples e uma de aprendizado profundo.

Figura 6 – Exemplo de arquiteturas de redes neurais

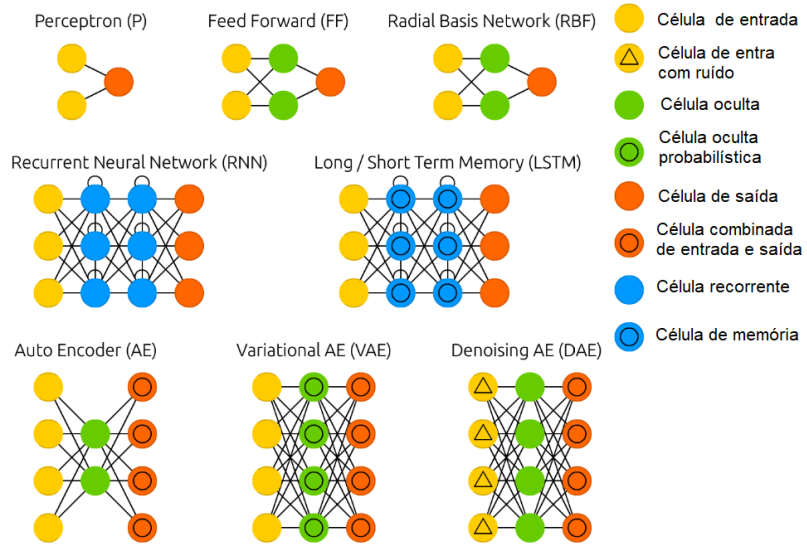


Fonte: Adaptado de Primo.ia (2020)

Os estudos de redes neurais com múltiplas camadas ocultas têm motivação na capacidade humana, que através do córtex visual é capaz de reconhecer imagens. Quando a luz é refletida para retina, sabe-se que essa é estimulada, dispara e percorre uma sequência de regiões do cérebro. Essas regiões são responsáveis em identificar características específicas. As primeiras regiões da rede neural que processam a imagem são repensáveis por identificar coisas mais simples, como as bordas e cantos das imagens. Conforme as informações fluem, as ultimas camadas geram imagens mais complexas.

Já as relações de ligação entre um neurônio e outro (ou entre ele mesmo), definem a arquitetura da rede, como pode ser visto na Figura 7.

Figura 7 – Exemplo de arquiteturas de redes neurais



Fonte: Adaptado de Veen (2016)

Dessa forma, cada região de neurônios combina padrões detectados pela região imediatamente anterior, para formar características mais complexas. Esse processo se repete até que os neurônios da região final são capazes de detectar características de mais alto nível de abstração, tais como faces ou objetos específicos (BENGIO; COURVILLE; VINCENT, 2013).

Desde o trabalho de Hornik (1991), que generalizou o Teorema da Aproximação Universal proposto por Cybenko (1989), sabe-se que, contanto que se possa definir uma quantidade suficiente de neurônios em uma única camada intermediária, é possível aproximar qualquer função contínua.

2.5 FUNÇÕES DE ATIVAÇÃO

Nos últimos anos, muitos trabalhos vêm sendo publicados buscando métodos para aperfeiçoamentos das RNAs. Uma parte importante destes relatam o papel das funções de ativação. Tais funções têm a capacidade de, a partir das entradas, definir se esta é ativada e assim, determinar as saídas da rede, bem como sua assertividade de treinamento. Outro impacto direto das funções de ativação é sua capacidade de alterar a velocidade de convergência (KARLIK; OLGAC, 2011). Além disso, elas podem limitar a amplitude do sinal de saída.

Pode-se definir função de ativação como sendo uma função matemática que está dentro do neurônio artificial trabalhando os dados de entrada para serem entregues para a próxima camada. Em outras palavras, estas são uma maneira de mapear

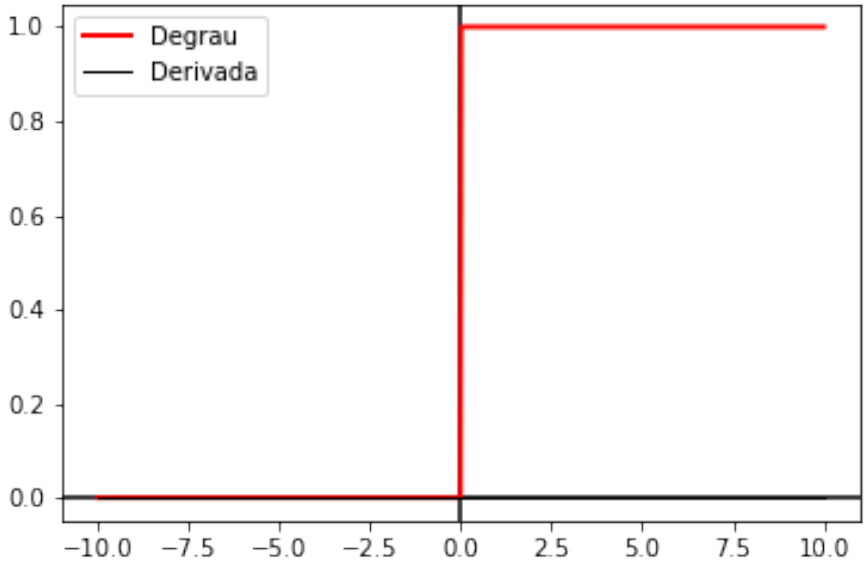
um sinal de entrada para a saída, de maneira não-linear, na maioria dos casos.

Uma das grandes vantagens das funções de ativação é a capacidade de utilizar funções não lineares, já que vários dos problemas práticos apresentam não linearidades, como o caso de reconhecimento de padrões gráficos. A seguir serão descritas algumas das funções mais populares da literatura.

2.5.1 Função Degrau

A função degrau pode ser definida como função de limiar e possui característica binária. Quando um valor de entrada multiplicado pelo peso chega ao neurônio, esse compara se o valor está acima ou abaixo do limiar de ativação. Caso o valor esteja acima, o neurônio passa a informação adiante, exatamente da mesma forma como entrou, para a próxima camada (ACADEMY, 2020). A Figura 8 apresenta a forma gráfica para a função degrau.

Figura 8 – Forma gráfica da função degrau



Fonte: Autoria própria

Equação da função degrau:

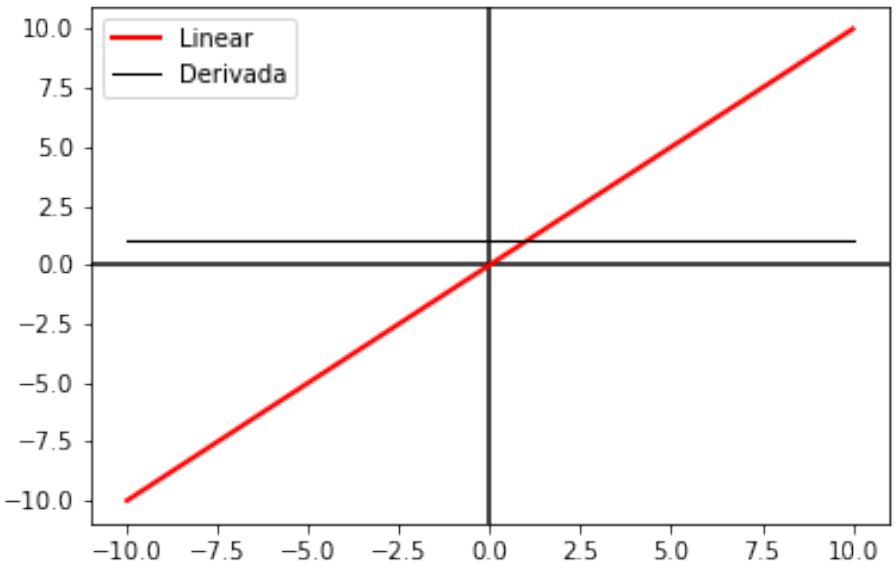
$$\theta(x) = \begin{cases} 1, & se \quad x \geq 0 \\ 0, & se \quad x < 0 \end{cases} \tag{2}$$

Um dos problemas em utilizar a função degrau é sua incapacidade de separar em vários grupos de saída.

2.5.2 Função Linear

Com a função linear, as entradas multiplicadas pelos pesos são passadas para a próxima camada de maneira proporcional à entrada, como mostra a Figura 9. Desta forma, é possível ter mais de uma categoria de saída. Contudo, essa função quando usada na camada intermediária dificulta a utilização de gradiente descendente e também por consequência o *backpropagation*, uma vez que a derivada de uma função linear é sempre uma constante, perdendo a capacidade de relacionar a entrada com o valor da derivada (MISSINGLINK, 2020).

Figura 9 – Forma gráfica da função linear



Fonte: Autoria própria

A função linear é dada pela equação 3:

$$f(x) = ax + b \tag{3}$$

Outro problema enfrentado ao se utilizar apenas funções lineares para a ativação de neurônios artificiais é que não importam quantas camadas possuem na rede neural, elas sempre acabam convergindo para apenas uma, já que a combinação linear de várias funções lineares continua sendo uma função linear. Por esse motivo, as funções lineares só permitem a rede neural executar tarefas simples, limitando a capacidade de lidar com problemas mais complexos e diferentes tipos de entrada.

2.5.3 Funções de Ativação Não Lineares

As funções não lineares são importantíssimas para a modelagem de redes neurais artificiais modernas, por conta da capacidade intrínseca que elas inserem de criar modelos complexos para realizar mapeamento entre entradas e saídas. Permitem que a rede aprenda a lidar com entradas que possuem características não lineares.

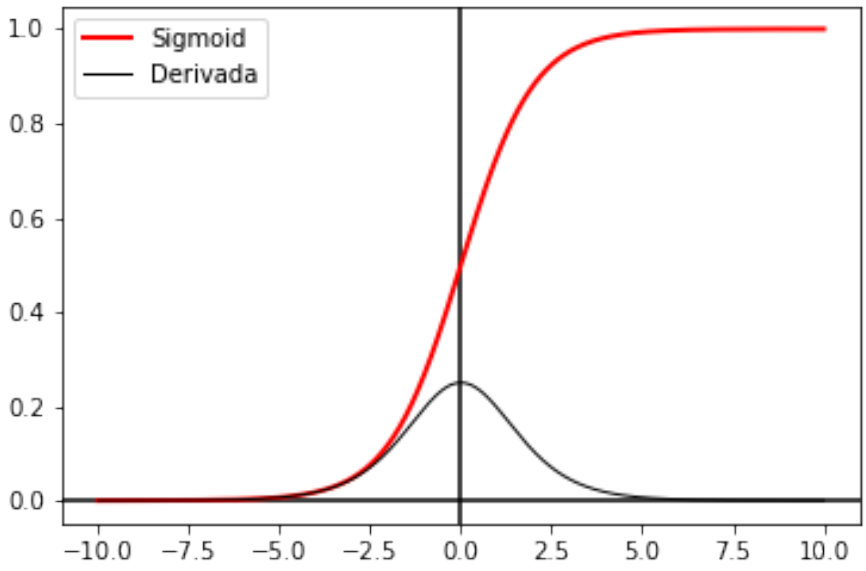
De certa forma, qualquer processo conhecido pode ser modelado matematicamente a partir de funções não lineares. Dessa forma, as RNAs que utilizam esse tipo de função de ativação ganham uma grande aplicabilidade.

Como as derivadas de funções não-lineares mantêm sua relação com a entrada é possível utilizar *backpropagation*. Também como benefício, ressalta-se a capacidade da rede neural operar com múltiplas camadas ocultas e assim, poder vir a aprender *sets* mais complexos com maior precisão.

2.5.3.1 Função sigmoide

A função sigmoide possui um gradiente suave, como mostra a Figura 10, o que previne um salto muito grande entre um valor e outro (ACADEMY, 2020). Esta também limita os valores de saída em um máximo de 1 e mínimo em 0, saturando os valores de saída dos neurônios artificiais. Por isso, é importante a normalização das entradas previamente à sua exposição à rede.

Figura 10 – Forma gráfica da função sigmoide



Fonte: Autoria própria

A expressão da função sigmoide é dada pela equação 4, em que β é uma constante arbitrária que altera a inclinação da função:

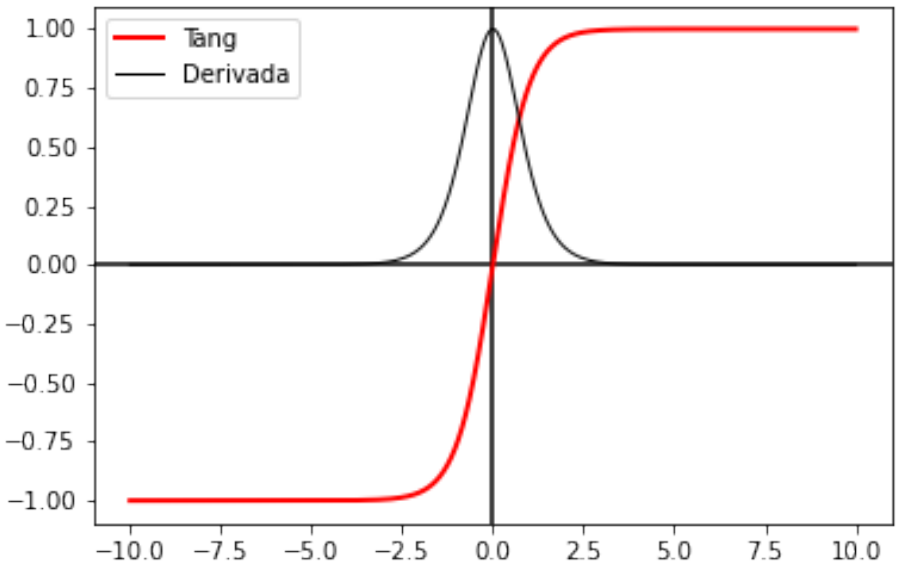
$$\sigma(x) = \frac{1}{1 + e^{-\beta x}} \tag{4}$$

Um dos problemas encontrados ao utilizar a função sigmoide é o *vanishing gradient*. Isto ocorre quando os valores são muito pequenos ou muito grandes, pois o gradiente da função nesses pontos é muito pequeno. Isso pode causar perdas na capacidade de aprendizado na rede neural.

2.5.3.2 Função tangente hiperbólica

A função tangente hiperbólica, Figura 11, é similar à sigmoide, porém possui a vantagem de ser centrada na origem do plano cartesiano. Desta forma, essa função favorece dados que possuem valores fortemente negativos, neutros e positivos (MISSINGLINK, 2020).

Figura 11 – Forma gráfica da função tangente hiperbólica



Fonte: Autoria própria

A função tangente hiperbólica é dada pela equação 5.

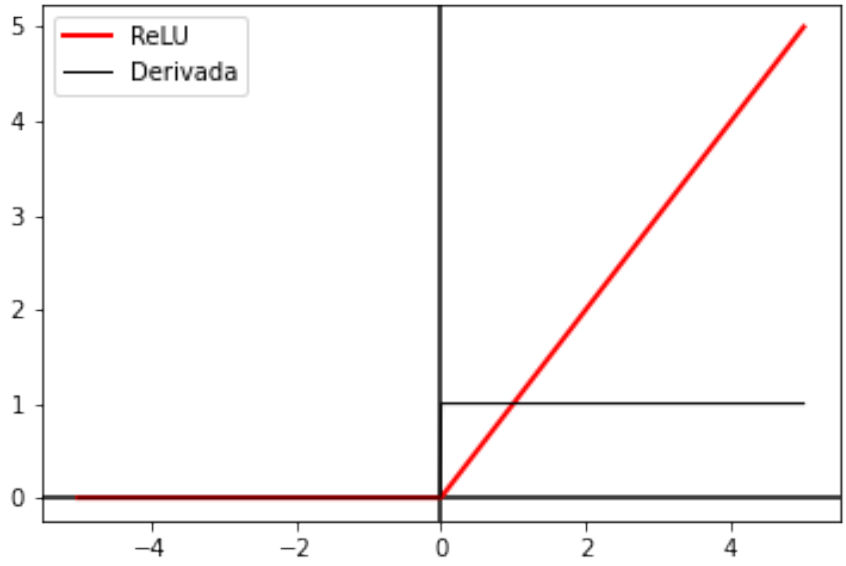
$$tanh(x) = 2\sigma(2x) - 1 \tag{5}$$

Como pode ser visto na equação 5, a função tangente hiperbólica é uma versão escalonada da função sigmoide, de modo que o *vanishing gradient* continua presente.

2.5.3.3 ReLU

Como uma das funções de ativação mais utilizadas em redes neurais profundas, a ReLU tem uma performance significativa para treinos e tarefas dinâmicas (RAMACHANDRAN; ZOPH; LE, 2017). A função mostrada na Figura 12 é não-linear, o que permite a aplicação do *backpropagation*. Também pode ser diferenciada, exceto na origem, em que a derivada pode ser arbitrada em 1 ou 0.

Figura 12 – Forma gráfica da função ReLU



Fonte: Missinglink (2020)

A função ReLU é dada pela equação 6.

$$f(x) = x^{+} = max(0,x) \tag{6}$$

Uma das vantagens de se utilizar ReLU é o seu baixo custo computacional, se comparada à tangente hiperbólica ou sigmoide, necessitando menos iterações para ser resolvida (MISSINGLINK, 2020).

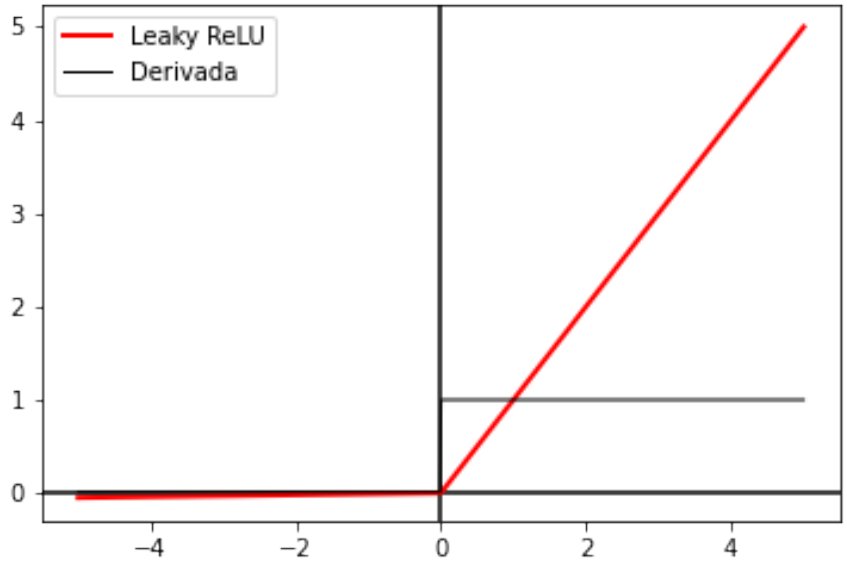
As desvantagens da ReLU aparecem quando as entradas se aproximam de zero ou são negativas. Nesse caso, o gradiente vai a zero o que impossibilita o *backpropagation*, impedindo a rede de aprender. Esse problema é chamado de *dying ReLU*.

2.5.3.4 *Leaky* ReLU

Com a finalidade de elucidar o *dying* ReLU foi elaborada a função *Leaky ReLU* que possui uma inclinação na parte dos valores negativos de entrada (MISSINGLINK,

2020), deixando de possuir gradiente zero, conforme a Figura 13.

Figura 13 – Forma gráfica da função *Leaky* ReLU



Fonte: Autoria própria

A Equação da função *Leaky* ReLU é dada por 7.

$$f(x) = max(0.1x,x) \tag{7}$$

A utilização da Leaky ReLU é necessária para retornar valores consistentes na previsões para valores com entrada negativa.

2.5.3.5 *Exponential Linear Unit* - ELU

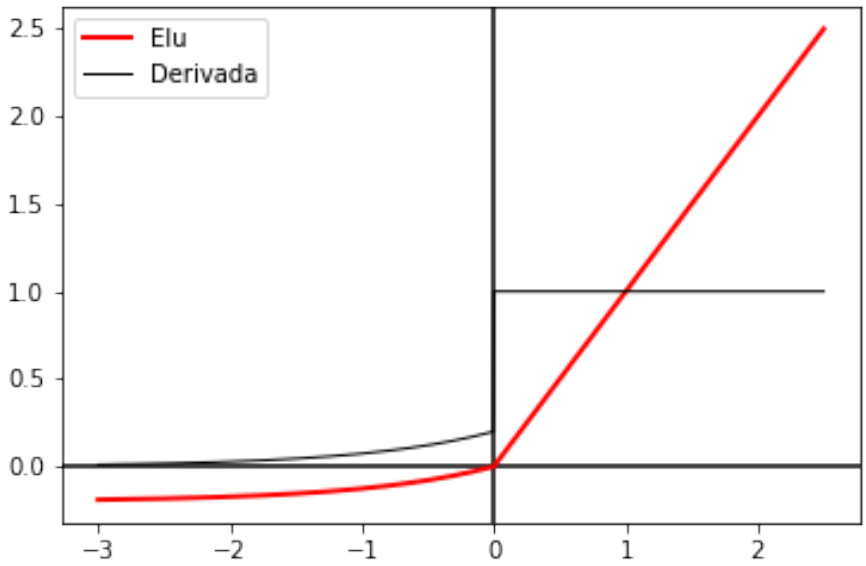
A *Exponential Linear Unit* (ELU) é uma função de ativação muito similar à *Leaky* ReLU, porém com uma inclinação mais suave quando os valores de entrada se aproximam de zero ou são negativos (MISSINGLINK, 2020). A Figura 14 mostra esta inclinação.

Ao contrario de outras funções de ativação, a ELU não possui problemas de *vanishing gradient*. Outro ponto positivo é a não ocorrência do *dying* ReLU, ou morte dos neurônios. Com isso, a ELU torna-se uma alternativa interessante.

A função ELU é dada pela equação 8.

$$f(x) = \begin{Bmatrix} x, & se & x \geq 0 \\ e^x - 1, & se & x < 0 \end{Bmatrix} \tag{8}$$

Figura 14 – Forma gráfica da função ELU



Fonte: Autoria própria

2.5.3.6 *Softmax*

A função de ativação *Softmax*, assim como a sigmoide, tem a capacidade de mapear os valores de saída entre 0 e 1, como na Figura 15. O seu diferencial consiste em que o somatório das saídas é sempre 1.

Esta função mais complexa tem z que representa um vetor de entradas e j um indicie inteiro positivo.

A função *Softmax* é dada pela equação 9:

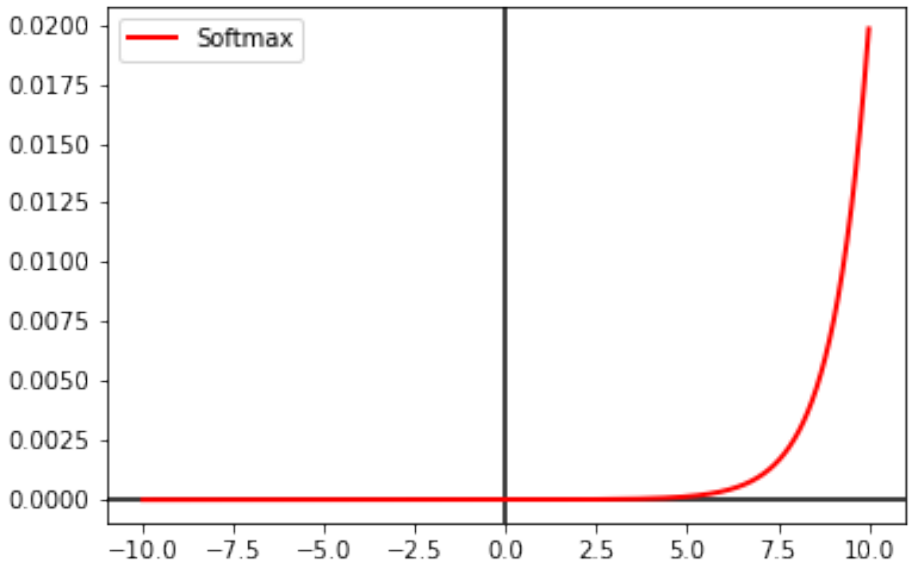
$$f(x) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{9}$$

Com isso, a *Softmax* pode ser definida com uma função de distribuição de probabilidade (MISSINGLINK, 2020). Por possuir essas duas características, uma das aplicações desta função é em camadas mais externas, ou de saída, da rede neural artificial, lidando muito bem com problemas de classificação.

2.6 DIVISÃO DAS AMOSTRAS

Para que as RNAs com aprendizado supervisionado tenham uma boa performance, o conjunto de amostras totais devem ser divididas em três subconjuntos. O primeiro é de treinamento, que é usado para calcular o gradiente e atualizar os pesos sinápticos da rede (SILVA DANILO HERNAME SPATTI, 2010). Este é o subconjunto o

Figura 15 – Forma gráfica da função *Softmax*



Fonte: Autoria própria

qual possui a maioria dos dados totais.

O segundo subconjunto é o de validação. O erro no conjunto de validação é monitorado durante o processo de treinamento, o qual normalmente diminui durante a fase inicial do treinamento. No entanto, quando a rede começa a sobre-treinar os dados, o erro deste conjunto tende a aumentar. Então, é neste ponto que os pesos da rede são salvos e utilizados para a aplicação da rede.

O restante das amostras pertencem ao subconjunto de teste, os quais não fazem parte dos ajustes de qualquer parâmetro da rede. Este é usado para comparar diferentes modelos, indicando qual obteve melhor desempenho, capacidade de generalização (GERON, 2019). Também é útil para apresentar graficamente o erro do conjunto de teste durante o processo de treinamento. Se o erro no conjunto de teste atingir um mínimo em um número de iteração significativamente diferente do erro do conjunto de validação, isso pode indicar uma má divisão do conjunto de dados.

2.7 APLICAÇÕES

Existem grandes diferenças entre as aplicações das RNAs. De maneira geral, quando a finalidade é prever valores quantitativos, o método de previsão utilizado é regressão. Em contrapartida, para a obtenção de valores qualitativos utiliza-se o método de classificação.

O modelo de previsão baseado em regressão consiste em algoritmos mate-

máticos de aproximação, que visam o mapeamento através de uma função de entrada $f(x)$ alimentada pelas variáveis x, para chegar em um resultado y contínuo (BROWNLEE, 2019).

O valor das variáveis contínuas são do domínio real, como inteiros e pontos flutuantes. Esse tipo de previsão é geralmente utilizado para se obter quantidades, tamanhos ou valores.

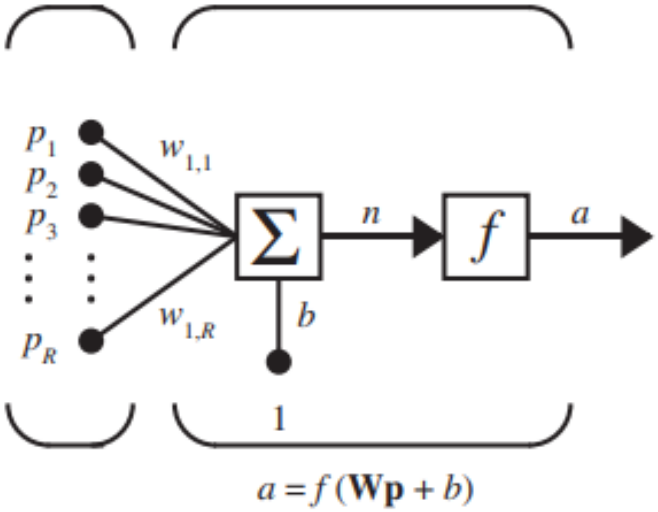
De forma similar à regressão, a classificação busca fazer a aproximação através de uma função $f(x)$ a partir dos valores de entrada x para uma saída y. Porém, nesse modelo de previsão, a saída é discreta. Usualmente, as saídas de uma rede para classificação são denominadas de categorias ou rótulos. A função de mapeamento então determina a qual categoria a entrada pertence, dada ao treinamento recebido pela rede (BROWNLEE, 2019).

2.8 *PERCEPTRON* - CONCEITOS BÁSICOS

O *perceptron* definiu a base para modelos de redes neurais em 1958. O algoritmo desenvolvido por Frank Rosenblatt foi compilado e publicado em *"Principles of Neuro-dynamics: Perceptrons and the Theory of Brain Mechanisms"* (ROSENBLATT, 1958). Por mais que hoje sejam utilizados outros modelos de neurônios, o *Perceptron* permite entender, de forma clara, como funciona uma rede neural de forma matemática.

O *perceptron* baseia-se, de certa forma, na lógica *threshold* apresentada na seção 2.2. No exemplo mostrado na Figura 16, as entradas p_1, p_2 ... p_R são multiplicados pelos pesos $w_{1,1}$, $w_{1,2}$ ... $w_{1,R}$.

Figura 16 – Exemplo de rede *Perceptron*



Fonte: Hagan, Demuth e Jesús (2002)

Após a multiplicação, os valores são somados e vão alimentar a função de

ativação da célula de saída. A rede *perceptron* é considerada um classificador binário linear, usada para segregar duas categorias de dados, por exemplo. O seu modelo matemático pode ser descrito pela equação 10.

$$output = \begin{cases} 1, & se \quad \sum_j p_j w_j \geq \textit{threshold} \\ 0, & se \quad \sum_j p_j w_j < \textit{threshold} \end{cases} \tag{10}$$

Para um exemplo teste a fim de mostrar seu funcionamento imagina-se uma rede que deve decidir se uma pessoa deve ou não sair de casa para comprar batata frita em uma lanchonete. Pode-se definir três entradas sendo:

- p_1: Se pessoa está ou não com fome.
- p_2: O tempo está ensolarado ou não.
- p_3: A pessoa está acompanhada ou não.

Para a finalidade desse exemplo, adota-se que as entradas afirmativas valem 1 e as negativas valem 0. Com esses parâmetros definidos, parte-se para a definição dos pesos. Conforme a importância do parâmetro, o peso pode ser maior ou menor. Depois disso, os valores estão prontos para serem aplicados na função de ativação.

Para a primeira simulação, arbitra-se para w_1, w_2 e w_3 peso 1, como limiar adota-se o valor 2. Nesse cenário, fica claro que se ao menos duas das entradas forem afirmativas a saída será 1.

Supõem-se agora que, ao observar um grande número de pessoas, a entrada p_1 seja determinante, logo o peso w_1 deve ser ajustado para 2. Nesse caso, basta a pessoa estar com fome para que a saída seja 1. Variando os pesos ou o limiar é possível obter diferentes modelos para as tomadas de decisão da rede.

2.8.1 Viés ou *Bias* do Perceptron

O viés pode ser descrito como uma medida de como a rede Perceptron irá disparar a saída positiva. Para uma rede com um alto valor de viés, é extremamente fácil emitir 1. Porém, se o viés é muito negativo, dificilmente a rede emitirá uma saída 1.

Com a finalidade de simplificar a equação 10 pode-se escrever as entradas p_R e os pesos w_n como vetores, conforme mostra a 11.

$$output = \begin{cases} 1, & se \quad \mathbf{X} \times \mathbf{W} \geq \textit{threshold} \\ 0, & se \quad \mathbf{X} \times \mathbf{W} < \textit{threshold} \end{cases} \tag{11}$$

Agora sendo:

$$-threshold = b \tag{12}$$

Substituindo 12 em 11 se obtém:

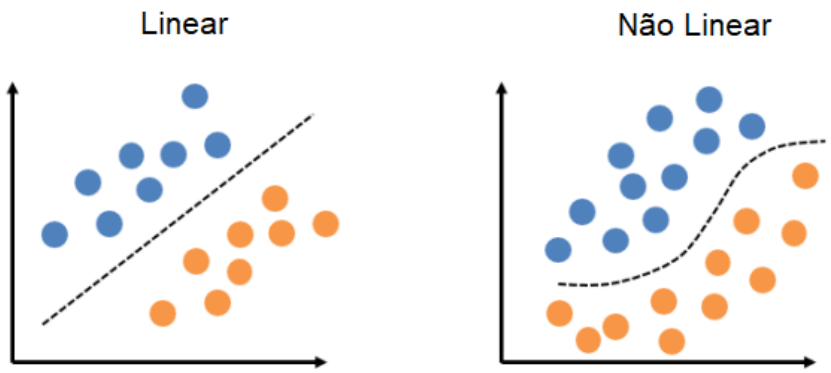
$$output = \begin{cases} 1, & se \quad \mathbf{X} \times \mathbf{W} \geq -b \\ 0, & se \quad \mathbf{X} \times \mathbf{W} < -b \end{cases} \tag{13}$$

Com isso, quanto maior o valor de *bias* b, mais difícil será para obter uma saída em nível 1.

Para o treinamento de um Perceptron, deve-se fazer com que o modelo encontre os valores ideais de pesos e bias. Para isso, é apresentado ao modelo dados de entrada rotulados com a saída esperada. A cada iteração, os pesos e *bias* são ajustados. Com o modelo ajustado, apresenta-se novos dados de entrada e este preverá a saída.

Com uma única rede *Perceptron* é possível resolver somente funções linearmente separáveis (ACADEMY, 2020). Porém, para aplicações reais, raramente os dados serão linearmente separáveis, como na Figura 17. Isto esclarece o porquê do *Perceptron* ter perdido espaço, se comparado com arquiteturas mais complexas.

Figura 17 – Exemplo de separação linear e não linear



Fonte: Academy (2020)

2.9 ARQUITETURA DE REDES NEURAIS

Como abordado na seção 2.4, as RNAs mais complexas podem conter uma ou mais camadas ocultas. Contudo, além do número de camadas é importante definir como os neurônios dessas camadas interagem. Alguns modelos apenas processam as informações e as passam para frente, as chamadas redes *Feedforward*. Outros

possuem *loops* de *feedback* que utilizam a saída do neurônio para compor parte da entrada de um neurônio anterior, as conhecidas rede recorrentes.

Ainda é possível classificar as RNAs em uma terceira classe, as redes conectadas simetricamente. Essas, por sua vez, são baseadas nas redes recorrentes, porém as conexões entre as unidades são simétricas (possuem o mesmo peso em ambas as direções).

2.9.1 Redes *Feedforward*

Uma rede *feedforward* padrão é composta apenas por uma camada de entrada, uma camada oculta e uma camada de saída. Outra característica é que, as células de processamento (neurônios) estão apenas conectadas às células das camadas anteriores e posteriores e não conectadas à mesma camada.

Os neurônios na camada oculta de uma rede *feedforward* têm a função de processar os dados, calculando a soma ponderada dos valores de entrada. Então, aplica-se essa soma em uma função de ativação $f(X)$. Por exemplo, na função sigmoide, fará com que o valor de saída seja expresso entre 0 e 1, por meio de um mapeamento não-linear.

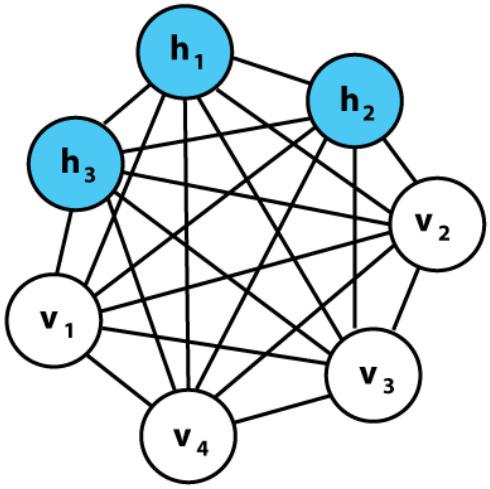
Esse processo é repetido em cada camada oculta para redes profundas, em que as redes neurais calculam uma série de transformações que alteram as semelhanças entre os casos e passam para a próxima camada. Essas atividades dos neurônios em cada camada se torna uma função não-linear das atividades da camada anterior (Schmidt; Kraaijveld; Duin, 1992).

2.9.2 Redes Recorrentes

As redes neurais recorrentes são estudas desde 1980, e foram desenvolvidas para aprender padrões sequenciais ou variantes no tempo. Uma rede recorrente é aquela em que existe uma realimentação (*loop* fechado) (LAURENE, 1994). Um exemplo de rede neural recorrente é a máquina de Boltzmann, desenvolvida por Ackley, Hinton e Sejnowski (1985). Com essa arquitetura, os neurônios são completamente conectados, como mostra a Figura 18.

As redes neurais recorrentes são aplicadas em uma vasta categoria de problemas, desde aprender sequências de caracteres ou o comportamento de sistemas dinâmicos com uma sequência de eventos. Para cada aplicação, as redes recorrentes podem variar de totalmente ou parcialmente conectadas. As redes primeiras não possuem uma camada de entrada característica, e sim cada neurônio alimentando todos os demais, as vezes incluindo ele mesmo (LAURENE, 1994).

Figura 18 – Representação gráfica da máquina de Boltzmann



Fonte: Domain (2020)

Com isso, este tipo de arquitetura pode ter uma dinâmica complexa, o que pode torná-la difícil de treinar. Entretanto, são biologicamente realistas (ACADEMY, 2020).

2.10 *PERCEPTRON* DE MÚLTIPLAS CAMADAS

A arquiteturas mais conhecidas de RNAs é o *Perceptron* de Múltiplas Camadas (MLP). Esta rede *feedforward* nada mais é que a generalização do *perceptron* de ROSENBLATT. Uma das grandes vantagens da MLP é possuir a característica de aproximar qualquer função não-linear contínua em um espaço compacto e com precisão arbitrária (SIQUEIRA, 2013).

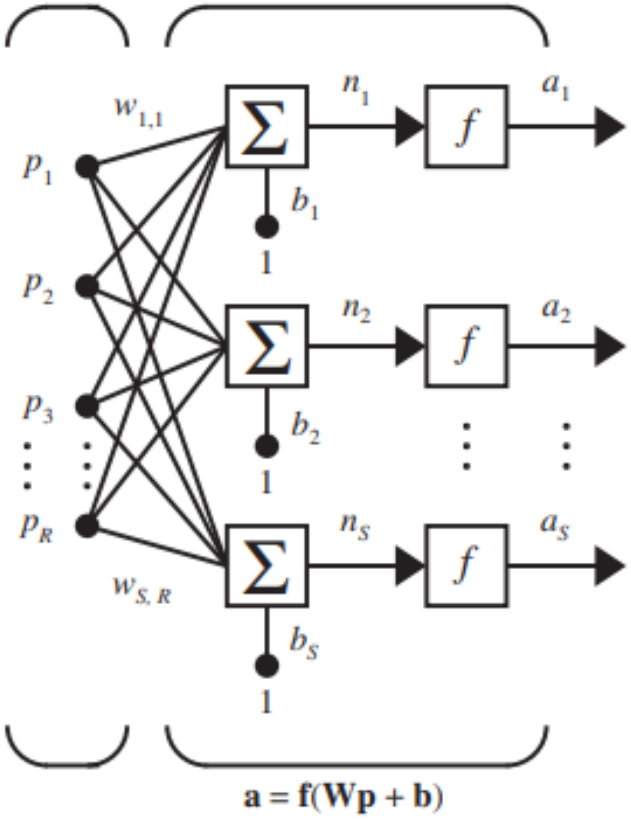
O processo de treinamento de uma MLP se dá através dos ajustes dos pesos ou ponderações sinápticas. Com isso, a rede busca adequar os valores a cada iteração, a fim de encontrar o ponto ótimo para realizar o mapeamento requerido. Esse processo pode ser realizado através de métodos de otimização não-linear irrestrita (HAYKIN, 2003).

O método utilizado para otimização desta rede consiste em minimização de uma função custo, utilizando gradiente descendente, por exemplo. A partir disso, pode-se empregar o algoritmo de retro-propagação de erro (*backpropagation*), para realizar os cálculos sistemáticos do gradiente da função custo. Em seguida, o valor dos pesos é ajustado, com vistas à redução do erro de treinamento.

A rede MLP clássica é composta por neurônios artificias (GERON, 2019). É dividida em uma camada de entrada, uma ou mais camadas ocultas, e uma camada de saída. A primeira é responsável por repassar os sinais de entrada para as cama-

das ocultas sem nenhuma manipulação, na maioria das vezes. Nas camadas ocultas, ocorre a transformação dos dados de entrada para outro espaço, em que os dados de entrada passam pelo somatório de cada neurônio e suas respectivas funções de ativação. Nesta etapa, ocorre a inserção de não-linearidade do modelo. A Figura 19 mostra uma das possibilidades de arranjo, com uma camada oculta (HAYKIN, 2003).

Figura 19 – Representação de uma camada oculta da rede MLP



Fonte: Hagan, Demuth e Jesús (2002)

Nesta imagem são mostrados os processos pelos os neurônios processam as informações, de maneira genérica, na camada intermediária. As R entradas denominadas P são multiplicadas pelo peso W. essa resposta é acrescida do viés b (o qual também possui ponderação), gerando as n conexões de saídas para a função de ativação, que resultam nas a saídas de cada neurônio da camada oculta.

Como já abordado na seção 2.2, as conexões entre neurônios artificiais é ponderada através dos pesos que simulam os impulsos sinápticos reguláveis do neurônio biológico, ou a capacidade de aprendizado. Fica explicitado o comportamento *feed-forward* desta rede, já que as a saídas apenas se conectam à próxima camada e não para à mesma camada, ou camadas anteriores, de modo que não há nenhuma realimentação. As MLP podem conter diversas camadas ocultas, porém para aplicações práticas é comum a utilização de apenas uma (HAYKIN, 2008).

2.10.1 Treinamento da rede MLP

O treinamento para redes do tipo MLP é supervisionado, consistindo em duas partes principais:

- Fase *forward*: os valores de entrada são processados conforme a Figura 19, gerando uma saída a partir de pesos fixos. Essa saída é comparada com o sinal desejado (saída esperada) e um erro é gerado;
- Fase *backward*: a próxima etapa consiste em ajustar os valores dos pesos de acordo com a metodologia de correção de erro estabelecida após o cálculo do gradiente da função custo, calculado a partir do erro do treinamento.

A função custo, mais utilizada para este tipo de treinamento é o erro quadrático médio, em inglês *Mean Squared Error* (MSE). Em problemas de classificação, quando usada a função *softmax* na camada de saída, é comum usar a função custo entropia cruzada, em inglês *categorial cross-entropy*, já que ambas garantem uma distribuição de probabilidade bem comportada, o que facilita o entender, se o modelo está tendo uma probabilidade alta de acerto ou não.

2.10.1.1 Gradiente descendente

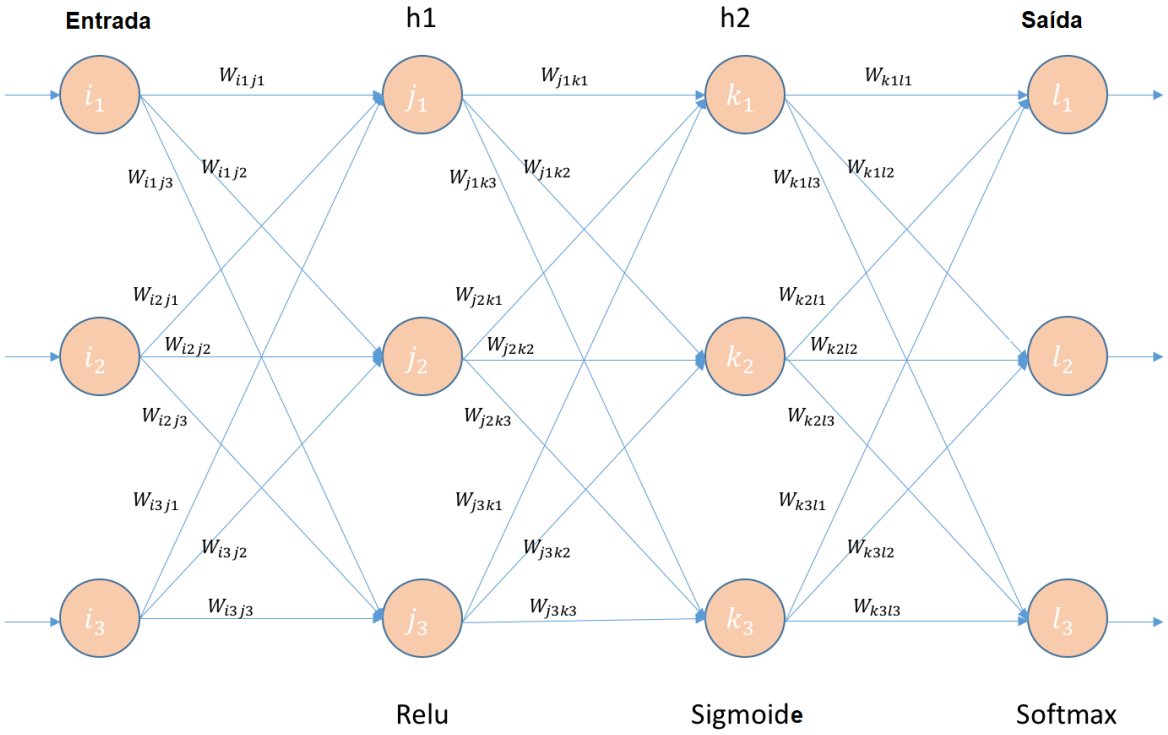
O objetivo de todos os algoritmos supervisionados de AM é estimar uma função de destino, que mapeia os dados de entrada para as variáveis de saída. No caso das redes neurais, o objetivo é mais especificamente encontrar os valores dos parâmetros (pesos) de uma função que fornecem a estimativa ótima da função custo, a qual irá ser ajustada com base nos dados de treinamento.

Há diversas metodologias para alcançar este objetivo. Neste trabalho, foi utilizado o gradiente descendente. Este utiliza-se do cálculo (especificamente da Álgebra Linear) para tentar encontrar o mínimo global da função dada pela rede neural. Com a finalidade de explicar o gradiente descendente, será usada a Figura 20 como exemplo.

A arquitetura da rede apresenta 2 camadas ocultas com 3 neurônios cada. A 1º e a 2º camadas ocultas têm ReLU e Sigmoide, respectivamente, como funções de ativação. A camada de saída tem a função de ativação *softmax* e, como função custo a entropia cruzada.

Após os pesos e bias serem inicializados aleatoriamente, é realizada a fase *forward*, em que os valores de entrada são processados, sendo gerada uma saída. Com a saída calculada pela rede e a saída desejada é feito o cálculo do erro. Com o erro calculado é possível começar o *backpropagation*.

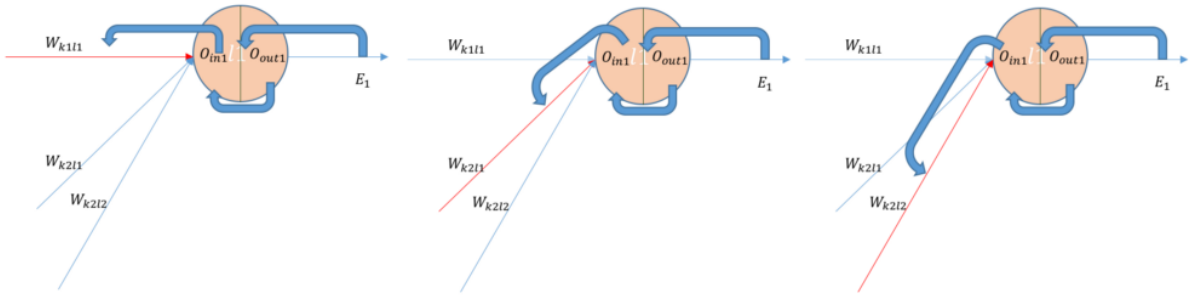
Figura 20 – Exemplificando a rede neural MLP



Fonte: Adaptado de Jay (2020)

Para corrigir os pesos referentes à terceira camada, é calculada a derivada (utilizando a regra da cadeia) e obtém-se três parcelas: a primeira referente à derivada do erro em relação à saída do neurônio, a segunda referente à derivada da função de ativação (transformação da somatória dos dados que entram no neurônio para o dado que sai do neurônio), e a terceira parte, à derivada do somatório das entradas em relação à cada peso. A Figura 21 ilustra esta etapa.

Figura 21 – *Backpropagation* referente a terceira camada

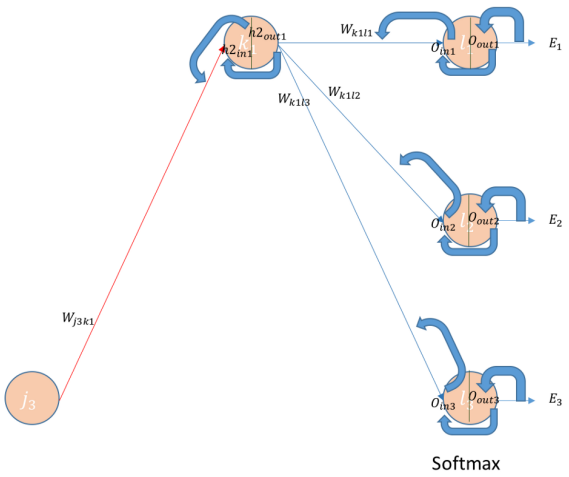


Fonte: Jay (2020)

No *backpropagation* referente à segunda camada, ao fazer a derivada da função de custo em relação aos pesos da segunda camada, também obtêm-se três parcelas de derivadas: a primeira é a soma dos erros calculados na etapa anterior (daí a origem do nome retro propagação do erro), a segunda em relação à função de ati-

vação desta camada, e por último, a derivada do somatório de entrada em relação à cada peso. A Figura 22 ilustra o *backpropagation* referente à segunda camada.

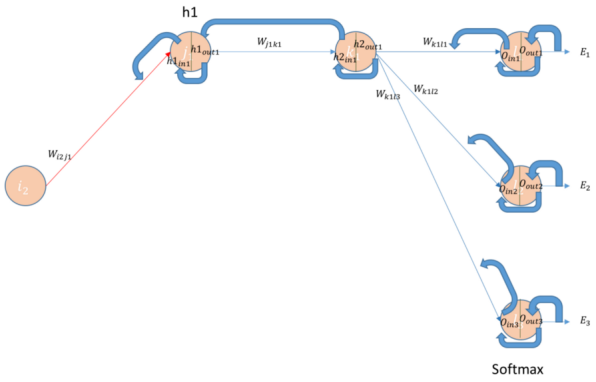
Figura 22 – *Backpropagation* referente a segunda camada



Fonte: Jay (2020)

O *backpropagation* referente à primeira camada, é semelhante ao da segunda. O que os difere é a primeira parcela em que a derivada é a soma do erro total em relação à saída do neurônio da primeira camada e não da segunda. A Figura 23 ilustra esta etapa.

Figura 23 – *Backpropagation* referente a primeira camada



Fonte: Jay (2020)

Estes cálculos são feitos para cada amostra, entrentanto é possível realizar o *backpropagation* sobre mais de uma amostra, a abordagem em batelada, neste caso, é calculado o erro médio. A quantidade de amostras utilizadas é denominada de mini-lote (*batch size*). Quanto maior o tamanho do mini-lote, maior será a variação do peso. Por um lado, ao aumentar este tamanho, a tendência é aumentar a velocidade de convergência, porém é possível pular ou dificultar encontrar os melhores mínimos da função.

Após finalizado o *backpropagation*, fase *backward*, repete-se o ciclo até que se satisfaça alguma condição de parada, como época máxima ou valor satisfatório do erro.

2.10.2 Validação Cruzada

Quando trata-se de treinamento de RNAs, o ajuste deve ser feito de modo que não prejudique demasiadamente a flexibilidade da rede, pois a grande utilidade de RNAs é a capacidade de generalização. Quando ocorre o sobre-treinamento (*over-fitting*), a rede apresentará um mal desempenho quando aplicada a dados que não pertençam ao conjunto original de treinamento. Isto pode ser causado por um excesso no número de neurônios ou por um treinamento mais longo do que o necessário. Uma técnica muito utilizada para contornar a segunda causa é a validação cruzada (SIQUEIRA, 2013).

Esta metodologia determina que deve-se dividir os dados de forma que um conjunto de validação, com amostras que não participem do ajuste dos pesos seja criado. A cada época (passagem da rede pelo conjunto de treinamento), após o ajuste dos parâmetros, os dados de validação são inseridos na rede e a resposta de saída irá gerar, de forma independente do treinamento, uma medida de erro. Com isso, o ponto de mínimo para o erro de validação é determinado como o de melhor generalização estimada, dessa forma, os valores dos pesos neste ponto são fixados como os melhores da rede.

Para isso, é preciso garantir que os dois conjuntos de dados sejam suficientemente representativos em relação com o mapeamento que se pretende aproximar. É comum determinar primeiro a quantidade de amostras referente ao subconjunto de treinamento e, posteriormente utilizar parte deste subconjunto para a validação.

2.11 *FRAMEWORKS* E PROGRAMAS JÁ EXISTENTES PARA MODELAGEM DE RNAS

Já existem alguns trabalhos que visam criar *softwares* para facilitar a coleta e manipulação dos dados, como o *Orange Data Mining*, bibliotecas de algoritmos para importação de RNAs completas para serem utilizadas em códigos fonte e, *frameworks* que possibilitam obter RNAs através de alguns passos, como é o caso do JustNN, Encog e Neuroph.

No caso do JustNN e Neuroph, é possível criar uma arquitetura de rede neural configurando o número de neurônios, camadas ocultas e saídas. Contudo, ambos sofrem com uma interface com design ultrapassada e contra intuitivo, o que deixa

aparente que estes programas foram desenvolvidos voltados para um publico que já possui experiência com redes neurais.

No caso do Encog, é necessário que o usuário possua alguma expertise em programação na linguagem *python* para implementar as bibliotecas disponibilizadas. Este fato se repete com outros diversos *frameworks* disponibilizados na internet, como: *PyTorch*, *OpenNN*, FANN, entre outras. Outro ponto negativo das três ferramentas avaliadas é a dificuldade de alimentar o *software* com a base de dados, sendo a maioria compatível apenas com CSV.

Com isso, é possível observar que existe uma lacuna a ser preenchida por *frameworks* que sejam desenvolvidos voltados à usuários menos experientes, que possua uma interface moderna agradável, permitam a extração dos dados de bases de diferentes tipos e apresente os resultados de maneira gráfica. No próximo capítulo, será apresentada uma proposta neste sentido.

3 DESENVOLVIMENTO

Neste capítulo será abordado como utilizar o *framework* implementado para utilização de redes neurais artificiais. Discurtir-se-á a interface gráfica, as bibliotecas utilizadas para o desenvolvimento e os componentes, ilustrando cada etapa desenvolvida.

O idioma escolhido para o *framework* foi o inglês, buscando abranger um número maior de usuários. Com relação à linguagem de programação, foi utilizado o *Python*. O *framework* possuí as seguintes características:

1. Rede Neural Artificial disponível:
    - Perceptron de múltiplas camadas;
2. Abordagens de Pré-processamentos:
    - Normalização;
    - Dessazonalização;
    - Codificação distribuída;
3. Formato das bases de dados suportadas:
    - *Microsoft Excel .xlsx*;
    - *Matlab .mat*;
    - Arquivo de Texto .txt;
    - Valores Separados por Vírgulas .csv.
4. Funções de Ativação:
    - Degrau;
    - Linear;
    - Sigmoid;
    - Tangente hiperbólica;
    - ReLU;
    - Leaky ReLU;
    - ELU;
    - Softmax;
5. Função de custo:

- *Erro quadrático médio*;
- *Crossentropia categórica*

Com todas as características acima citadas o framework desenvolvido atende as necessidades que as ferramentas desse segmento demandam, como (SUPTITZ IVAN LUIS E FROZZA, 2015) apontam: "Uma interface gráfica com aquisição de dados, possibilitando entrada de dados brutos, fornecendo opções de pré-processamento e também uma opção de execução do modelo. Com isso, a interface do *framework* é dividia em:

- Uma barra lateral: transição entre telas;
- Uma barra de ferramentas: importação e organização dos dados;
- Tela *AI*: escolha das entradas, saídas e especificações do modelo a ser treinado;
- Tela *Tables*: exibição de dados em tabela;
- Tela *Graphics*: visualização gráfica do dados.

Para o desenvolvimento das telas, as principais bibliotecas utilizadas foram:

- Pandas: importação, exportação e organização dos dados;
- Scikit-learn: pré-processamento e bases de dados do *framework*;
- TensorFlow: redes neurais artificiais;
- PyQt5: interface gráfica (*Graphical User Interface* (GUI)).

3.1 BARRA LATERAL

A barra lateral tem como função a mudança entre uma tela e outra. Como há três telas, há portanto três botões. O botão superior direciona para a tela da inteligência artificial (*AI*); o do meio para a tela das tabelas (*Tables*); e o inferior para a tela dos gráficos (*Graphics*). A Figura 24 mostra a barra lateral.

Figura 24 – Barra lateral

Fonte: Autoria própria.

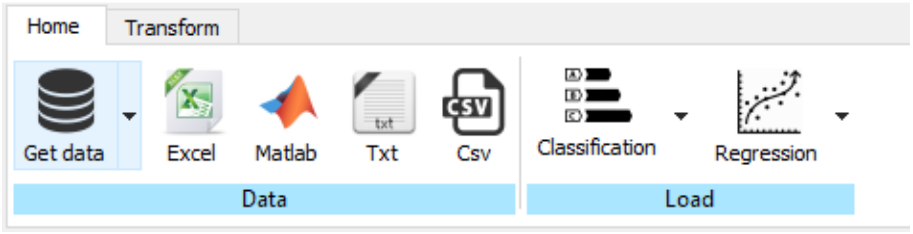
3.2 BARRA DE FERRAMENTAS

A barra de ferramentas fica no topo da interface e é dividida em duas abas: *Home* e *Transform*.

3.2.1 Aba *Home*

A primeira etapa a ser realizada no uso da ferramenta é a importação dos dados, a qual será feita na aba *Home*. É possível importar uma base de dados do computador do usuário, como também carregar dados provenientes do *framework*. A Figura 25 ilustra a aba *Home*.

Figura 25 – Barra de ferramentas - aba *Home*



Fonte: Autoria própria.

3.2.1.1 Importar base de dados do computador

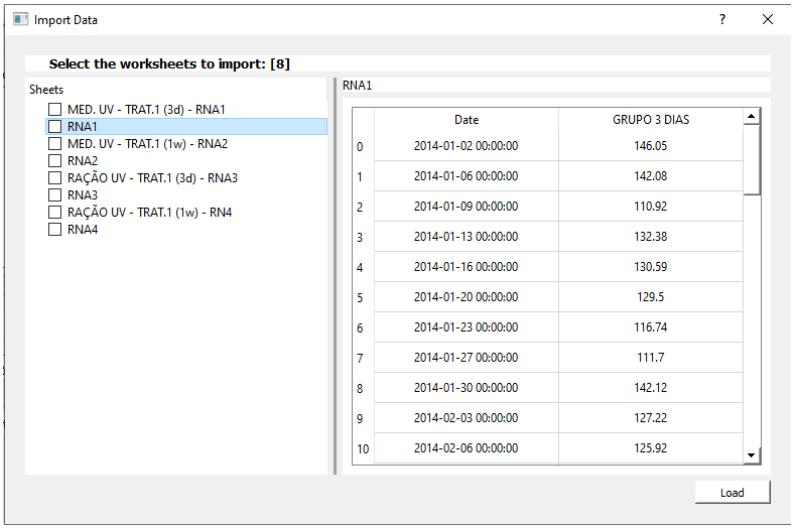
Para importar arquivos do computador do usuário há quatro opções disponíveis:

- *Microsoft Excel .xlsx .xls*
- *Arquivo de dados Matlab .mat*
- Arquivo de Texto .txt
- Arquivo CSV .csv

Para a seleção de arquivo *Microsoft Excel*, há a opção com extensão .xlsx e a opção .xls. Se um arquivo *Excel* ou *Matlab* (.mat) for selecionado, caso este contenha mais de uma tabela, irá aparece uma tela para selecionar quais das tabelas serão importadas, como na Figura 26. De modo a auxiliar, é possível visualizar parte dos dados ao selecionar uma das tabelas. Esta etapa é importante para arquivos grandes, e permitem filtrar os dados importados, consumindo menos memória do computador.

Para os arquivos de texto (.txt), há a possibilidade de selecionar o separador de dados utilizado no arquivo, já que podem estar separados por espaço, vírgula ou algum outro caracter.

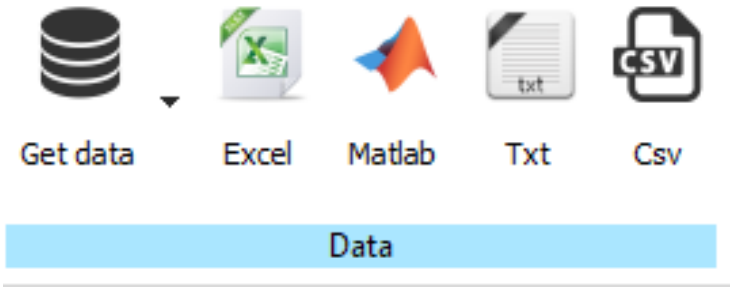
Figura 26 – Selecionador de tabelas



Fonte: Autoria própria.

Para a importação dos dados, foi utilizado a biblioteca Pandas, que possui, além dessas, outras possíveis extensões para ler os arquivos e criar um *DataFrame* (tabela com cabeçalho, índices e os dados em si). A Figura 27 mostra as opções para importação do computador do usuário.

Figura 27 – Opções para importar base de dados

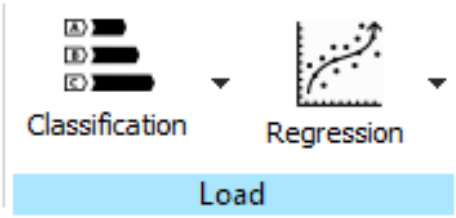


Fonte: Autoria própria.

3.2.1.2 Importar base de dados do *framework*

É possível importar algumas bases de dados do próprio *framework*, contendo opções tanto para classificação, como também para regressão. Para problemas de classificação, foram setadas três opções preliminares: *Iris*, *Wine* E *Breast Cancer*. Para regressão, disponibiliza-se as opções: *Boston Houses* e *Diabetes*. As base de dados estão disponíveis e são importadas diretamente da biblioteca *scikit-learn*. A Figura 28 mostra as opções para carregar dados do próprio *framework*.

Figura 28 – Base de dados disponíveis no *framework*



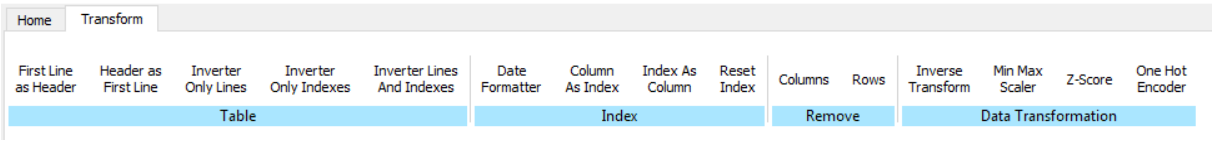
Fonte: Autoria própria.

3.2.2 Aba *Transform*

Usualmente, as bases de dados não estão apropriadas para serem usadas diretamente nos modelos de *machine learning*. Então, acaba sendo necessário organiza-los e trata-los previamente. A aba *Transform*, Figura 29, possui estas finalidades e está dividida em quatro partes:

1. *Table*: definir cabeçalho e ordem das linhas;
2. *Index*: ajuste dos índices;
3. *Remove*: remover linhas ou colunas;
4. *Data Transformation*: pré-processamento

Figura 29 – Barra de ferramentas - aba *Transform*



Fonte: Autoria própria.

Como as funções da aba *Transform* estão relacionados diretamente com as base de dados, esta só está disponível enquanto estiver na tela *Tables*.

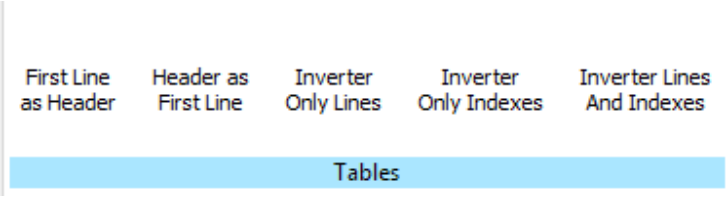
3.2.2.1 *Table*

É possível que ao importar os dados, eles venham diferente do padrão desejado. Por exemplo, ao se fazer a importação, a primeira linha é considerada como cabeçalho porém, é possível que os dados não possuam um cabeçalho. Por isso, foi criada a opção de definir o cabeçalho como primeira linha. Caso seja necessário, há a opção de fazer o inverso: colocar a primeira linha de dados como cabeçalho.

Outra necessidade possivel é inverter a ordem apenas dos índices, apenas dos dados ou ambos. Por exemplo, em problemas de regressão é necessário que os

dados estejam do mais antigo (primeira linha da tabela) para o mais novo (última linha da tabela). Então, é possível que seja necessário reverter a ordem dos dados. A Figura 30 ilustra as opções da parte *Table*.

Figura 30 – Barra de ferramentas, opções *Tables*



Fonte: Autoria própria.

3.2.2.2 *Index*

Como opções de configurações dos índices da tabela, há quatro possibilidades: selecionar uma coluna, formatá-la como data e defini-la coluna como índice; apenas definir uma coluna como índice; definir os índices como uma coluna, resetando os índices (deixando como números de zero até a quantidade de amostras); ou apenas resetar os índices. A Figura 31 mostra estas opções.

Figura 31 – Barra de ferramentas, opções *Index*

Fonte: Autoria própria.

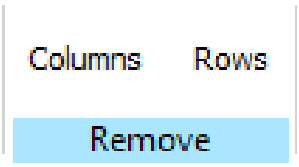
3.2.2.3 *Remove*

Outra funcionalidade implementada é a possibilidade de remover as linhas ou colunas da tabela selecionando um *range*. A opção *Rows* é utilizada para excluir linhas e a opção *Columns* para excluir as colunas. A Figura 32 mostra as duas opções.

3.2.2.4 *Data Transformation*

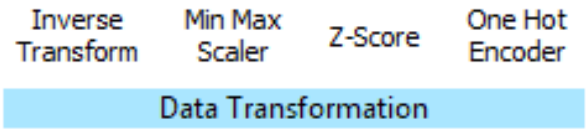
A aba *Data transformation* é onde ocorre o pré-processamento da base de dados. Como opções, há a normalização (*Min Max Scaler*), a dessazonalização (*Z-Score*), e a codificação distribuída (*One Hot Encoder*). A Figura 33 ilustra as opções do *Data Transformation*.

Figura 32 – Barra de ferramentas, opções *Remove*



Fonte: Autoria própria.

Figura 33 – Barra de ferramentas, opções *Data Transformation*



Fonte: Autoria própria.

Para executar o pré-processamento basta selecionar ao menos uma coluna e clicar na barra de ferramentas sobre o método desejado. Internamente, serão armazenados: o método e a(s) coluna(s) selecionada(s). Estes serão utilizados posteriormente para realizar tal etapa.

O botão *Inverse Transform* serve para posteriormente realizar o pós-processamento. Deve-se selecionar ao menos uma coluna, e então ao clicar sobre o botão referente a esta etapa. Aparecerá uma tela para escolher qual o pré-processamento inicialmente utilizado e, ao lado, escolher a coluna que gerou este processo.

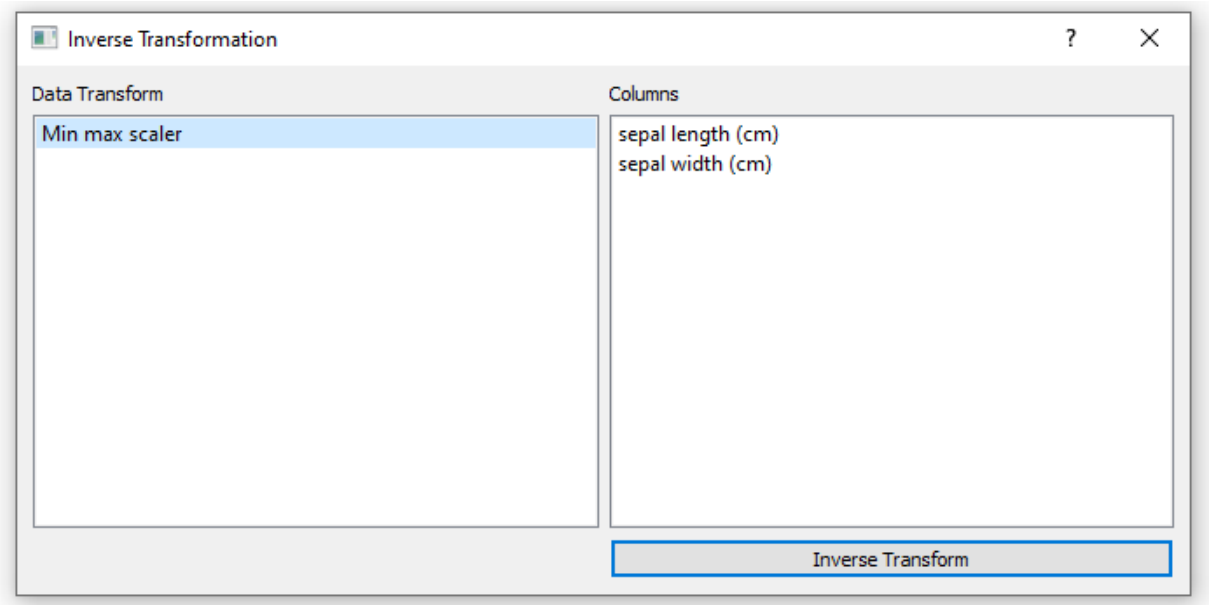
Eis um exemplo: utilizando a base de dados IRIS foi aplicado o pré-processamento *Min Max Scaler* nas colunas *"sepal width (cm)"* e *"sepal length (cm)"*. Ao clicar no botão *Inverse Transform*, irá aparecer duas listas. A primeira mostra os pré-processamentos utilizados, neste caso, o *Min Max Scaler*. Na segunda lista, aparecerá as opções *"sepal width (cm)"* e *"sepal length (cm)"*, que correspondem aos nomes das colunas que inicialmente foram selecionadas. A Figura 34 ilustra este exemplo.

3.3 *TABLES*

Após os dados serem extraídos, é possível fazer a sua visualização. Esta é a finalidade da tela *Tables*, Figura 35. Para cada base de dados é gerada uma nova tabela em uma nova aba. Ao clicar com o botão direito sobre a aba, é possível renomeá-la (*Rename*), duplicá-la (*Duplicate*) ou exportar os dados em formato Excel (.xlsx) (*Save*). A Figura 36 ilustra essas opções.

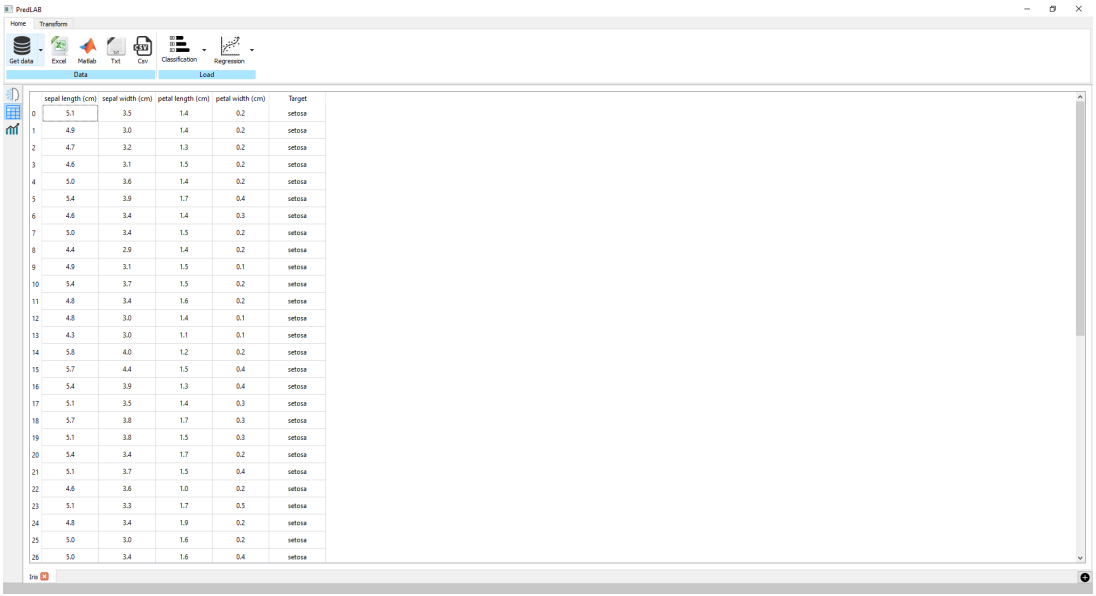
A biblioteca Pandas faz toda a lógica da tabela (*DataFrame*), enquanto a biblioteca PyQt5 é responsável por passar os dados para a interface gráfica.

Figura 34 – *Inverse Transform*



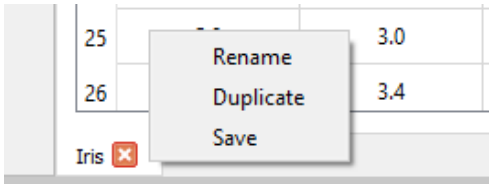
Fonte: Autoria própria.

Figura 35 – Tela *Tables*



Fonte: Autoria própria.

Figura 36 – Ferramentas da tabela

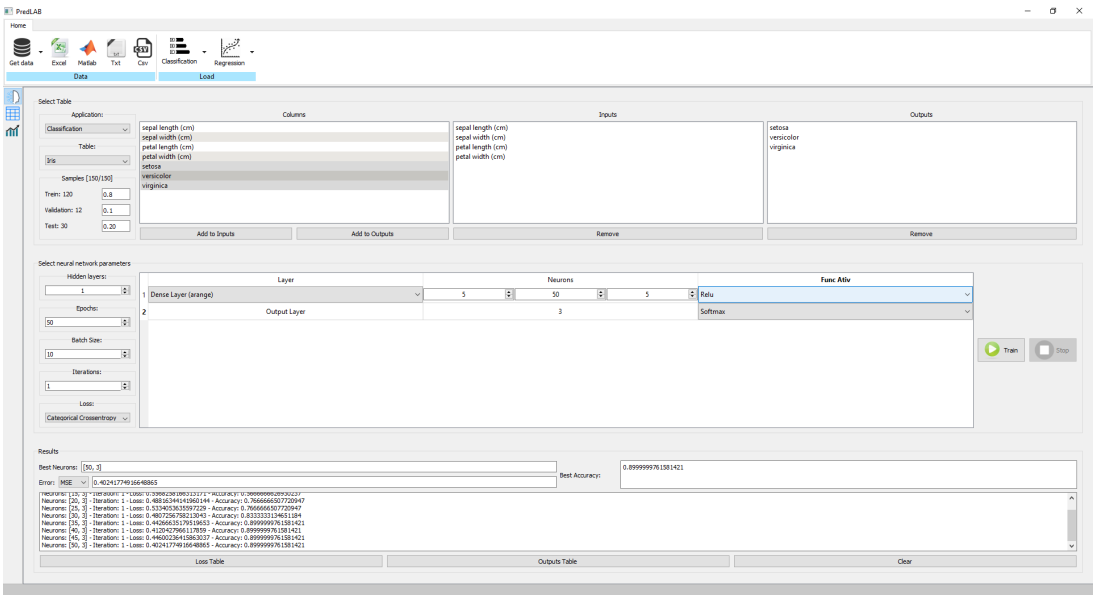


Fonte: Autoria própria.

3.4 *AI*

A parte mais importante da interface ocorre na tela referente à inteligência artificial (*Artificial Inteligence*). Esta tela é composta por três partes: seleção das entradas e das saídas, seleção dos parâmetros do modelo e, por último, os resultados gerados. A Figura 37 ilustra esta tela.

Figura 37 – Tela *AI*



Fonte: Autoria própria.

3.4.1 Seleção de *Inputs* e *Outputs*

Primeiramente, é necessário definir para que tipo de problema a rede neural será treinada, se classificação ou regressão (previsão). Posteriormente, o usuário irá definir quais as entradas e saídas e a base de dados escolhida.

Para definir a base de dados, há um opção com o nome de todas as abas criadas na tela *Tables*. Após a seleção de uma delas, o campo ao lado é atualizado com o nome das colunas da tabela selecionada. Ao escolher ao menos um item da lista (colunas da tabela), é possível defini-lo como entrada ou saída do modelo.

Caso seja selecionado como um problema de classificação, basta definir quais as entrada e quais as saídas. Já para o caso de regressão, é necessário determinar qual a quantidade de passos à frente a ser feita a previsão e quais os atrasos para as variáveis de entrada, sendo possível utilizar o método *Wrapper* ou escolher manualmente.

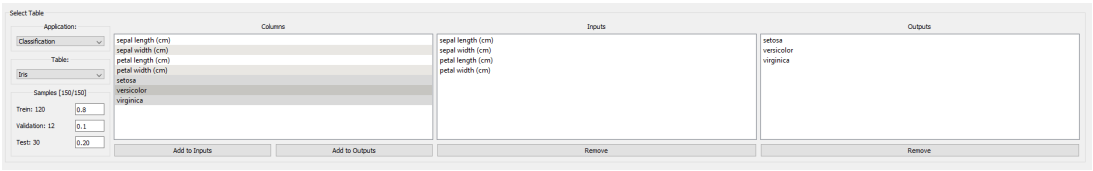
É nesta etapa que será feita a divisão da base de dados, na qual se define a quantidade de amostras a serem utilizadas para o treinamento, validação e teste. Para

isso, existe um campo para cada caso, no qual pode ser escrito um valor inteiro, que corresponde à quantidade de amostras destinadas a cada uma das divisões. Ainda, há a opção de setar uma porcentagem (valor entre 0 e 1), em que já será feito o cálculo automático do número de amostras em cada grupo. Para auxiliar, este valor é mostrado no rotulo (*label*), referente a divisão das amostras. Por exemplo: em uma base de dados com 100 amostras, se for escrito 0.6 no campo de treinamento, a *label* será atualizada para ”Trein: [60]”.

Para o conjunto de validação, a porcentagem é calculada em relação ao conjunto de treinamento, enquanto a porcentagem do treinamento e do teste é em relação ao conjunto total. Ao clicar sobre a *label* referente ao treinamento, o campo que define a quantidade de amostras será preenchido automaticamente, calculando a diferença entre a quantidade total e a aquela destinada para o teste. O mesmo pode ser feito ao clicar sobre a *label* referente ao teste, este sendo calculado como a diferença entre amostras total e amostras do treinamento.

A Figura 38 ilustra esta parte.

Figura 38 – Seleção de *Inputs* e *Outputs*



Fonte: Autoria própria.

3.4.2 Seleção dos parâmetros do modelo

Após definida a divisão de amostras do treinamento, validação e teste e quais variáveis de entrada e saída do modelo, a próxima etapa é determinar os parâmetros da rede neural para, então poder treina-la. Para isso, é necessário escolher a quantidade de camadas, quantidade de neurônios por camada, as funções de ativação, número de épocas, tamanho do lote, quantidade de iterações e qual a métrica de erro.

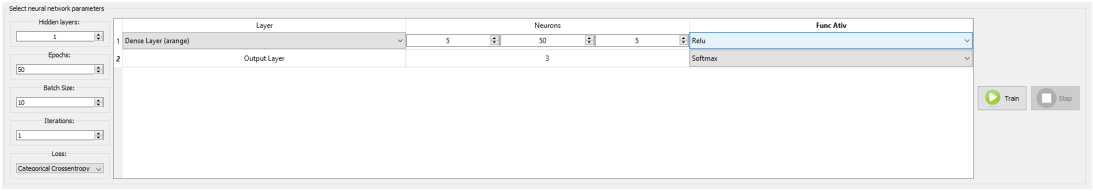
Ao mudar a quantidade de camadas escondidas (*hidden layers*), caso seja um valor maior que o atual, as novas camadas irão replicar a última camada intermediária. Caso seja um valor menor, irá excluir a partir da última camada intermediária.

Para cada camada adicionada, deve-se escolher o tipo da camada, a quantidade de neurônios e a função de ativação. Como o modelo implementado é uma rede neural MLP, a camada é do tipo densa, de modo que todos os neurônios da camada anterior são conectados com a atual. Porém, há possibilidade de escolher entre uma quantidade única de neurônios ou uma sequência de valores definindo o início, o fim e o passo dessa sequência. A quantidade de neurônios da camada de saída é atuali-

zado automaticamente, de acordo com as saídas definidas na seleção de *Outputs*.

A Figura 39 ilustra esta funcionalidade.

Figura 39 – Seleção dos parâmetros do modelo

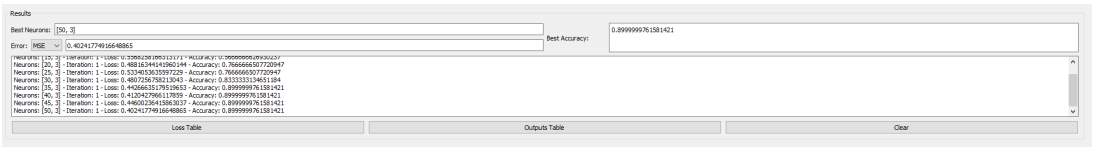


Fonte: Autoria própria.

3.4.3 Saídas/resultados gerados

Enquanto o modelo é treinado, a cada variação de parâmetro é acrescentada uma nova linha em um campo de texto, servindo para auxiliar o usuário com a indicação de, por quais parâmetros a rede já treinou/está sendo treinada. Há outros campos em que fica armazenado apenas os valores referentes aos melhores parâmetros. A Figura 40 ilustra estes campos.

Figura 40 – Resultados



Fonte: Autoria própria.

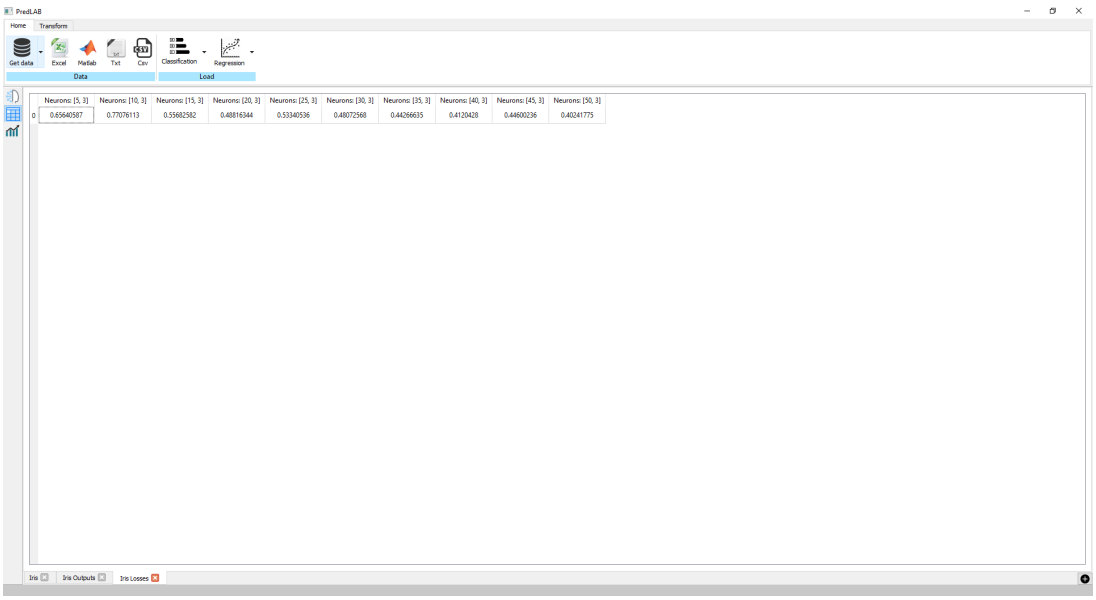
Após o treinamento ser finalizado, é possível gerar duas tabelas. A primeira é referente aos erros gerados por cada variação e, a segunda é refente às saídas calculadas. A Figura 41 ilustra a tabela de erro, enquanto a Figura 42 ilustra a tabela das saídas calculadas.

3.5 GRÁFICOS

A terceira tela da interface tem como finalidade a visualização gráfica dos dados (*data visualize*). A visualização é de extrema importância, pois através desta ferramenta é possível entender os dados com mais facilidade. Por exemplo, ao olhar os dados plotados num gráfico, é possível entende-los e interpreta-los, analisando as tendências e/ou exceções presentes.

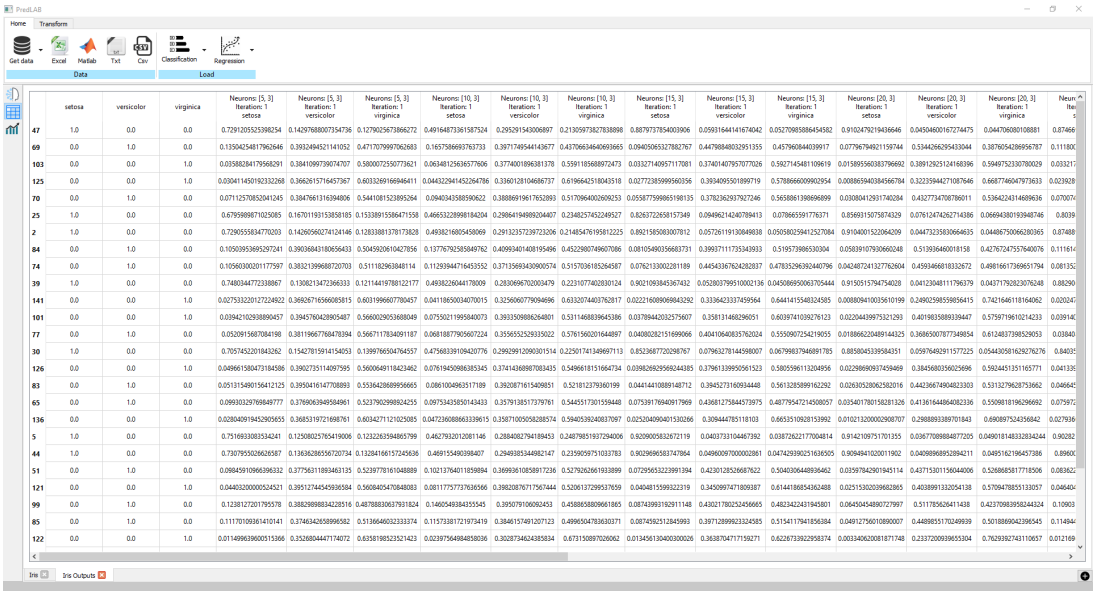
A tela dos gráficos está dividida em três partes: criação de uma figura (com um ou mais gráficos), escolha de quais dados serão plotados, e o(s) gráfico(s) gerado(s). A Figura 43 mostra esta tela.

Figura 41 – Tabela de *Loss*



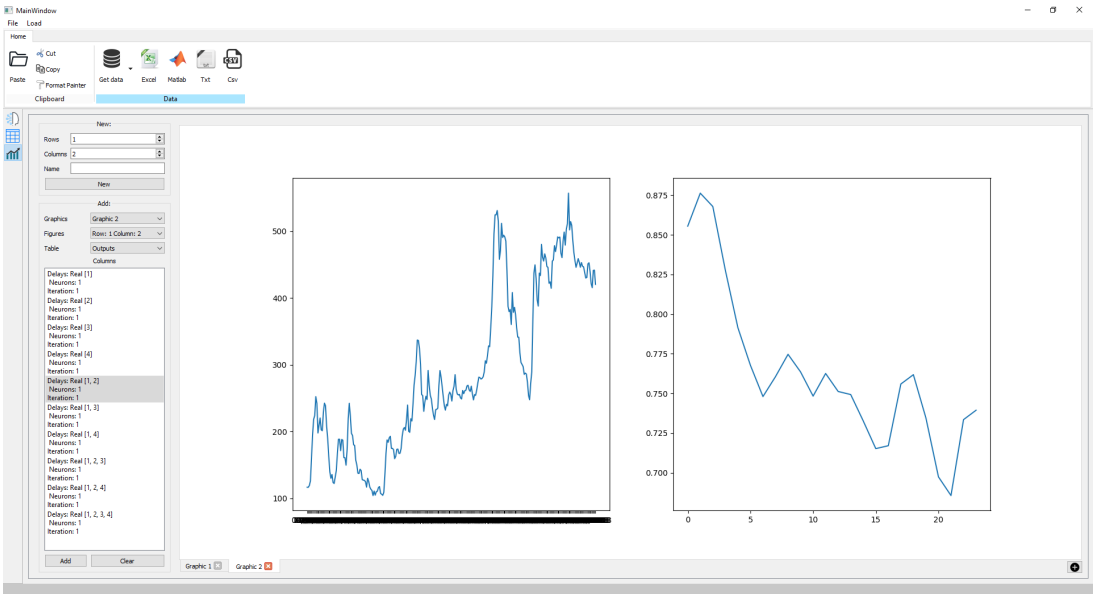
Fonte: Autoria própria.

Figura 42 – Tabela de *Ouputs*



Fonte: Autoria própria.

Figura 43 – Tela *Graphics*

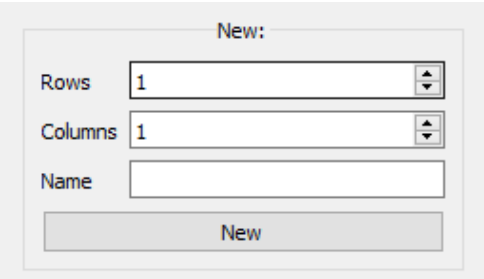


Fonte: Autoria própria.

3.5.1 Gerar novo gráfico

Antes de plotar os dados em um gráfico é necessário criar uma figura, em que se pode escolher quantos eixos ela apresentará. Por exemplo, é possível dentro de uma figura haver dois gráficos um ao lado do outro. Para isso, basta selecionar uma linha e duas colunas. Também é possível, mas não obrigatório, inserir um nome para o gráfico. A Figura 44 mostra o menu de criação de gráficos e a Figura 43 é a representação de uma figura com uma linha e duas colunas.

Figura 44 – Geração de um novo gráfico



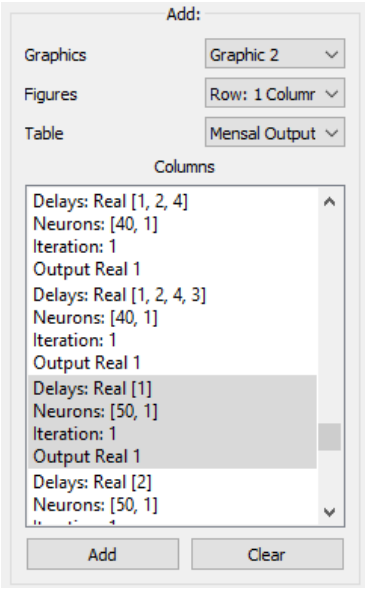
Fonte: Autoria própria.

Cada figura pode ter no máximo três linhas e três colunas, ou seja, um total de 9 gráficos. A cada figura gerada é criada uma nova aba com o nome fornecido pelo campo *Name*. Se este campo não for preenchido será atribuído o nome ”*Graphic*”.

3.5.2 Plotar os dados em um gráfico

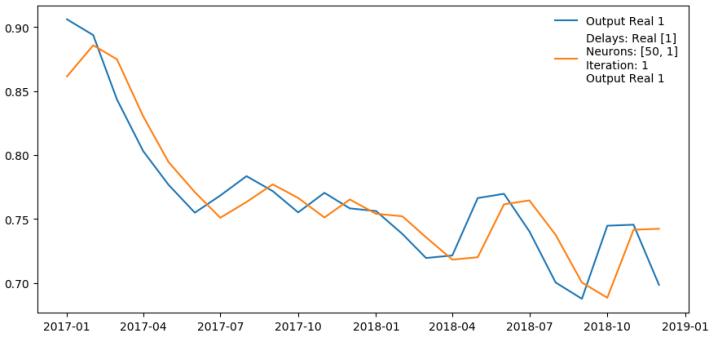
Para plotar os dados em um dos gráficos gerados é necessário escolher primeiramente qual a figura, um dos gráficos (eixos) criados e por fim os dados, selecionando a tabela em que estes estão. As escolher a tabela, será atualizado uma lista com o nome das colunas. Ao selecionar ao menos uma coluna e clicar no botão *Add* será plotada a(s) coluna(s) selecionada(s) no eixo selecionado. Também há a opção de deletar o gráfico ao clicar no botão *Clear*. É possível observar essas opções na Figura 45 e os gráficos gerados na Figura 46.

Figura 45 – Plotar dados



Fonte: Autoria própria.

Figura 46 – Gráficos gerados



Fonte: Autoria própria.

4 RESULTADOS E DISCUSSÃO

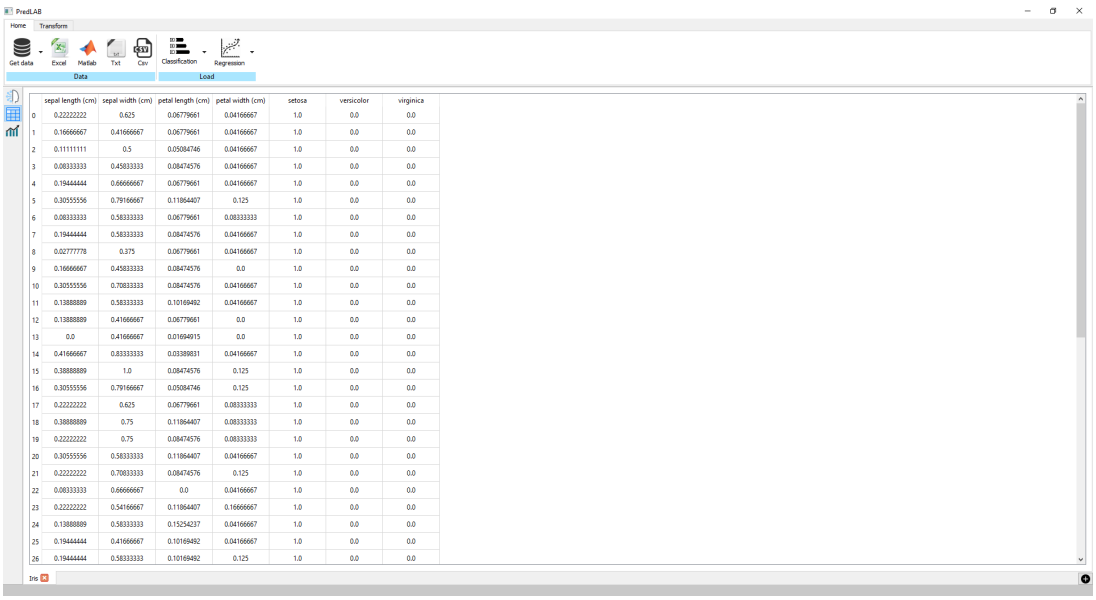
Neste capítulo será apresentada uma aplicação prática do *framework* utilizando duas bases de dados. A primeira para aplicação de um problema de classificação utilizando a base de dados iris e a segunda referente ao preço do café arábica, destinado a exemplificar um problema de regressão.

4.1 CLASSIFICAÇÃO

A base de dados iris é composta por quatro variáveis de entrada: *sepal length (cm)*, *sepal width (cm)*, *petal length (cm)*, *petal width (cm)*. Como alvo (*target*), há três categorias: setosa, versicolor e virgínica.

Como os dados da iris já vêm disponibilizada junto com o *framework*, basta clicar na aba *Home* da barra de ferramentas e selecionar ”Iris” na parte de classificação. Os dados importados vão aparecer na tela *Tables*. A Figura 47 ilustra este *dataset*.

Figura 47 – Base de dados iris



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	setosa	versicolor	virginica
0	0.22222222	0.625	0.06779661	0.04166667	1.0	0.0	0.0
1	0.16666667	0.41666667	0.06779661	0.04166667	1.0	0.0	0.0
2	0.11111111	0.5	0.05084746	0.04166667	1.0	0.0	0.0
3	0.08333333	0.45833333	0.08474576	0.04166667	1.0	0.0	0.0
4	0.19444444	0.66666667	0.06779661	0.04166667	1.0	0.0	0.0
5	0.30555556	0.79166667	0.11864407	0.125	1.0	0.0	0.0
6	0.08333333	0.58333333	0.06779661	0.08333333	1.0	0.0	0.0
7	0.19444444	0.58333333	0.08474576	0.04166667	1.0	0.0	0.0
8	0.02777778	0.375	0.06779661	0.04166667	1.0	0.0	0.0
9	0.16666667	0.45833333	0.08474576	0.0	1.0	0.0	0.0
10	0.30555556	0.70833333	0.08474576	0.04166667	1.0	0.0	0.0
11	0.13888889	0.58333333	0.10169492	0.04166667	1.0	0.0	0.0
12	0.13888889	0.41666667	0.06779661	0.0	1.0	0.0	0.0
13	0.0	0.41666667	0.01694915	0.0	1.0	0.0	0.0
14	0.41666667	0.83333333	0.03389831	0.04166667	1.0	0.0	0.0
15	0.38888889	1.0	0.08474576	0.125	1.0	0.0	0.0
16	0.30555556	0.79166667	0.05084746	0.125	1.0	0.0	0.0
17	0.22222222	0.625	0.06779661	0.08333333	1.0	0.0	0.0
18	0.38888889	0.75	0.11864407	0.08333333	1.0	0.0	0.0
19	0.22222222	0.75	0.08474576	0.08333333	1.0	0.0	0.0
20	0.30555556	0.58333333	0.11864407	0.04166667	1.0	0.0	0.0
21	0.22222222	0.70833333	0.08474576	0.125	1.0	0.0	0.0
22	0.08333333	0.66666667	0.0	0.04166667	1.0	0.0	0.0
23	0.22222222	0.54166667	0.11864407	0.16666667	1.0	0.0	0.0
24	0.13888889	0.58333333	0.15254237	0.04166667	1.0	0.0	0.0
25	0.19444444	0.41666667	0.10169492	0.04166667	1.0	0.0	0.0
26	0.19444444	0.58333333	0.10169492	0.125	1.0	0.0	0.0

Fonte: Autoria própria.

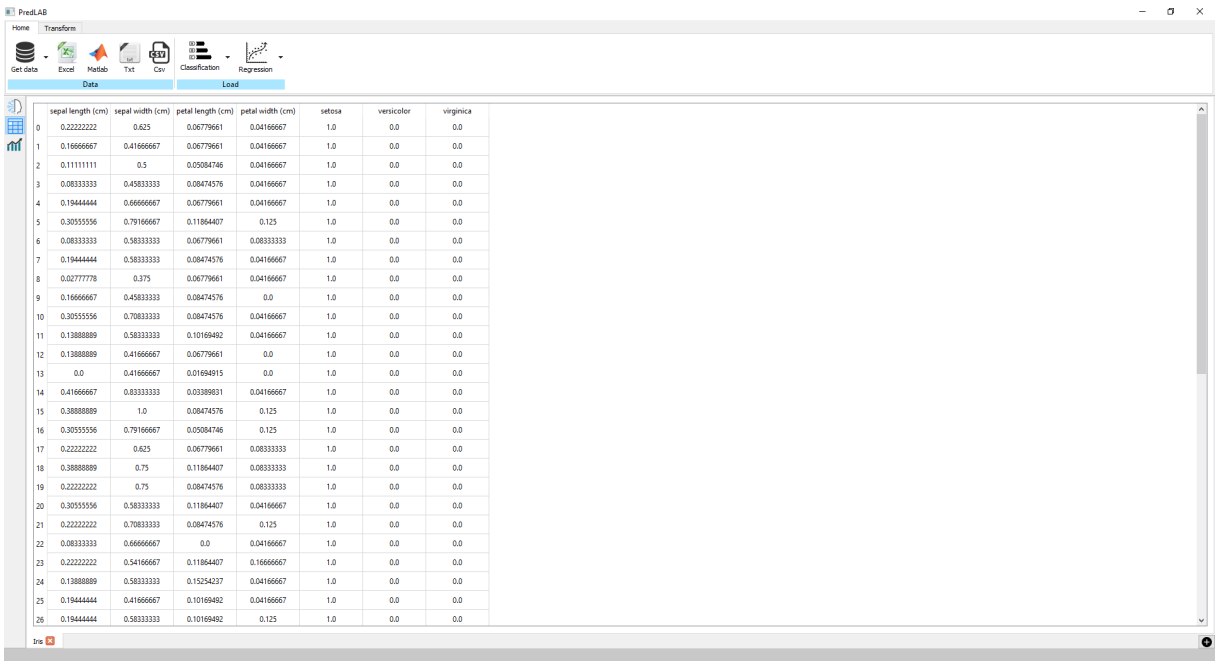
4.1.1 Pré-processamento - base de dados iris

Observando o *dataset* é possível notar que a variável (coluna) *Target* está categorizada (em rótulos OU *labels*). Neste caso é necessário realizar o *One Hot Encoder*, transformando os rótulos em um valor bínario, transformando todos os valores zero, exceto o índice correspondente. Para executar esta etapa, na aba da tela *Tables*,

que contém os dados, seleciona-se a coluna *Target* e então na aba *Transform* da barra de ferramentas seleciona-se a opção *One Hot Encoder*.

Já para as demais colunas, as variáveis de entrada, foi efetuada a normalização (*Min Max Scaler*) dos dados. Este processo é semelhante ao anterior: seleciona as colunas e desta vez escolhe-se a opção *Min Max Scaler*. A Figura 48 mostra os dados com o pré-processamento já realizado.

Figura 48 – Base de dados iris com o pré-processamento



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	setosa	versicolor	virginica
0	0.22222222	0.625	0.06779661	0.04166667	1.0	0.0	0.0
1	0.16666667	0.41666667	0.06779661	0.04166667	1.0	0.0	0.0
2	0.11111111	0.5	0.05084746	0.04166667	1.0	0.0	0.0
3	0.08333333	0.45833333	0.08474576	0.04166667	1.0	0.0	0.0
4	0.19444444	0.66666667	0.06779661	0.04166667	1.0	0.0	0.0
5	0.30555556	0.79166667	0.11864407	0.125	1.0	0.0	0.0
6	0.08333333	0.58333333	0.06779661	0.08333333	1.0	0.0	0.0
7	0.19444444	0.58333333	0.08474576	0.04166667	1.0	0.0	0.0
8	0.02777778	0.375	0.06779661	0.04166667	1.0	0.0	0.0
9	0.16666667	0.45833333	0.08474576	0.0	1.0	0.0	0.0
10	0.30555556	0.70833333	0.08474576	0.04166667	1.0	0.0	0.0
11	0.13888889	0.58333333	0.10169492	0.04166667	1.0	0.0	0.0
12	0.13888889	0.41666667	0.06779661	0.0	1.0	0.0	0.0
13	0.0	0.41666667	0.01694915	0.0	1.0	0.0	0.0
14	0.4166667	0.83333333	0.03389831	0.04166667	1.0	0.0	0.0
15	0.38888889	1.0	0.08474576	0.125	1.0	0.0	0.0
16	0.30555556	0.79166667	0.05084746	0.125	1.0	0.0	0.0
17	0.22222222	0.625	0.06779661	0.08333333	1.0	0.0	0.0
18	0.38888889	0.75	0.11864407	0.08333333	1.0	0.0	0.0
19	0.22222222	0.75	0.08474576	0.08333333	1.0	0.0	0.0
20	0.30555556	0.58333333	0.11864407	0.04166667	1.0	0.0	0.0
21	0.22222222	0.70833333	0.08474576	0.125	1.0	0.0	0.0
22	0.08333333	0.66666667	0.0	0.04166667	1.0	0.0	0.0
23	0.22222222	0.54166667	0.11864407	0.16666667	1.0	0.0	0.0
24	0.13888889	0.58333333	0.15254237	0.04166667	1.0	0.0	0.0
25	0.19444444	0.41666667	0.10169492	0.04166667	1.0	0.0	0.0
26	0.19444444	0.58333333	0.10169492	0.125	1.0	0.0	0.0

Fonte: Autoria própria.

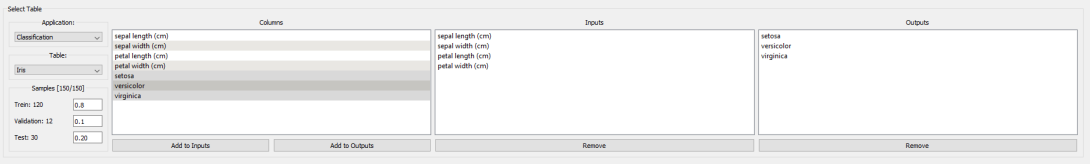
Com os dados prontos para serem colocados no modelo, as próximas configurações ocorrem na tela *AI*.

4.1.2 Seleção dos parâmetros do modelo - base de dados iris

Como a base de dados iris é utilizada para resolver um problema de classificação, é necessário conferir se esta é a opção selecionada no campo *Application*. Logo abaixo há o campo para escolher a tabela que contêm os dados pré-processados. Após selecionada a tabela, a lista de colunas ao lado será atualizada, sendo a próxima etapa definir quais variáveis serão entrada (botão *Add to Inputs*) e quais serão saídas (*Add to Outputs*). A Figura 49 mostra esta tela com os respectivos parâmetros já ajustados.

A seguir escolhe-se os parâmetros da rede neural artificial. Neste exemplo será usada uma camada oculta com a função de ativação ReLU. Para quantidade de neurônios foi utilizando um *arange* (vetor) de possibilidades, começando em 5 até 50 e um passo de 5 em 5 (5, 10, 15, 20, ..., 40, 45, 50). Por se tratar de um problema de

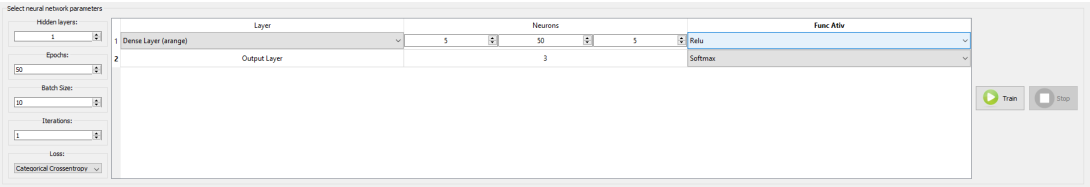
Figura 49 – Seleção das variáveis de entrada e saída - base da dados iris



Fonte: Autoria própria.

classificação, foi utilizado também a *loss Categorical Crossentropy* e a função de ativação *Softmax* para camada de saída. Os outros parâmetros ficaram definidos como: 30 para *epochs*, 10 para *batch size*, e 1 para *Iterations*. A Figura 50 ilustra estes parâmetros ajustados.

Figura 50 – Seleção dos parâmetros do modelo - base da dados iris

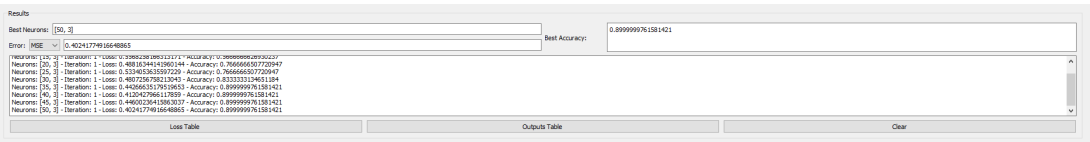


Fonte: Autoria própria.

4.1.3 Resultados gerados - base de dados iris

Após a exibição da mensagem de sinalização que o modelo terminou de treinar, é possível visualizar todos os erros e acurácias relativas a cada variação de parâmetro realizado no treino, como na Figura 51. Também é possível observar os valores que originaram as melhores saídas. Como ilustrado na Figura 51, a rede neural com melhor acurácia foi a que apresentou 50 neurônios na camada intermediária e 3 na camada de saída, um erro de 0,4024 e acurácia de 89%.

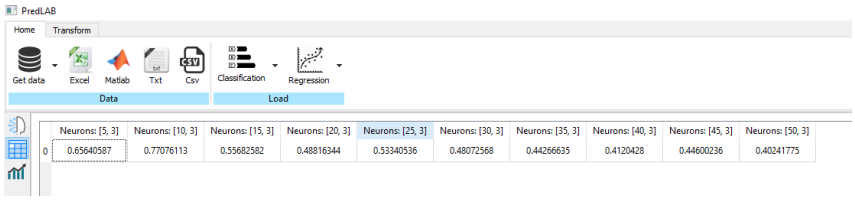
Figura 51 – Resultados gerados - base da dados iris



Fonte: Autoria própria.

Caso seja desejado visualizar todos os erros em formato de tabela há o botão *Loss Table*, Figura 52. Também há a opção de visualizar as saídas previstas *clickando* no botão *Outputs Table*, como mostra a Figura 53.

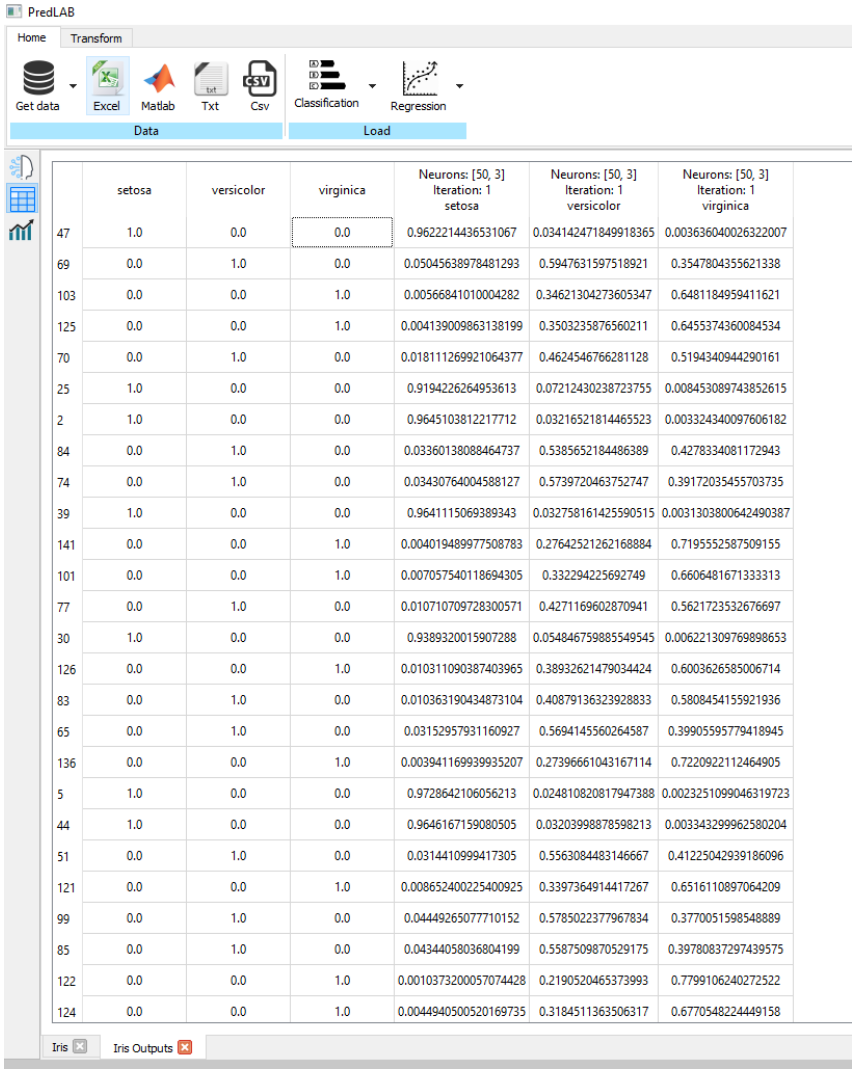
Figura 52 – Tabela de erros - base da dados IRIS



	Neurons: [5, 3]	Neurons: [10, 3]	Neurons: [15, 3]	Neurons: [20, 3]	Neurons: [25, 3]	Neurons: [30, 3]	Neurons: [35, 3]	Neurons: [40, 3]	Neurons: [45, 3]	Neurons: [50, 3]
0	0.65640587	0.77076113	0.55682582	0.48816344	0.53340536	0.48072568	0.44266635	0.4120428	0.44600236	0.40241775

Fonte: Autoria própria.

Figura 53 – Tabela de saídas - base da dados iris



	setosa	versicolor	virginica	Neurons: [50, 3] Iteration: 1 setosa	Neurons: [50, 3] Iteration: 1 versicolor	Neurons: [50, 3] Iteration: 1 virginica
47	1.0	0.0	0.0	0.9622214436531067	0.034142471849918365	0.003636040026322007
69	0.0	1.0	0.0	0.05045638978481293	0.5947631597518921	0.3547804355621338
103	0.0	0.0	1.0	0.00566841010004282	0.34621304273605347	0.6481184959411621
125	0.0	0.0	1.0	0.004139009863138199	0.3503235876560211	0.6455374360084534
70	0.0	1.0	0.0	0.018111269921064377	0.4624546766281128	0.5194340944290161
25	1.0	0.0	0.0	0.9194226264953613	0.07212430238723755	0.008453089743852615
2	1.0	0.0	0.0	0.9645103812217712	0.03216521814465523	0.003324340097606182
84	0.0	1.0	0.0	0.03360138088464737	0.5385652184486389	0.4278334081172943
74	0.0	1.0	0.0	0.03430764004588127	0.5739720463752747	0.39172035455703735
39	1.0	0.0	0.0	0.9641115069389343	0.032758161425590515	0.0031303800642490387
141	0.0	0.0	1.0	0.0040194899775087 83	0.27642521262168884	0.7195552587509155
101	0.0	0.0	1.0	0.0070575401186943 05	0.332294225692749	0.6606481671333313
77	0.0	1.0	0.0	0.0107107097283005 71	0.4271169602870941	0.5621723532676697
30	1.0	0.0	0.0	0.9389320015907288	0.054846759885549545	0.006221309769898653
126	0.0	0.0	1.0	0.0103110903874039 65	0.38932621479034424	0.6003626585006714
83	0.0	1.0	0.0	0.0103631904348731 04	0.40879136323928833	0.5808454155921936
65	0.0	1.0	0.0	0.031529579311609 27	0.5694145560264587	0.39905595779418945
136	0.0	0.0	1.0	0.0039411699399352 07	0.27396661043167114	0.7220922112464905
5	1.0	0.0	0.0	0.9728642106056213	0.024810820817947388	0.0023251099046319723
44	1.0	0.0	0.0	0.9646167159080505	0.03203998878598213	0.003343299962580204
51	0.0	1.0	0.0	0.031441099941730 5	0.5563084483146667	0.41225042939186096
121	0.0	0.0	1.0	0.0086524002254009 25	0.3397364914417267	0.6516110897064209
99	0.0	1.0	0.0	0.0444926507771015 2	0.5785022377967834	0.3770051598548889
85	0.0	1.0	0.0	0.0434405803680419 9	0.5587509870529175	0.39780837297439575
122	0.0	0.0	1.0	0.0010373200057074428	0.2190520465373993	0.7799106240272522
124	0.0	0.0	1.0	0.0044940500520169735	0.3184511363506317	0.6770548224449158

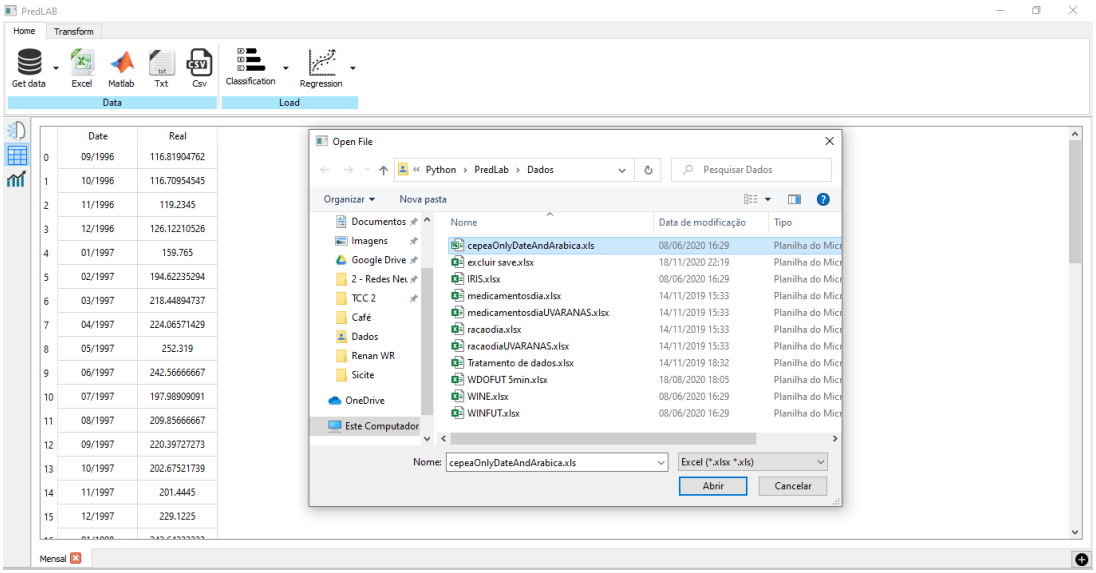
Iris
Iris Outputs

Fonte: Autoria própria.

4.2 REGRESSÃO

Para exemplificar uma aplicação voltada para regressão foi utilizada uma base de dados com o histórico do preço da saca de 60kg do café arábica. A série possui um total 268 amostras referentes ao preço médio mensal de setembro de 1996 até dezembro de 2018. Os dados foram coletados no site do *Centro de Estudos Avançados em Economia Aplicada* (CEPEA) e salvos no computador em formato .xls. Para importá-los utilizou-se a opção de importar dados do computador do usuário com formato *Microsoft Excel*. A Figura 54 ilustra a tela para escolher o arquivo do computador e também os dados refentes a base de dados.

Figura 54 – Base da dados café arábica



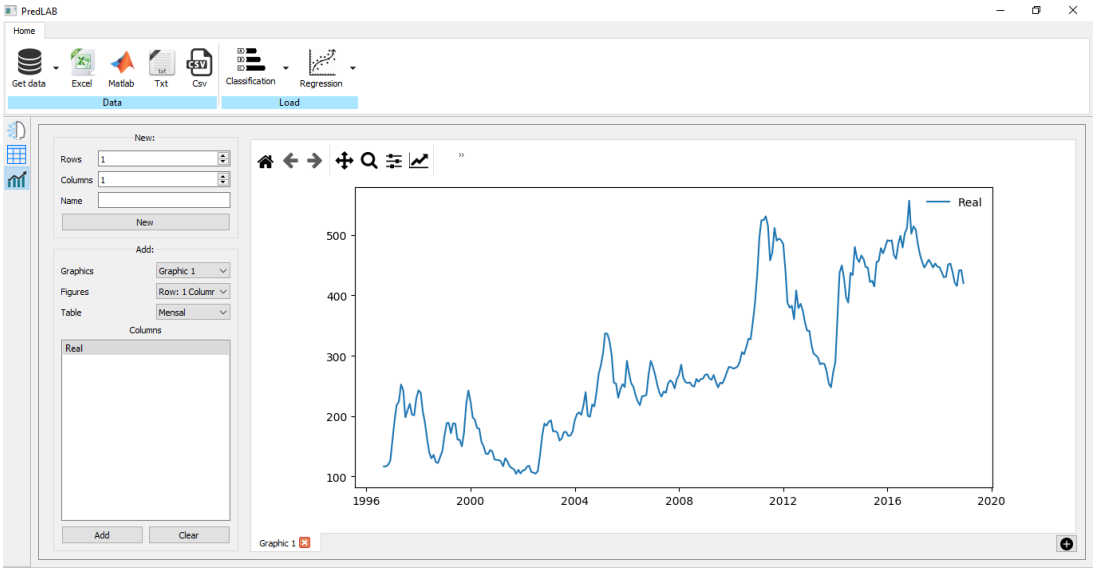
Fonte: Autoria própria.

4.2.1 Pré-processamento - base de dados café arábica

Com o intuito de melhorar o entendimento da série, foi formatada a coluna *Date* como índice da tabela usando a função *Date Formatter* e em seguida os dados foram plotados num gráfico apresentado na Figura 55. Como o índice da tabela foi formatado como data, o eixo x do gráfico pôde ser ajustado de acordo com as datas também, caso contrário seria apenas índices, números.

Para finalizar o tratamento dos dados, foi aplicado o pré-processamento *Min Max Scaler* na coluna ”Real”.

Figura 55 – Gráfico do café arábica



Fonte: Autoria própria.

4.2.2 Seleção dos parâmetros do modelo - café arábica

Após o término do pré-processamento, as próximas etapas ocorrem na tela *AI*. Primeira é ajustar o tipo de aplicação, neste caso um problema de regressão. Posteriormente define-se a escolha das entradas e saídas do modelo.

A série histórica café arábica possui apenas a coluna ”Real”, esta sendo definida tanto como entrada como saída da rede, sendo que a diferença é o nível de deslocamento no tempo. A Figura 56 apresenta um exemplo com 1, 2 e 4 atrasos que seriam a entrada do modelo, enquanto para saída seria feito a previsão da próxima amostra, ou seja, um passo a frente (*Step Forward*).

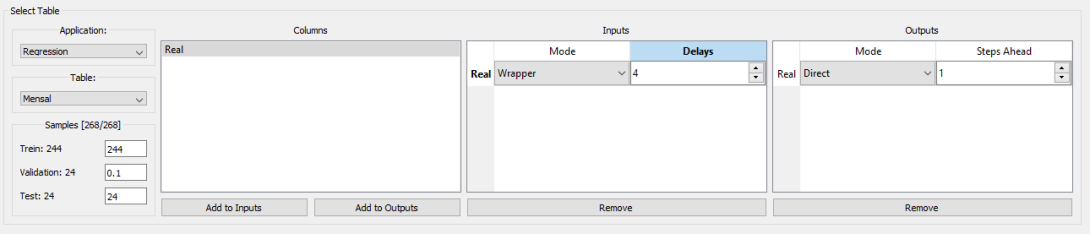
Figura 56 – Dados café arábica deslocados em 1, 2 e 4 atrasos

Date	Delay Real 4	Delay Real 2	Delay Real 1	Output Real 1
01/1997	116.819048	119.234500	126.122105	159.765000
02/1997	116.709545	126.122105	159.765000	194.622353
03/1997	119.234500	159.765000	194.622353	218.448947
04/1997	126.122105	194.622353	218.448947	224.065714
05/1997	159.765000	218.448947	224.065714	252.319000

Fonte: Autoria própria.

Após definida a variável de entrada e de saída, utilizou-se o método *Wrapper* com até 4 atrasos para a variável de entrada, enquanto para variável de saída será feita a previsão de um passo a frente. A Figura 57 ilustra estas configurações.

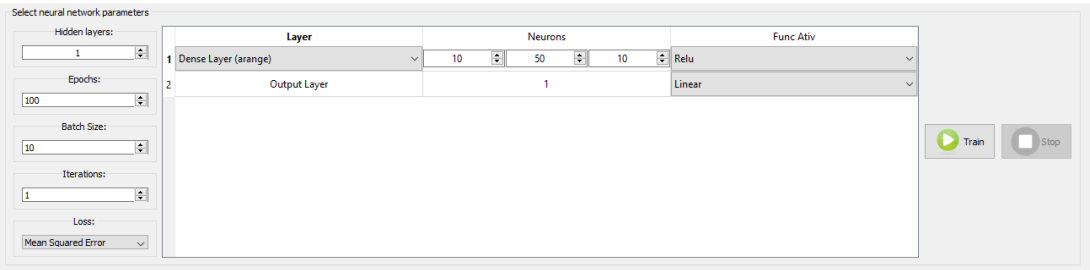
Figura 57 – Seleção das variáveis de entrada e saída - base de dados café arábica



Fonte: Autoria própria.

Para parâmetros da rede neural, há apenas uma camada oculta com função de ativação ” ReLU” e com os neurônios variando entre 10 a 50 com passo de 10. Já para camada de saída optou-se pela função de ativação ”Linear” com apenas um neurônio, que corresponde a previsão da variável ”Real” para um passo a frente. Os demais parâmetros ficaram definidos como: 100 para *epochs*, 10 para *batch size*, e 1 para *Iterations*. A Figura 58 ilustra estes parâmetros.

Figura 58 – Seleção dos parâmetros do modelo - base de dados café arábica



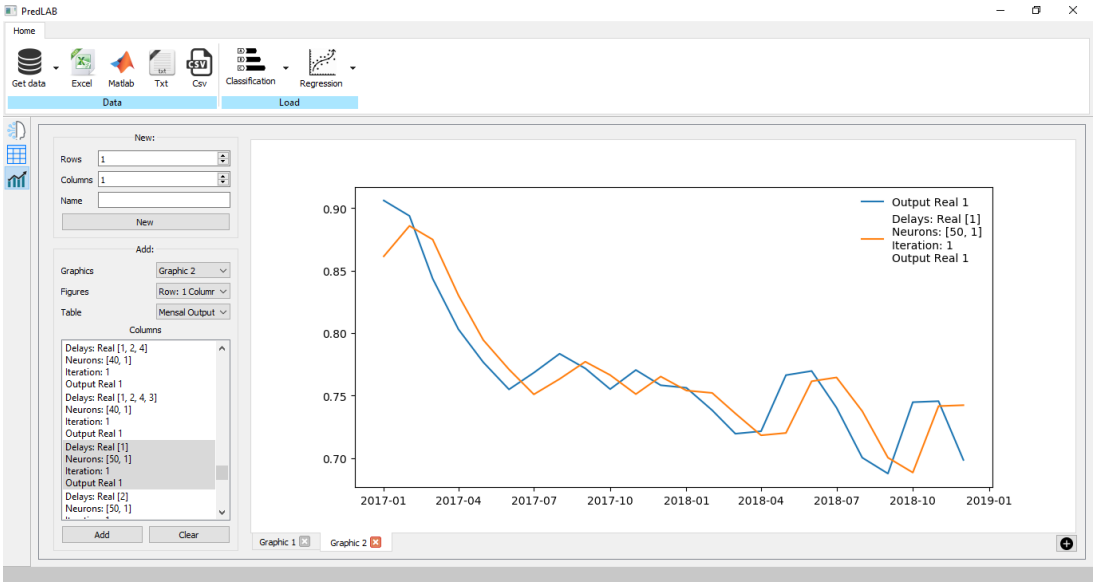
Fonte: Autoria própria.

4.2.3 Resultados gerados - base de dados café arábica

Após o termino do treinamento concluiu-se que o melhor modelo foi com 50 neurônios na camada oculta e apenas um atraso. Este modelo apresentou um MSE de 0,000648.

Buscando entender melhor o desempenho do modelo, foi realizado o pós-processamento das saídas calculadas e então estes plotados junto aos dados reais referente ao teste. A Figura 59 ilustra este gráfico comparativo.

Figura 59 – Gráfico comparativo - base de dados café arábica



Fonte: Autoria própria.

Estes resultados mostram que o *framework* consegue funcionar corretamente para os dois tipos de problemas para o qual foi elaborado: classificação e regressão. Assim, podendo auxiliar em grande medida pesquisadores de áreas diversas, que tenham pouca experiência com este tipo de ferramenta.

5 CONCLUSÕES E PERSPECTIVAS

Este trabalho teve o objetivo de apresentar um *framework*, que visa popularizar a aplicação de redes neurais entre os mais diversos perfis de usuários, mesmo os menos experientes. Possuindo mais de 5300 linhas de código desenvolvidas na linguagem *python*, o principal ganho da sua elaboração é trazer para perto dos usuários o mundo das redes neurais artificiais, agregando mais confiabilidade em suas análises e processamento de dados.

Para tal, este documento apresentou o arcabouço teórico das diversas etapas relativas a pre-processamento de dados, elaboração e arquiteturas de redes neurais, além de aspectos construtivos e de aplicação.

O *framework* disponibiliza a rede MLP para aplicações tanto em regressão quanto em classificação, o que o permite abranger uma quantidade considerável de casos. Junto à compatibilidade com vários tipos de base de dados, este *software* auxiliará o processo, eliminando uma possível necessidade de migrar os dados para outro formato. Outra vantagem trazida pela utilização do *framework* é sua capacidade de tratar os dados inseridos através de técnicas de pré-processamento e *data mining*.

Após os dados já estarem tratados, o usuário pode, em poucos passos, configurar a rede de acordo com as necessidades da aplicação na tela inteligência artificial. Uma vez que a rede esteja configurada, inicia-se o treinamento e os resultados são gerados, sendo possível manipular e exibir as saídas previstas de forma gráfica.

Com isso, a o *sofware* apresentado por este trabalho cumpre os requisitos necessários para uma ferramenta de RNAs acessível.

Para aprimoramento do *framework*, em trabalhos futuros pode-se adicionar mais arquiteturas de *RNAs*, outros algoritmos para ajustes de parâmetros, bem como outros métodos de *data mining*, tornando a ferramenta mais robusta e abrangendo cada vez mais aplicações.

REFERÊNCIAS

ACADEMY, Data Science. **Deep Learning Book**. 2020. Data Science Academy. Disponível em: <http://deeplearningbook.com.br/contato/>. Citado 6 vezes nas páginas 19, 21, 24, 26, 34 e 36.

ACKLEY, David H.; HINTON, Geoffrey E.; SEJNOWSKI, Terrence J. A learning algorithm for boltzmann machines. **Cognitive Science**, v. 9, n. 1, p. 147 – 169, 1985. ISSN 0364-0213. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0364021385800124>. Citado na página 35.

ACTION, Portal. **SAZONALIDADE ESTOCÁSTICA**. 2020. Online. Disponível em: <http://www.portalaction.com.br/series-temporais/232-sazonalidade-estocastica>. Citado na página 18.

BENGIO, Yoshua; COURVILLE, Aaron; VINCENT, Pascal. Representation learning: A review and new perspectives. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 35, n. 8, p. 1798–1828, 2013. Citado na página 23.

BEZERRA, Eduardo. Introdução à Aprendizagem Profunda. **Simpósio Brasileiro de Banco de Dados**, v. 31, 2016. Disponível em: <http://sbbd2016.fpc.ufba.br/sbbd2016/minicursos/minicurso3.pdf>. Citado na página 22.

BORGELT, Christian. **Artificial Neural Networks and Deep Learning**. 2017. Disponível em: <https://www.computational-intelligence.eu/wp-content/uploads/2017/10/NN_02_Threshold_Logic_Units.pdf>. Citado na página 20.

BROWNLEE, Jason. **Difference Between Classification and Regression in Machine Learning**. 2019. Disponível em: <https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/>. Citado na página 32.

CARVALHO, André Ponce de Leon F. de. **Redes Neurais Artificiais Multi Layer Perceptron treinadas com BackPropagation**. 2020. Pagina Pessoal. Disponível em: <https://sites.icmc.usp.br/andre/research/neural/#topicos>. Citado 2 vezes nas páginas 13 e 18.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals, and Systems (MCSS)**, Springer, v. 2, 1989. Citado na página 23.

DOMAIN, Public. **A graphical representation of an example Boltzmann machine**. 2020. Disponível em: <https://commons.wikimedia.org/w/index.php?curid=22915782>. Citado na página 36.

GERON, Aurélien. **Hands–On Machine Learning with Scikit–Learn and TensorFlow**. [S.l.]: O'Reilly Media, Inc., 2019. ISBN 9781492032649. Citado 3 vezes nas páginas 13, 31 e 36.

HAGAN, Martin T.; DEMUTH, Howard B.; JESúS, Orlando De. An introduction to the use of neural networks in control systems. **International Journal of Robust and Nonlinear Control**, John Wiley and Sons, v. 12, 2002. Disponível em: <https://doi.org/10.1002/rnc.727>. Citado 2 vezes nas páginas 32 e 37.

HAYKIN, Simon. **Redes Neurais: Princípios e Prática**. [S.l.]: Bookman, 2003. 898 p. Citado 3 vezes nas páginas 13, 36 e 37.

HAYKIN, Simon O. **Neural Networks and Learning Machines**. 3rd edition. ed. [S.l.]: Prentice Hall, 2008. ISBN 0131471392,9780131471399. Citado na página 37.

HONDA, Hugo; FACURE, Matheus; YAOHAO, Peng. **Os Três Tipos de Aprendizado de Máquina**. 2017. Disponível em: <https://lamfo-unb.github.io/2017/07/27/tres-tipos-am>. Citado 2 vezes nas páginas 21 e 22.

HORNIK, Kurt. Approximation capabilities of multilayer feedforward networks. **Neural Networks**, v. 4, p. 251—-257, 1991. Disponível em: <https://doi.org/10.1016/0893-6080(91)90009-T>. Citado na página 23.

IZQUIERDO, Ivan. **Memoria**. 2002. 95 p. Citado na página 17.

JAY, Prakash. **Back-Propagation is very simple. Who made it Complicated ?** 2020. Online. Disponível em: <https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c>. Citado 2 vezes nas páginas 39 e 40.

JUNIOR, Sergio Luiz Stevan. **Iot - Internet Das Coisas**: Fundamentos e aplicações em arduino e nodemcu). Paraná, Brasil: Editora Erica, 2018. 224 p. Citado na página 13.

KARLIK, Bekir; OLGAC, A Vehbi. 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, p. 111–122, 2011. Citado na página 23.

LAURENE, Fausett. **Fundamentals of Neural Networks: Architectures, Algorithms, and Applications**. USA: Prentice-Hall, Inc., 1994. ISBN 0133341860. Citado na página 35.

LEARN, Scikit. **Sklearn datasets load**$_iris.2020.Scikit-learndevelopers.Disponvelem:$ **<>**. Citado na página 17.

MCCULLOCH, Warren S.; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, p. 115—-133, 1943. Disponível em: <https://doi.org/10.1007/BF02478259>. Citado na página 19.

MISSINGLINK. **7 Types of Neural Network Activation Functions**. 2020. Missinglink.ai. Disponível em: <https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/>. Citado 5 vezes nas páginas 25, 27, 28, 29 e 30.

PAULA, Camila. **Como funciona o sistema nervoso?** 2015. Disponível em: <https://descomplica.com.br/artigo/como-funciona-o-sistema-nervoso/4JL/>. Citado na página 19.

PRIMO.IA. **Deep Learning**. 2020. Online. Disponível em: <http://primo.ai/index.php?title=Deep_Learning>. Citado na página 22.

RAMACHANDRAN, Prajit; ZOPH, Barret; LE, Quoc V. **Searching for Activation Functions**. 2017. Citado na página 28.

RAMASUBRAMANIAN, Karthik; MOOLAYIL, Jojo. **Applied Supervised Learning with R: Use machine learning libraries of R to build models that solve business problems and predict future trends**. [S.l.]: Packt Publishing, 2019. ISBN 1838556338,9781838556334. Citado na página 21.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, American Psychological Association, v. 65, 1958. Disponível em: <https://doi.org/10.1037/h0042519>. Citado 3 vezes nas páginas 13, 32 e 36.

RUMELHART, E. David; HINTON, E. Geoffrey; WILLIAMS, J. Ronald. Learning representations by back-propagating errors. **Nature**, Nature Publishing Group, v. 323, 1986. Citado na página 13.

Schmidt, W. F.; Kraaijveld, M. A.; Duin, R. P. W. Feedforward neural networks with random weights. p. 1–4, 1992. Citado na página 35.

SCHWARTZ, J.; KANDEL, E. **Principios De Neurociencias**. [S.l.]: MCGRAW HILL - ARTMED, 2014. Citado na página 13.

SILVA DANILO HERNAME SPATTI, Rogério Andrade Flauzino Ivan Nunes Da. **Redes neurais artificiais para engenharia e ciências aplicadas: curso prático**. [S.l.]: Artliber Editora Ltda, 2010. ISBN 978-85-88098-53-4. Citado 2 vezes nas páginas 13 e 30.

SIQUEIRA, Hugo Valadares. **Máquinas Desorganizadas para Previsão de Séries de Vazões**. nov. 2013. 244 p. Tese (Doutorado) — Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, nov. 2013. Citado 3 vezes nas páginas 13, 36 e 41.

SUPTITZ IVAN LUIS E FROZZA, Rejane e Molz Rolf Fredi. AnÁlise comparativa de ferramentas de redes neurais artificiais. In: ABEPRO (Ed.). **XXXV ENCONTRO NACIONAL DE ENGENHARIA D E PRODUCAO**. Fortaleza: [s.n.], 2015. Citado na página 44.

VEEN, Fjodor Van. **The Neural Network Zoo**. 2016. The Asimov Institute. Disponível em: <https://www.asimovinstitute.org/neural-network-zoo/>. Citado na página 23.

WERBOS, Paul. **Beyond regression : new tools for prediction and analysis in the behavioral sciences**. Harvard University, 1974. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=ccc7b846fbb77305d1418b3b74b38e98>. Citado na página 13.

WIDROW, B; HOFF, M E. Introdução à Aprendizagem Profunda. **Simpósio Brasileiro de Banco de Dados**, v. 31, 1960. Disponível em: <http://sbbd2016.fpc.ufba.br/sbbd2016/minicursos/minicurso3.pdf>. Citado na página 13.