CENTRO UNIVERSITÁRIO FEI

MURILLO FREITAS BOUZON

ESTUDO DE ALGORITMOS DE OTIMIZAÇÃO INSPIRADOS NA NATUREZA APLICADOS AO TREINAMENTO DE REDES NEURAIS ARTIFICIAIS

São Bernardo do Campo

2021

MURILLO FREITAS BOUZON

ESTUDO DE ALGORITMOS DE OTIMIZAÇÃO INSPIRADOS NA NATUREZA APLICADOS AO TREINAMENTO DE REDES NEURAIS ARTIFICIAIS

Dissertação de Mestrado apresentada ao Centro Universitário FEI para obtenção do título de Mestre em Engenharia Elétrica. Orientado pelo Prof. Dr. Paulo Sérgio Silva Rodrigues.

São Bernardo do Campo

2021

Freitas Bouzon, Murillo.
ESTUDO DE ALGORITMOS DE OTIMIZAÇÃO INSPIRADOS NA NATUREZA APLICADOS AO TREINAMENTO DE REDES NEURAIS ARTIFICIAIS / Murillo Freitas Bouzon. São Bernardo do Campo, 2021.
117 p. : il.

Dissertação - Centro Universitário FEI.
Orientador: Prof. Dr. Paulo Sergio Silva Rodrigues.

1. Algoritmos inspirados na natureza. 2. Aprendizado de máquina. 3. Aprendizado Supervisionado. 4. Redes Neurais Artificiais. 5. Perceptron multicamadas. I. Sergio Silva Rodrigues, Paulo, orient. II. Título.

Elaborada pelo sistema de geração automática de ficha catalográfica da FEI com os dados fornecidos pelo(a) autor(a).

centro universitário FEI

APRESENTAÇÃO DE DISSERTAÇÃO
ATA DA BANCA EXAMINADORA

Programa de Pós-Graduação Stricto Sensu em Engenharia Elétrica

Mestrado

PGE-10

Aluno: Murillo Freitas Bouzon **Matrícula:** 119126-1

Título do Trabalho: ESTUDO DE ALGORITMOS DE OTIMIZAÇÃO INSPIRADOS NA NATUREZA APLICADOS AO TREINAMENTO DE REDES NEURAIS ARTIFICIAIS.

Área de Concentração: Processamento de Sinais e Imagens

Orientador: Prof. Dr. Paulo Sérgio Silva Rodrigues

Data da realização da defesa: 18/03/2021

ORIGINAL ASSINADA

Avaliação da Banca Examinadora:

A banca deliberou em sessão reservada no dia 18 de Março de 2021, tendo considerado o Referido aluno Aprovado. No entanto, recomenda que os comentários realizados durante a arguição devem ser levados em conta pelo aluno, para a entrega da versão final do texto

São Bernardo do Campo, 18 / 03 / 2021 .

MEMBROS DA BANCA EXAMINADORA	
Prof. Dr. Paulo Sérgio Silva Rodrigues	Ass.: _____
Prof. Dr. Guilherme Alberto Wachs Lopes	Ass.: _____
Prof. Dr. Hemerson Pistori	Ass.: _____

A Banca Julgadora acima-assinada atribuiu ao aluno o seguinte resultado:

APROVADO ☒ REPROVADO ☐

VERSÃO FINAL DA DISSERTAÇÃO

APROVO A VERSÃO FINAL DA DISSERTAÇÃO EM QUE FORAM INCLUÍDAS AS RECOMENDAÇÕES DA BANCA EXAMINADORA

_____

Aprovação do Coordenador do Programa de Pós-graduação

_____

Prof. Dr. Carlos Eduardo Thomaz

Dedico este trabalho a Deus, meus pais e a todos que me apoiaram durante essa etapa da minha vida.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Paulo Sergio Rodrigues, pela confiança em mim e no meu trabalho, por toda a paciência e atenção, por todas as aulas e ensinamentos passados a mim, e por me ajudar a amadurecer gradualmente como um pesquisador profissional ao longo dos últimos anos.

A todos os amigos e colegas que interagi ao longo do curso, pelo compartilhamento de ideias e dúvidas, e pelos momentos de discussão e descontração.

A todos os familiares, por sempre terem me apoiado e entenderem a minha decisão de realizar esse curso.

Ao Centro Universitário FEI, pela infraestrutura, disponibilidade de recursos para o desenvolvimento do trabalho e pelos funcionários que me auxiliaram durante esse período.

“O começo de todas as ciências é o espanto de as coisas serem o que são.”

Aristóteles

RESUMO

Redes Neurais Artificiais (RNAs) são técnicas de aprendizado de máquina e inteligência artificial muito populares, propostas desde os anos 50. Entre seus maiores desafios estão o treinamento de parâmetros tais como pesos, parâmetros das funções de ativação e constantes, assim como dos seus hiper-parâmetros, como a arquitetura das redes e densidade de neurônios por camada. Entre os algoritmos mais conhecidos para a otimização paramétrica das redes estão o Adam e o Retropropagação ou *Backpropagation* (BP), aplicados sobretudo em arquiteturas populares como o Perceptron multicamadas ou *Multilayer Perceptron* (MLP), Rede Neural Recorrente ou *Recurrent Neural Network* (RNN), *Long-Short Term Memory* (LSTM), Rede neural de Base Radial ou *Radial Basis Function Neural Network* (RBFNN), entre muitas outras. Recentemente, o grande sucesso das redes neurais profundas, as chamadas *Deep Learnings*, bem como das redes totalmente conectadas, tem enfrentado problemas de tempo de treinamento e utilização de hardware especializado. Esses desafios deram novo fôlego à utilização de algoritmos de otimização para o treinamento dessas redes, e mais recentemente aos algoritmos inspirados na natureza, os chamados Inspirados na natureza ou *Nature-Inspired* (NI). Essa estratégia, embora não seja uma técnica tão recente, ainda não obteve grande atenção de pesquisadores, necessitando hoje de maior número de testes experimentais e avaliação, sobretudo devido ao recente aparecimento de uma gama muito maior de algoritmos NI. Alguns dos elementos que carecem de atenção, sobretudo para os NI mais recentes, estão relacionados principalmente ao tempo de convergência e estudos sobre o uso de diferentes funções custo. Assim, a presente dissertação de mestrado tem por objetivo realizar testes, comparação e estudos sobre algoritmos NI aplicados ao treinamento de redes neurais. Foram testados algoritmos NI tanto tradicionais quanto recentes, sob vários pontos de vista, incluindo o tempo de convergência e funções objetivos, elementos que receberam até o momento pouca atenção dos pesquisadores em testes prévios. Os resultados mostraram que a utilização de algoritmos NI para treinamento de RNAs tradicionais obtiveram resultados com boa classificação, similares a algoritmos populares como o *Adam* e o Algoritmo *Backpropagation* com Momento (BPMA), mas superando esses algoritmos em termos de tempo de convergência na ordem de 20 a mais de 70%, dependendo da rede e dos parâmetros envolvidos. Isso indica que a estratégia de usar algoritmos NI, sobretudo os mais recentes, para treinamento de redes neurais é um método promissor que pode impactar cada vez mais no tempo e qualidade dos resultados das aplicações recentes e futuras de aprendizado de máquina e inteligência artificial.

Palavras-chave: Algoritmos inspirados na natureza. Aprendizado de máquina. Aprendizado Supervisionado. Redes Neurais Artificiais. Perceptron multicamadas.

ABSTRACT

Artificial Neural Networks are a popular machine learning and artificial intelligence technique, proposed since the 1950s. Among their greatest challenges is the training of parameters such as weights, parameters of the activation functions and constants, as well as their hyperparameters, such as network architecture and density of neurons per layer. Among the best known algorithms for parametric optimization of networks are Adam and BP, applied mainly in popular architectures such as MLP, RNN, LSTM, *Feed-forward Neural Network* (FNN), RBFNN, among many others. Recently, the great success of deep neural networks, known as Deep Learnings, as well as fully connected networks, has faced problems with training time and the use of specialized hardware. These challenges gave new impetus to the use of optimization algorithms for the training of these networks, and more recently to the algorithms inspired by nature, also called as NI. This strategy, although not a recent technique, has not yet received much attention from researchers, requiring today a greater number of experimental tests and evaluation, mainly due to the recent appearance of a much larger range of algorithms NI. Some of the elements that need attention, especially for the most recent NI, are mainly related to the time of convergence and studies on the use of different cost functions. Thus, the present master's dissertation aims to perform tests, comparisons, and studies on algorithms NI applied to the training of neural networks. Both traditional and recent NI algorithms were tested, from many perspectives, including convergence time and cost functions, elements that until now have received little attention from researchers in previous tests. The results showed that the use of NI algorithms for the training of traditional RNAs obtained results with good classification, similar to popular algorithms such as Adam and BPMA, but surpassing these algorithms in terms of convergence time in 20 up to 70%, depending on the network and the parameters involved. This indicates that the strategy of using NI algorithms, especially the most recent ones, for training neural networks is a promising method that can impact the time and quality of the results of recent and future machine learning applications and artificial intelligence.

Keywords: Nature-inspired Algorithms. Machine Learning. Supervised Learning. Artificial Neural Network. Multilayer perceptron.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS

ABC	Colônia de abelhas artificiais ou *Artificial Bee Colony*.
ACO	Colônia de formigas ou *Ant Colony Optimization*.
ACRNN	*Artificial Chemical Reaction Neural Network*.
ACRO	*Artificial Chemical Reaction Optimization*.
AFSA	*Artificial Fish Swarm Algorithm*.
ALO	*Antlion Optimizer*.
ANFIS	*Adaptive Neuro-Fuzzy Inference System*.
ANN	*Artificial Neural Network*.
AUC	*Area Under the Curve*.
BAT	*Bat Algorithm*.
BBO	*Biogeography Based Optimization*.
BP	Retropropagação ou *Backpropagation*.
BPMA	Algoritmo *Backpropagation* com Momento.
BPNN	*Backpropagation Neural Network*.
BSO	*Brain Storm Optimization*.
CAD	*Computer-Aided Diagnostic*.
CAPSO	*Centripetal Accelerated Particle Swarm Optimization*.
CDOA	*Collective Decision Optimization Algorithm*.
CE	Erro de classificação ou *Classification Error*.
CER	Taxa de erro de classificação ou *Classification Error Rate*.
CNN	Rede neural convolucional ou *Convolutional Neural Network*.
CS	*Cuckoo Search*.
DE	Evolução Diferencial ou *Differential Evolution*.
DFO	*Dragonfly Optimization*.
DNN	Rede neural profunda ou *Deep Neural Network*.
DS	*Differential Search*.
EA	Evolutionary Algorithm.
ELM	*Extreme Learning Machine*.
EPC	*Emperor Penguins Colony*.
ES	*Evolution Strategy*.
FCM	*Fuzzy C-Means*.
FFA	*Firefly Algorithm*.
FMD	Frequência de Melhor Desempenho.
FN	*False Negative*.
FNN	*Feed-forward Neural Network*.
FP	*False Positive*.
FSCABC	*Fitness-Scaled Chaotic Artificial Bee Colony*.

GA	Algoritmo Genético ou *Genetic Algorithm*.
gbest	*Global Best*.
GGBFS	*Ground Granulated Blast Furnace Slag*.
GLCM	*Gray-Level Co-Occurrence Matrix*.
GSA	*Gravitational Search Algorithm*.
GWO	*Grey Wolf Optimizer*.
HPABC	*Hybrid Particle-move Artificial Bee Colony*.
HS	*Harmony Search*.
HSI	*habitat suitability index*.
IALO	*Improved Antlion Optimizer*.
ICA	*Imperialist Competitive Algorithm*.
IHS	*Improved Harmony Search*.
IMBO	*Improved Monarch Butterfly Optimization*.
IWO	*Invasive Weed Optimization*.
KHA	*Krill Herd Algorithm*.
LDW	*Linearly Decreasing Weight*.
LDWPSO	*Linearly Decreasing Weight Particle Swarm*.
LM	*Levenberg-Marquardt*.
LSA	*Lightining Search Algorithm*.
LSTM	*Long-Short Term Memory*.
MAE	Erro médio absoluto ou *Mean Absolute Error*.
MAPE	Porcentagem do erro médio absoluto ou *Mean Absolute Percentage Error*.
MBO	*Monarch Butterfly Optimization*.
MFO	*Moth-Flame Optimization*.
MLP	Perceptron multicamadas ou *Multilayer Perceptron*.
MLR	*Multiple Linear Regression*.
MSE	Erro médio quadrático ou *Mean Squared Error*.
MVO	*Multi-verse Optimizer*.
NGHS	*Novel Global Harmony Search*.
NI	inspirados na natureza ou *Nature-Inspired*.
NMPSO	Novo modelo do *Particle Swarm Optimization*.
pbest	*Personal Best*.
PBIL	*Population-based incremental learning*.
PCA	Análise de components principais ou *Principal Component Analysis*.
PPV	*Peak Particle Velocity*.
PSNN	*Pi-Sigma Neural Network*.
PSO	*Particle Swarm Optimization*.
PUNN	*Product Unit Neural Network*.

RBFNN	Rede neural de Base Radial ou *Radial Basis Function Neural Network*.
ReLU	Unidade Linear Retificada ou *Rectfied Linear Unity*.
RMSE	Raiz do erro médio quadrático ou *Root Mean Square Error*.
RNA	Rede Neural Artificial.
RNN	Rede Neural Recorrente ou *Recurrent Neural Network*.
SCA	Algoritmo Seno-Cosseno ou *Sine Cosine Algorithm*.
SCG	*Scaled Conjugate Gradient*.
SGHS	*Self-adaptive Global Best Harmony Search*.
SGPSO	Segunda geração do *Particle Swarm Optimization*.
SIV	*suitability index variables*.
SMO	*Spider Monkey Optimization*.
SSE	Soma dos erros quadráticos ou *Sum of Squared Errors*.
SVM	*Support Vector Machine*.
Tanh	Tangente Hiperbólica.
TDR	*Travelling Distance Rate*.
TE	*Tennessee Eastman*.
TN	*True Negative*.
TO	*Torus*.
TP	*True Positive*.
TPR	*True Positive Rate*.
UCI	Universidade de California Irvine.
WEP	*Wormhole Existence Probability*.
WOA	*Whale Optimization Algorithm*.

SUMÁRIO

1 INTRODUÇÃO

Aplicações que utilizam aprendizado de máquina estão se tornando cada vez mais populares e sendo requisitadas por empresas que ainda não aderiram a essa tecnologia, fazendo com que o investimento no desenvolvimento de projetos que utilizam essa tecnologia crescesse bastante, recebendo mais de 80 bilhões de dólares ao redor do mundo apenas no primeiro trimestre de 2019 (FELDMAN, 2019). Alguns exemplos do uso de técnicas de aprendizado de máquina são a automação de tarefas, otimização de desempenho, previsões de risco e personalização de serviços. Na área médica, por exemplo, o aprendizado de máquina pode ser utilizado para auxiliar médicos, aprimorando a acurácia de diagnósticos e permitindo a identificação de pacientes em risco. Outro exemplo é a recomendação de produtos, como aplicações de lojas virtuais, que adaptam as suas recomendações ao cliente, de acordo com as informações de compras e navegação utilizadas para auxiliar o aprendizado dos gostos e desejos do consumidor.

O aprendizado supervisionado é um tipo de aprendizado de máquina, onde é necessária a alimentação de dados com respostas conhecidas para o modelo aprender os padrões relacionados à base recebida e conseguir predizer ou classificar corretamente novas amostras. Com o crescimento da quantidade de dados coletados e disponíveis abertamente, diversos modelos de classificação foram idealizados e implementados para resolverem problemas do mundo real. Todavia, esses modelos ainda possuem diversas limitações, encontrando os principais desafios em bases de dados de problemas mais complexos, que possuem soluções não lineares.

As RNAs são uma técnica de aprendizado de máquina, inspirados no sistema biológico neural, que atraiu o interesse de muitos pesquisadores nas últimas três décadas, sendo exploradas na classificação, reconhecimento de padrões, regressão e agrupamento de dados. Os algoritmos tradicionais de treinamento de uma rede neural, como o BP, por exemplo, possibilitaram a resolução de diversos problemas de classificação, mostrando uma boa capacidade de generalização e aprendizado da rede, fazendo com que ela fosse amplamente utilizada em problemas complexos, sendo aplicadas para o auxílio de hospitais (BAHADORI et al., 2020), controle de consumo de energia (LIN et al., 2020), classificação de imagens (QIN et al., 2020), controle de robôs (YANG et al., 2019), entre outros exemplos (ABIODUN et al., 2018). No entanto, o processo de treinamento acaba tendo um alto custo computacional e pode levar à soluções ótimas locais, essencialmente em problemas que possuem soluções não lineares, dificultando a otimização dos parâmetros para a rede obter resultados melhores (ZHANG; SUGANTHAN, 2016; YANG; MA, 2019).

Paralelamente, uma categoria de algoritmos de otimização que ganhou notoriedade nas últimas décadas foram os algoritmos de otimização NI, sendo algoritmos de otimização computacional desenvolvidos a partir da observação de processos naturais ou comportamento de organismos biológicos, como animais e insetos, por exemplo. A sua popularidade crescente vem da sua capacidade de aprender e se adaptar a diversos problemas complexos que ainda não tinham sido resolvidos, fornecendo soluções precisas com um bom custo computacional (MO-

LINA et al., 2020). Um dos problemas para os quais esses algoritmos estão sendo explorados é a otimização dos parâmetros de uma rede neural, existindo diversas possibilidades de pesquisa devido ao número de algoritmos NI que existem, tendo como vantagem, em relação aos algoritmos de treinamento tradicionais, a boa capacidade de exploração de soluções no espaço de busca, evitando mínimos locais, a rápida taxa de convergência e a resolução de problemas desafiadores (HEMEIDA, Ashraf Mohamed et al., 2020).

Sendo assim, esse trabalho apresenta uma metodologia para avaliar e comparar o uso de algoritmos de otimização NI para o treinamento de redes neurais artificiais. Foram selecionados 9 algoritmos NI diferentes e 3 algoritmos de treinamento tradicionais, que são aplicados em uma rede neural MLP para a classificação de 9 bases de dados selecionadas. Após o treinamento, é feita a validação da rede para gerar métricas de avaliação de modo a analisar o desempenho dos algoritmos e compará-los entre si em termos de acurácia, robustez e tempo de execução.

1.1 OBJETIVO

Esse trabalho tem como objetivo a avaliação e comparação do uso de algoritmos de otimização inspirados na natureza para treinar redes neurais artificiais classificadoras.

1.2 ESTRUTURA DO TRABALHO

O restante deste trabalho é dividido da seguinte maneira:

No Capítulo 2, serão descritos trabalhos relacionados disponíveis na literatura, com o objetivo de apresentar o cenário atual de pesquisa da área.

No Capítulo 3, serão apresentados todos os conceitos utilizados e relacionados ao tema abordado, para que o leitor possa entender com clareza as técnicas que estão sendo tratadas no trabalho e compreender os termos que serão descritos posteriormente.

O Capítulo 4 detalhará a metodologia que foi utilizada para o desenvolvimento deste trabalho, demonstrando as técnicas que foram utilizadas e os passos realizados para atingir o objetivo final.

No Capítulo 5 serão apresentados os resultados obtidos e discutirá sobre eles, analisando o desempenho dos algoritmos em diferentes cenários e sendo feita uma discussão sobre.

No capítulo 6 será feita a conclusão do trabalho, levando em conta todo o processo para alcançar o resultados obtidos nesse trabalho, apresentando os principais pontos que puderam ser concluídos nesse trabalho.

2 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados os trabalhos que possuem alguma relação com a aplicação de algoritmos NI para a otimização de redes neurais MLP. O capítulo foi estruturado dividindo os trabalhos em seções, sendo consideradas seções para categorias diferentes de algoritmo NI, uma seção para artigos de revisão pelo fato de terem sido encontrados artigos de revisão da área recentes, e uma seção para trabalhos que utilizaram técnicas de otimização NI aplicadas à aprendizado profundo, por ser uma área nova que está sendo cada vez mais requisitada em diversas aplicações. Assim, os trabalhos foram divididos da seguinte forma: Algoritmos de otimização evolucionários (Seção 2.1); Algoritmos de otimização baseados em inteligência de enxame (Seção 2.2); Algoritmos de otimização híbridos (Seção 2.3); Algoritmos de otimização baseados em conceitos físicos/químicos (Seção 2.4); Algoritmos de otimização baseados em comportamentos sociais humano (Seção 2.5); Artigos de revisão (Seção 2.6); Algoritmos de otimização aplicados em aprendizado profundo (Seção 2.7).

2.1 ALGORITMOS DE OTIMIZAÇÃO EVOLUCIONÁRIOS

Algoritmos evolucionários são algoritmos de otimização NI que se inspiram em princípios de evolução natural, onde cada indivíduo representa uma possível solução do problema em questão e se reproduzem em um ambiente em que as características das soluções mais fortes (mais próximas da solução esperada) são priorizadas para as próximas gerações. Alguns exemplos conhecidos de algoritmos evolucionários são o Algoritmo Genético ou *Genetic Algorithm* (GA), *Biogeography Based Optimization* (BBO) e Evolução Diferencial ou *Differential Evolution* (DE).

Um dos primeiros trabalhos que obteve sucesso em aplicar um algoritmo NI evolucionário para treinar uma rede neural MLP foi o artigo de Montana e Davis (1989), utilizando a técnica GA. No mesmo ano, Miller, Todd e Hegde (1989) seguiram por uma linha diferente, buscando otimizar a arquitetura da rede, ao invés dos seus pesos, também utilizando o GA. Como essas técnicas de otimização eram recentes na época, não se tinha muito conhecimento do impacto que os parâmetros dos algoritmos causavam no resultado final. Por este motivo, como apontado na revisão feita por Schaffer, Whitley e Eshelman (1992), a grande maioria dos trabalhos onde foi feita a otimização de redes neurais utilizando o GA demonstrou resultados inferiores comparado com métodos baseados em gradiente.

No estudo de Mirjalili, Mirjalili e Lewis (2014b) foi proposto o uso de um algoritmo de otimização meta-heurístico que se baseia em conceitos geográficos, chamado de BBO, para treinar uma rede MLP. Os pesos da rede neural são codificados para serem guardados em cada *habitat* do algoritmo BBO, que otimiza os pesos minimizando o valor de MSE de saída. Para testar o método, foram utilizadas 5 bases de classificação e 6 funções matemáticas para aproximação. Os resultados foram comparados com 5 algoritmos meta-heurísticos consolidados e

com as técnicas BP e *Extreme Learning Machine* (ELM). Comparado com os algoritmos metaheurísticos, o BBO se mostrou melhor em evitar mínimos locais, além de ser superior em termos de acurácia e tempo de convergência.

O uso de uma rede MLP, treinada pelo algoritmo BBO, para identificação de *spam* foi proposto por Rodan, Faris, Alqatawna et al. (2016). No algoritmo BBO, cada indivíduo é chamado de *habitat*. Para o problema de treinar uma rede MLP, cada hábitat representou um conjunto de pesos, sendo gerado um número pré-determinado de hábitats. Para cada hábitat gerado, foi calculado o seu MSE como função objetivo, utilizando uma base de treinamento dada como entrada para cada rede. Em seguida, as redes MLP são combinadas e mutacionadas. Algumas redes com o menor valor de MSE são mantidas para serem passadas para a próxima geração. Esse processo é repetido por um número fixo de iterações. O método foi testado nas bases *Spambase* e *SpamAssassin*, sendo comparado com os algoritmos GA, PSO, DE, Colônia de formigas ou *Ant Colony Optimization* (ACO) e BP. O algoritmo BBO foi o método que convergiu mais rápido e obteve a melhor acurácia em ambas as bases, com os valores de 0.88 na base *Spambase* e 0.866 na base *SpamAssassin*.

Um sistema para classificação de frutas foi proposto por Yudong Zhang et al. (2016). Na primeira etapa do método, foi feito um pré-processamento da imagem, onde a imagem foi segmentada pelo algoritmo *split-and-merge* para eliminar o plano de fundo e então a resolução da imagem foi reduzida para aumentar a velocidade do algoritmo. A segunda parte consiste na extração das características de cor, formato e textura. As características desnecessárias foram eliminadas com o algoritmo PCA na terceira etapa. Na última etapa, foi construído um classificador com uma rede neural *feed-forward* treinada pelo algoritmo BBO, chamada de BBO-FNN. Foi utilizada uma base contendo 1653 imagens de frutas, com 18 classes no total, para testar o método, sendo comparado com outras 5 técnicas. Os resultados mostraram que a BBO-FNN obtiveram uma acurácia de 89.11%, mostrando-se superior aos outros métodos comparados.

Um método para o diagnóstico de diabete utilizando uma rede neural geneticamente otimizada foi apresentado por Mahajan, Kumar e Bansal (2017). Foi feita a leitura da base de entrada e então utilizou-se o algoritmo Análise de components principais ou *Principal Component Analysis* (PCA) para reduzir a sua dimensionalidade. Em seguida, foi utilizado o GA para encontrar os pesos ótimos da rede neural, minimizando o valor de MSE de saída como função objetivo. O método foi comparado com a técnica BP para o diagnóstico de diabete. Foi obtida uma acurácia de 96.1% para o método proposto e 92.2% do BP.

O trabalho de Dorado-Moreno et al. (2017) teve como objetivo a criação de um modelo eficiente de suporte de decisão para auxiliar médicos especialistas no processo de alocação de órgãos durante o transplante de fígado. Para isso, foi utilizada uma base de dados com informações de transplantes conduzidos em 8 hospitais diferentes, com o estado de pacientes acompanhados por até 12 meses. O problema é tratado como um problema de classificação, onde é previsto quanto tempo o órgão do paciente irá sobreviver, sendo delimitados 4 períodos: menos de 15 dias; entre 15 e 90 dias; entre 90 e 365 dias; mais de 365 dias. Essa delimitação foi feita para

apurar melhor as informações de sobrevivência, no entanto, resultou em um desbalanceamento da base, com 85% das amostras pertencendo à última classe. Para resolver isso, foi combinado o uso de uma rede neural treinada pelo algoritmo Evolutionary Algorithm (EA), utilizando uma função de avaliação dinâmica, e uma técnica para ampliar a amostragem da base e aliviar o seu desbalanceamento. O modelo foi comparado com outras técnicas do estado da arte na época, conseguindo predizer corretamente mais de 73% dos resultados dos transplantes com uma média geométrica de 31.46%, sendo superior aos valores encontrados pelos outros métodos.

O treinamento de uma RBFNN foi abordado por Aljarah et al. (2018), sendo adotado o algoritmo BBO para encontrar os valores ótimos do centro, largura, pesos da RBFNN, visando minimizar o MSE obtido pela saída da rede. O método foi empregado em 12 bases de dados e comparado com 11 métodos de treinamento disponíveis na literatura. Com os resultados obtidos, os autores concluíram que o algoritmo BBO apresenta uma rápida convergência e um valor alto de acurácia, conseguindo evitar ótimos locais no espaço de busca do problema de treinamento de RBFNN, além de treiná-las efetivamente para a classificação de diferentes bases de dados que possuem um número diverso de características e amostras de treino.

2.2 ALGORITMOS DE OTIMIZAÇÃO BASEADOS EM INTELIGÊNCIA DE ENXAME

Outra categoria de algoritmo NI são algoritmos baseados em inteligência de enxame, se inspirando em comportamentos coletivos de sociedades do reino animal, como colônias de insetos, para solucionar problemas de otimização de forma eficiente. Essa categoria de algoritmos NI contém a maior quantidade de algoritmos em relação às outras categorias, possuindo algoritmos conhecidos como o PSO, ACO, FFA e o GWO.

Com o surgimento da categoria de algoritmos de otimização por inteligência de enxame, esses algoritmos logo começaram a serem aplicados para o treinamento de redes neurais, como no trabalho de Engelbrecht, Engelbrecht e Ismail (1999), que utilizou os algoritmos PSO, GA, *Random Search* e *Leap Frog* para treinar uma *Product Unit Neural Network* (PUNN). Além disso, também foram exploradas estratégias diferentes para combinar o uso de algoritmos metaheurísticos NI com redes neurais, como no trabalho de Van den Bergh (1999), onde foi utilizado o algoritmo PSO para selecionar os pesos iniciais ótimos que maximizam a performance do algoritmo *Scaled Conjugate Gradient* (SCG) para treinar uma rede MLP. Outra aplicação observada na bibliografia, relacionada ao uso de algoritmos de otimização NI em conjunto com redes neurais, é a otimização da seleção de características da base de entrada da rede utilizando o algoritmo ACO, como apresentada no estudo de Sivagaminathan e Ramakrishnan (2007).

No trabalho de Mavrovouniotis e Yang (2015) foi aplicado o algoritmo ACO para otimizar os pesos de uma rede neural. Os autores geraram uma população de formigas que selecionam diferentes combinações de pesos das conexões de uma rede neural, buscando minimizar o valor de Soma dos erros quadráticos ou *Sum of Squared Errors* (SSE). Além disso, foi criado um método híbrido com a técnica BP chamado de ACO-BP. O desempenho do ACO e do método

híbrido ACO-BP foi avaliada em 9 bases de dados diferentes, sendo comparada com outros métodos de treinamento convencionais e com os algoritmos PSO, GA, Colônia de abelhas artificiais ou *Artificial Bee Colony* (ABC) e DE. A partir dos resultados obtidos, os autores concluíram os seguintes pontos: (i) o algoritmo ACO foi a melhor escolha para selecionar pesos iniciais para o treinamento feito pelo método BP, sendo o ACO-BP o método que obteve o melhor desempenho; (ii) a qualidade dos resultados dos métodos baseados em descida de gradiente se degrada de acordo com o aumento do tamanho do problema quando comparado com o ACO-BP; (iii) os algoritmos de descida de gradiente geralmente são mais precisos que os métodos meta-heurísticos em relação à busca; (iv) o algoritmo ACO possui um bom desempenho comparado com outros algoritmos meta-heurísticos no treinamento de redes para classificação de dados.

No estudo de Garro e Vázquez (2015) foi apresentada uma metodologia para construir automaticamente uma rede neural utilizando os algoritmos PSO, Segunda geração do *Particle Swarm Optimization* (SGPSO) e Novo modelo do *Particle Swarm Optimization* (NMPSO). Os algoritmos evoluem, ao mesmo tempo, os pesos sinápticos, as conexões e as funções de ativação de cada neurônio da rede. Foram consideradas 8 funções de avaliação diferentes, baseadas no MSE e Taxa de erro de classificação ou *Classification Error Rate* (CER), e foi implementada uma estratégia para evitar o treinamento excessivo e reduzir o número de conexões da rede. Os algoritmos foram testados em 10 bases de classificação. Em geral, o método NMPSO proposto obteve resultados melhores que o PSO e SGPSO. As funções de ativação mais utilizadas foram a função gaussiana para o PSO e NMPSO e a função sinusoidal para o SGPSO.

Os autores Mandal, Saha e Pal (2015) utilizaram o algoritmo de otimização FFA, que se baseia no comportamento dos vaga-lumes para buscar uma solução ótima, para treinar uma rede neural *feed-forward* de classificação. O problema foi modelado de forma que cada vaga-lume representasse um conjunto de pesos utilizados por uma rede MLP. Os vaga-lumes foram utilizados para buscar no espaço da solução os pesos que minimizam o erro quadrático, sendo o vaga-lume mais brilhoso aquele que está mais próximo da solução ótima, guiando os outros vaga-lumes para uma resposta melhor. Foi feita uma simulação com uma base contendo 2 entradas, 1 saída e 6 amostras. Os autores concluíram que a rede neural treinada pelo FFA conseguiu minimizar o erro do treinamento em uma taxa de convergência rápida. No entanto, o tamanho da amostra de treino, a população de vaga-lumes e o número de iterações devem ser suficientemente alto para obter uma melhor acurácia.

Outro trabalho que utilizou o FFA para otimizar os pesos de uma rede neural foi o de Brajevic e Tuba (2013). Os vaga-lumes representavam um conjunto de pesos e buscaram minimizar a função de erro MSE. Foi feita uma comparação entre as técnicas FFA, GA e ABC para resolver o problema do XOR e de paridade de bits. O FFA obteve um desempenho melhor comparado ao GA, mas pior em relação ao ABC na maioria dos testes realizados. A autora apontou que o uso da função sigmoide fez com que o algoritmo tivesse uma tendência de cair em mínimos locais. Ao mudar a função de saída para a função $sen(x)$, essa tendência foi reduzida e o FFA obteve 100% de sucesso e uma rápida convergência em quase todos os testes.

No trabalho de Nayak, Naik e Behera (2016) também foi aplicada a técnica FFA para treinar uma rede neural, sendo adotada uma *Pi-Sigma Neural Network* (PSNN) neste caso. O FFA foi utilizado para computar os pesos ótimos da PSNN para aprimorar a acurácia da sua classificação. Cada vaga-lume busca pelo melhor conjunto de pesos seguindo como função de avaliação a minimização do Raiz do erro médio quadrático ou *Root Mean Square Error* (RMSE) de saída da rede. A PSNN treinada pelo FFA foi testada em 11 bases de classificação diferentes e comparada com os métodos PSO, GA e o método híbrido GA-PSO. Os resultados demonstraram que o FFA-PSNN pode ser utilizado para resolver problemas de classificação de diversos tipos, atingindo uma rápida convergência e baixa taxa de erro comparado com os outros métodos testados.

No estudo de Jaddi, Abdullah e Hamdan (2015) foi proposta uma nova solução para representar os pesos e a estrutura de uma rede neural através de uma modificação da técnica BAT. Foi feita uma modificação no ajuste da velocidade do BAT, considerando as melhores soluções locais e as soluções globais, e também são empregados 3 mapas caóticos. Essas modificações têm como objetivo aprimorar a capacidade de busca e exploração das soluções já encontradas. Diferentes versões do BAT foram incorporadas para selecionar a estrutura e o peso de redes neurais durante o seu processo de treinamento. Foi então utilizado o método *Taguchi* para melhorar os parâmetros da versão do algoritmo que se demonstrou melhor. O método foi testado em 6 bases de classificação e 2 bases de séries temporais. Os testes estatísticos demonstraram que o método proposto gerou alguns dos melhores resultados comparados com os métodos disponíveis na literatura.

A comparação da técnica *Krill Herd Algorithm* (KHA) com outras técnicas de otimização NI e convencionais foi feita por Kowalski e Łukasik (2016), no contexto de treinamento de redes neurais. Cada krill do algoritmo representou um conjunto de pesos. Como função objetivo, utilizaram-se as métricas Erro de classificação ou *Classification Error* (CE) e SSE em conjunto, de forma que os pesos fossem ajustados para minimizar as duas métricas. O método foi testado em 4 bases de dados do repositório Universidade de California Irvine (UCI) e comparado com as técnicas *Harmony Search* (HS), GA e BP. O KHA se mostrou o melhor entre os algoritmos NI, porém, teve uma acurácia inferior ao método BP na maioria das bases. Por outro lado, o número de interações para convergir foi bem menor em relação ao BP.

No trabalho de Mirjalili (2015) foi utilizado o algoritmo GWO para treinar uma rede MLP, sendo aplicado para encontrar os pesos ótimos de forma que seja minimizado o MSE calculado a partir das saídas da rede. O método foi testado em 5 bases de classificação e 3 funções para serem aproximadas, sendo comparado com outras 5 técnicas. O algoritmo GWO apresentou a acurácia mais alta na maioria das bases testadas, graças à sua estratégia de exploração, que evita a estagnação dos agentes de busca em ótimos locais.

No trabalho de Yamany et al. (2015) foi proposto um método chamado de MFO-MLP, que utiliza o algoritmo *Moth-Flame Optimization* (MFO) para buscar por pesos de uma rede neural MLP que minimizam o valor de MSE da saída dada pela rede. O método foi testado para

resolver o problema do XOR, a base *Iris*, *Heart, Breast Cancer* e *Balloon*. Além disso, ele foi testado para encontrar a aproximação de 3 funções matemáticas, sendo comparado com outros 4 algoritmos de otimização. Para as 3 funções matemáticas, o MFO se mostrou o método com o menor valor de MSE. Para as outras bases de dados, obteve a maior acurácia nas 4 bases de classificação testadas e ficou em segundo lugar para o problema do XOR.

O algoritmo PSO foi aplicado para otimizar os parâmetros de uma rede neural *feed-forward* no trabalho de Chhachhiya, Sharma e Gupta (2017). Os parâmetros considerados são o número de neurônios da camada escondida, a taxa de aprendizado e a função de ativação. A função de avaliação escolhida foi a minimização do RMSE. A rede é treinada pelo algoritmo BP que recebe como entrada uma base de dados com informações sobre colégios particulares e retornando como saída uma nota para colégio. O método encontrou como parâmetros ótimos 0.7 ou 0.8 de taxa de aprendizado; 4, 5 ou 6 neurônios na camada escondida e a função sigmoide bi-polar como função de ativação, obtendo 0.0000119 de RMSE e uma acurácia de 92.1671%.

No trabalho de Devikanniga e Raj (2018) foi desenvolvido um modelo classificador para discriminar pacientes osteoporóticos de pacientes saudáveis se baseando em valores de densidade mineral óssea. Foi utilizada uma rede neural de classificação com os pesos otimizados pelo algoritmo *Monarch Butterfly Optimization* (MBO), adotando a minimização do MSE da saída da rede como função objetivo. Para os experimentos, utilizou-se uma base com dados da espinha lombar e outra base com dados do colo do fêmur. O método obteve uma acurácia de 97.9%, 98.33% de especificidade e 95.24% de sensitividade na primeira base. Para a segunda base, foram obtidos os valores de 99.3%, 99.2% e 100% de acurácia, especificidade e sensitividade respectivamente.

O algoritmo MBO também foi utilizado no trabalho de Faris, Aljarah e Mirjalili (2018), sendo proposta uma versão aprimorada chamada *Improved Monarch Butterfly Optimization* (IMBO). Foi feita uma modificação da atualização de posições do algoritmo MBO para considerar soluções anteriores combinadas com a melhor solução encontrada até o momento com o intuito de melhorar o equilíbrio entre a esquiva de ótimos locais e a velocidade de convergência. O método foi aplicado para a otimização de pesos de uma MLP, minimizando o MSE como função objetivo, sendo comparado com outros 11 métodos em dois experimentos. O primeiro avaliou o método para aproximar funções matemáticas, mostrando que o IMBO se beneficia da evasão de ótimos locais e da sua rápida convergência, superando o algoritmo MBO tradicional e apresentando resultados comparáveis, e alguns até superiores, com outras abordagens presentes na literatura. Para o segundo experimento, o método foi aplicado em 15 bases de dados de classificação e obteve uma acurácia alta na maioria das bases de dados, ficando entre os melhores algoritmos testados.

No trabalho de Kumaran e Sasikala (2017) foi apresentado um modelo de rede neural baseado no algoritmo *Dragonfly Optimization* (DFO) para predizer a demanda de combustível na Índia. O método proposto utiliza duas redes MLP treinadas pelo algoritmo DFO, visando otimizar os pesos minimizando o valor de MSE. A primeira rede prediz a população e o PIB por

capita dado um ano no futuro e a segunda rede prediz a demanda futura por carvão, lignito, óleo cru e gás natural considerando a saída da primeira rede como entrada. Utilizou-se como entrada uma base contendo dados de demanda de combustível na Índia de 1980 até 2012 e o método foi comparado com um modelo de regressão. Foram feitas previsões dos anos 2013 a 2025 e os resultados mostraram que o modelo proposto obteve um valor de médio de Porcentagem do erro médio absoluto ou *Mean Absolute Percentage Error* (MAPE) de 1.8508, sendo menor que o valor de 6.3208 do modelo de regressão, o que indica uma previsão mais robusta e precisa da rede neural treinada pelo algoritmo DFO.

Um algoritmo de treinamento de rede neural, baseado na técnica *Whale Optimization Algorithm* (WOA), foi proposto por Aljarah, Faris e Mirjalili (2018). O problema de treinar uma MLP foi modelado como um problema de minimização, onde o objetivo é minimizar o valor de MSE da saída da rede e os parâmetros são os pesos da rede, sendo aplicado o WOA para resolver esse problema. O método foi testado em 20 bases de dados de classificação e comparado com outros 7 algoritmos de treinamento, sendo eles o BP, GA, PSO, ACO, DE, *Evolution Strategy* (ES) e *Population-based incremental learning* (PBIL). Os resultados mostraram que o WOA mostrou-se superior aos outros métodos na maioria das bases de dados, tanto em questão de acurácia quanto em tempo de convergência. Os autores apontam que o algoritmo foi superior graças à sua alta capacidade de exploração e evasão de ótimos locais.

Uma versão aprimorada do algoritmo *Antlion Optimizer* (ALO), chamada de *Improved Antlion Optimizer* (IALO), foi proposta por Hu et al. (2019). No primeiro passo as formigas e as *antlions* são inicializadas aleatoriamente, e as suas *fitness* são calculadas. A melhor *antlion* é considerada como a elite. Para cada formiga, é selecionada uma *antlion* pelo método da roleta e então atualiza-se a posição das formigas. Depois disso, os novos valores de *fitness* das formigas são calculados e os *antlions* são substituídos pelas suas formigas correspondentes. Se um *antlion* obter um *fitness* melhor que a elite, ele se torna a nova elite. Foram feitos dois experimentos para avaliar a performance do método. O primeiro avaliou a otimização do algoritmo em 23 funções clássicas, mostrando que o algoritmo IALO foi superior aos algoritmos Algoritmo Seno-Cosseno ou *Sine Cosine Algorithm* (SCA), PSO, MFO, *Multi-verse Optimizer* (MVO) e ALO. No segundo experimento, o algoritmo IALO foi aplicado para otimizar os pesos de uma rede neural para predizer a incidência de *influenza* e o número de pacientes de *influenza*. Para isso, foi construído o modelo IALO-*Backpropagation Neural Network* (BPNN) e comparado com os modelos BPNN, SCA-BPNN, MFO-BPNN, MVO-BPNN e ALO-BPNN. Os resultados mostraram que o método obteve os menores valores de Erro médio absoluto ou *Mean Absolute Error* (MAE), MSE, RMSE e MAPE para a predição da incidência de *influenza* e os menores valores de RMSE e MAPE para a predição de número de pacientes com *influenza*.

No estudo de Moayedi et al. (2019) foram aplicados os algoritmos DFO, WOA e *Invasive Weed Optimization* (IWO) para o treinamento de uma rede neural para predição da resistência ao cisalhamento do solo. Após a leitura de uma base de dados com fatores de resistência ao cisalhamento, foram criados e otimizados os modelos DFO-*Artificial Neural Network* (ANN),

WOA-ANN e IWO-ANN. Os resultados confirmaram a efetividade dos três algoritmos, reduzindo o erro do treinamento em 17%, 27% e 32% para o DFO, WOA e IWO respectivamente, comparados com a rede MLP treinada pela técnica BP. Para a fase de testes, os algoritmos IWO e DFO obtiveram acurácias próximas, porém, o IWO-ANN mostrou-se superior aos outros métodos.

No trabalho de Sadowski et al. (2019) foi desenvolvido um método NI meta-heurístico para predizer a tensão de fluência de *Ground Granulated Blast Furnace Slag* (GGBFS) utilizando um modelo de rede neural. Foi utilizado o FFA para otimizar os pesos da rede neural, que recebe como entrada as características do concreto e da sua produção e retorna como saída a tensão de fluência do concreto GGBFS. A rede neural otimizada pelo algoritmo FFA, chamada de FFA-ANN foi comparada com os algoritmos GA, *Imperialist Competitive Algorithm* (ICA) e PSO. Os resultados indicaram que a FFA-ANN conseguiu predizer os valores de tensão de fluência com uma alta precisão, obtendo um valor de R^2 de 0.99, sendo superior aos outros algoritmos comparados.

No trabalho de Rao e Reddy (2018) foram introduzidas novas técnicas de redução de características e classificação na área de recuperação de imagens. São utilizadas as técnicas *Gray-Level Co-Occurrence Matrix* (GLCM) e *multi-texton* para extrair características visuais da imagem. Para a textura, utilizam-se detectores de canto e de borda, SIFT e o método *Fuzzy C-Means* (FCM). Também são utilizadas as técnicas *Active Appearance Model* e *Shape Surface Plot* para extrair características do formato. Em seguida, foi utilizada a técnica OLPP para reduzir as características que foram utilizadas como entrada em uma rede neural de classificação treinada pelo FFA. Após a classificação, a imagem foi recuperada utilizando distância euclidiana para encontrar a imagem mais parecida com a classificada. O método foi testado em uma base de radiografia em três contextos diferentes. As métricas utilizadas foram a acurácia, *recall*, precisão e *F-measure*. Os resultados demonstraram que o método proposto obteve uma acurácia alta e a recuperação da imagem foi precisa. Também foi mostrado que a rede neural treinada pelo FFA aprimorou a classificação comparada com outros classificadores.

Bosire (2019) avaliou o uso do algoritmo ABC para otimizar redes neurais recorrentes aplicadas na análise de volume de tráfego de veículos. Os pesos da rede foram otimizados pelo algoritmo ABC de forma que minimizasse o valor de MSE das saídas da rede. O método proposto foi testado em uma base com informações de tráfegos de estradas. Os valores ótimos de MSE obtidos foram de 359.8409 para a fase de treinamento, 569.0172 para a fase de validação e 673.4512 para a fase de teste. Os autores concluíram que a técnica superou o método BP, no entanto, as configurações de parâmetros do algoritmo precisam ser refinadas para ele poder ser generalizado.

Foi apresentada uma abordagem de treinamento de rede neural utilizando 4 técnicas de otimização diferentes por AM Hemeida et al. (2020). Foram utilizados os algoritmos WOA, DFO, MVO e GWO para otimizar os pesos de uma rede neural para a classificação das bases de dados *Iris* e *Breast Cancer*, utilizando o MSE como função objetivo. Os resultados mostraram

que, em geral, os algoritmos GWO e MVO foram os melhores algoritmos em questão do tempo de execução, convergência, taxa de classificação e MSE.

No trabalho de Sheta, Braik e Al-Hiary (2019) foram utilizados algoritmos bio-inspirados para o treinamento de redes neurais aplicadas na área de modelagem de processos químicos. Foi considerado o problema do reator químico de *Tennessee Eastman* (TE) como estudo de caso. A chave do problema é encontrar uma arquitetura de rede neural que melhor modele os processos químicos do TE. Para isso, foi proposto o uso dos algoritmos BAT, FFA e ABC para atualizar os pesos sinápticos da rede neural automaticamente, utilizando o MSE como função objetivo. Os resultados mostraram que os modelos desenvolvidos demonstraram uma alta acurácia para modelar o reator TE.

Uma variação do algoritmo PSO, chamada de *Torus* (TO)-PSO, foi proposta no trabalho de Ashraf et al. (2020) e aplicada para o treinamento de redes neurais *feed-forward*. A técnica inicializa o PSO baseando-se em uma sequência semi-aleatória, distribuindo as partículas do enxame de forma eficiente no espaço de busca do problema para evitar problemas de convergência prematura. A abordagem proposta foi testada em 9 bases de dados de classificação diferentes. Os resultados empíricos demonstraram a superioridade da técnica TO-PSO quando utilizada para o treinamento de redes *feed-forward*, mantendo a diversidade do espaço de busca e evitando ótimos locais durante o treinamento, resultando em classificações mais precisas comparadas com outros métodos de inicialização.

O algoritmo *Emperor Penguins Colony* (EPC) foi aplicado no trabalho deHarifi et al. (2020) para otimizar um *Adaptive Neuro-Fuzzy Inference System* (ANFIS). O método recebe a base de treinamento e cria um sistema *fuzzy* básico a partir dela. Em seguida, os parâmetros do sistema *fuzzy* foram otimizados de forma que minimizem o valor de RMSE pelo algoritmo EPC para obter o sistema *fuzzy* com os melhores parâmetros. O método foi comparado com outros 7 métodos, sendo testado em 7 bases de dados diferentes. Os autores afirmaram que o método proposto obteve uma taxa de erro menor e melhor desempenho em comparação com outros algoritmos estado-da-arte.

2.3 ALGORITMOS DE OTIMIZAÇÃO HÍBRIDOS

Com a aplicação de técnicas de otimização meta-heurística NI para o treinamento de redes neurais, uma nova abordagem híbrida, que combina algoritmos de otimização NI com técnicas de treinamento tradicionais, como o BP, começou a ser estudada na literatura, como apresentada por Kim, Seo e Kang (2005). Outro algoritmo híbrido para o treinamento de redes neurais *feed-forward* foi proposto por Jing-Ru Zhang et al. (2007). O algoritmo combina as técnicas PSO e BP, sendo chamado de PSO-BP. Além de algoritmos híbridos que combinam técnicas de treinamento convencionais com algoritmos NI, também existem trabalhos que exploraram a combinação de dois ou mais algoritmos NI, como no trabalho de Chen et al. (2007),

que propôs a combinação das técnicas PSO e *Artificial Fish Swarm Algorithm* (AFSA) para o treinamento de redes neurais.

No trabalho de Beheshti et al. (2014) foi usado o algoritmo *Centripetal Accelerated Particle Swarm Optimization* (CAPSO), um algoritmo NI híbrido que se inspira em conceitos da física, combinados com o PSO tradicional, para evoluir o treinamento e acurácia de uma rede MLP aplicada em classificação de diagnóstico médico. O método foi testado em 9 bases de diagnóstico médico e os resultados mostraram que o CAPSO produz resultados melhores comparados ao PSO, *Gravitational Search Algorithm* (GSA) e ICA para treinar uma rede neural, em termos de taxa de convergência e acurácia. O autor aponta que as vantagens do algoritmo é a sua facilidade de implementação e a quantidade pequena de parâmetros que precisam ser variados.

Os autores Shuihua Wang et al. (2015) propuseram um método para classificação de frutas utilizando técnicas de aprendizado de máquina. Foi utilizado um descritor *wavelet-entropy* para extrair características de imagens de frutas coloridas. Em seguida, foi aplicado o PCA para reduzir a dimensionalidade dessas características e então utilizar esses dados como entrada de uma rede neural *feed-forward*. Foram propostas duas alternativas para o treinamento da rede, sendo a primeira dela com o uso do algoritmo *Fitness-Scaled Chaotic Artificial Bee Colony* (FSCABC) e a segunda com o algoritmo BBO, ambos utilizando como função objetivo a minimização do MSE de saída obtida pela rede. Foram utilizadas 1653 imagens de frutas, com 18 classes no total, para testar os métodos. A acurácia obtida por ambos os métodos propostos foram de 89.5%, ficando acima de outras técnicas testadas, mostrando-se efetivo para a classificação de frutas.

No trabalho de Leema, Nehemiah e Kannan (2016) foi proposto um sistema *Computer-Aided Diagnostic* (CAD) que utiliza uma rede neural treinada pelos algoritmos DE, PSO e BP, chamado de DEGI-BP, para a classificação de bases de dados clínicas. O algoritmo DE, com uma modificação no seu operador de mutação, foi utilizado para aprimorar a exploração do PSO. Em seguida, o PSO foi utilizado para treinar uma rede neural e o melhor valor global obtido foi adotado como semente do BP. A busca local foi feita utilizando o BP, onde os pesos da rede foram ajustados para encontrar os pesos ótimos que minimizem o erro da saída da rede. O método foi testado nas bases *Pima Indian Diabetes*, *Wisconsin Breast Cancer* e *Cleveland Heart Disease*. Os resultados de acurácia obtidos foram de 85.71%, 98.52% e 86.66% para as três bases respectivamente.

Um algoritmo híbrido, chamado de *Hybrid Particle-move Artificial Bee Colony* (HPABC), foi utilizado para o treinamento de redes neurais *feed-forward* no trabalho de Al Nuaimi e Abdullah (2017). O HPABC combina os algoritmos ABC e PSO, sendo aplicado para ajustar os pesos para aprimorar a velocidade e a performance da rede neural, utilizando como função objetivo a minimização do valor de SSE de saída da rede de acordo com os pesos ajustados. A performance do algoritmo foi investigada em 4 bases de dados para classificação, sendo comparado com outros algoritmos recentes e obtendo uma acurácia de 99.33% para a base *Iris*, 99.9%

para a base de dados *Câncer*, 99.63% para a base de dados *Diabetes* e 77.74% para a base *Glass*, sendo a acurácia mais alta entre todos os algoritmos testados.

Também foi utilizado um algoritmo de otimização NI híbrido no trabalho de Rani R e Victoire T (2018), aplicando-o para o treinamento de uma RBFNN. Foi feita uma integração do algoritmo de treinamento *Differential Search* (DS) com o algoritmo PSO, sendo o algoritmo DS o principal otimizador e o PSO foi utilizado para explorar a vizinhança das soluções encontradas pelo DS. Para superar o problema de ficar preso em soluções ótimas locais, foi proposta uma abordagem de inicialização de população utilizando sequência lógica caótica, que melhora a diversidade da população e a sua capacidade de busca. A técnica proposta foi testada em 7 bases de dados diferentes, demonstrando a sua superioridade de tempo de convergência e baixa taxa de erro em relação aos algoritmos PSO e DS aplicados isoladamente. O algoritmo também foi testado em uma aplicação prática para a predição da velocidade de vento, sendo comparado com outras 3 técnicas de treinamento de uma RBFNN, demonstrando a maior acurácia entre todas as técnicas testadas.

LAKSHMI (2018) utilizaram algoritmos NI para a otimização da arquitetura de uma rede neural aplicada em reconhecimento de voz. Foram utilizadas as técnicas EA, GA e PSO para encontrar o número de camadas e o número de neurônios por camada que fornecem a saída ótima do problema. Os resultados demonstraram que as redes otimizadas pelos algoritmos NI produzem uma taxa de erro menor se comparado com a rede neural tradicional, com o algoritmo PSO obtendo os melhores resultados entre os 3 algoritmos testados.

No trabalho de Bui (2019) foram utilizados os algoritmos BBO, GSA e GWO para o treinamento de uma rede neural para mapear queimadas de florestas. Foram utilizadas 1338 localizações com histórico de queimadas como variáveis dependentes e informações topológicas, climáticas e socio-econômicas foram usadas como variáveis preditoras independentes. Para a função objetivo, adotou-se o RMSE da saída da rede neural para ser otimizada pelos 3 algoritmos. Os resultados de *Area Under the Curve* (AUC) obtidos foram de 0.9515 pelo BBO, 0.9509 do GWO e 0.9398 do GSA, superando o valor de AUC de 0.9271 do método BP. Os autores concluíram que os três modelos híbridos são viáveis otimizar uma rede aplicada no mapeamento de queimadas de florestas.

Os autores Naganna et al. (2019) introduziram duas abordagens híbridas de MLP, chamadas de MLP-GSA e MLP-FFA, para estimar de forma eficiente o ponto de orvalho de regiões semiáridas e úmidas da Índia. Foram utilizados os algoritmos FFA e GSA para otimizar o número de neurônios da camada escondida e a taxa de aprendizado da rede, enquanto o treinamento da rede foi feito utilizando o algoritmo *Levenberg-Marquardt* (LM)-BP. Medidas diárias a respeito das informações climáticas foram utilizadas como base de entrada da rede. Os dois modelos híbridos foram comparados com as técnicas *Support Vector Machine* (SVM) e ELM. Os 3 índices de avaliação indicaram que a técnica MLP e as suas variações híbridas são superiores aos modelos SVM e ELM. Foi concluído que os algoritmos de otimização NI aprimoram a precisão do algoritmo MLP clássico, com as redes MLP-FFA e MLP-GSA obtendo uma acu-

rácia aceitável para ambas as zonas climáticas estudadas, sendo que o MLP-FFA conseguiu os melhores resultados em critérios de minimização de erro e tempo de convergência.

No trabalho de Qiao, Moayedi e Foong (2020) foram utilizado os algoritmos IWO, DFO, ES, GA e ICA no treinamento de uma rede MLP para estimar o consumo de gás natural mensal. Para isso, foi utilizada uma base de dados com 8 características e 39 amostras, sendo dividida em $\frac{2}{3}$ para treino e $\frac{1}{3}$ para teste aleatoriamente. Os parâmetros ótimos das redes MLP, IWO-MLP, DFO-MLP, ES-MLP, GA-MLP e ICA-MLP foram encontrados por tentativa e erro. Para estimar o erro das técnicas, foram consideradas as métricas MSE, RMSE e R^2. Os resultados mostraram que a rede IWO-MLP consegue ser a melhor rede comparada com outras redes híbridas, obtendo um valor de MSE de 0.000013, 0.99 de R^2 e 0.00002 de RMSE.

2.4 ALGORITMOS DE OTIMIZAÇÃO BASEADOS EM CONCEITOS FÍSICOS/QUÍMICOS

Os algoritmos NI de otimização dessa categoria buscam por soluções imitando comportamentos de fenômenos físicos ou químicos, como conceitos de gravidade, cargas elétricas e eletromagnetismo em abordagens baseadas em física, e reações químicas e movimento de gases em algoritmos de otimização inspirados na química.

Os autores Saha, Chakraborty e Dutta (2014) realizaram um estudo investigativo para aprimorar e aplicar o algoritmo GSA, um algoritmo NI baseado em conceitos de leis da gravidade de Newton, para treinar uma rede neural *feed-forward*. Para testar o método, foram utilizadas 5 bases de classificação. Os resultados mostraram que o método conseguiu superar o GSA tradicional, obtendo uma acurácia superior para a classificação em todas as bases testadas.

No estudo de Lee, Geem e Suh (2016) foi desenvolvido um modelo de rede neural artificial para predizer a estabilidade do número de pedras de um quebra-mar. Para isso, foi utilizado o algoritmo HS para encontrar os pesos ótimos iniciais que foram utilizados para realizar o treinamento da rede pelo algoritmo LM. O método foi comparado com uma rede neural com os pesos inicializados pelo método Monte Carlo e a análise dos resultados mostrou que, com os parâmetros adequados, o algoritmo consegue fornecer previsões melhores e mais estáveis comparadas com as obtidas por redes neurais convencionais.

No trabalho de Hossam Faris et al. (2016b) foi investigada a eficiência do algoritmo *Lightining Search Algorithm* (LSA), um algoritmo de otimização baseado no fenômeno natural de iluminação, para o treinamento de uma rede neural foi investigada por. O LSA foi utilizado para otimizar uma rede neural com uma camada escondida. Cada indivíduo do LSA foi codificado como uma lista de números reais com números aleatórios gerados de -1 a 1, representando os pesos das conexões da rede. Para a função objetivo, foi considerada a minimização do MSE até um determinado número máximo de iterações ser alcançado. Foram utilizadas 16 bases de dados para testar o algoritmo, sendo comparado com outros 8 métodos. Os resultados quantitativos demonstraram que o algoritmo LSA produziu os melhores resultados na maioria das bases. Com os resultados obtidos, os autores concluíram que o algoritmo LSA supera algoritmos apli-

cados ao treinamento de redes neurais com diferentes estruturas utilizadas para a classificação de diversas bases de dados, mostrando uma rápida convergência e se beneficiando da sua grande exploração do espaço de busca e evasão de ótimos locais.

Os autores Faris, Aljarah e Mirjalili (2016) utilizaram o algoritmo MVO, um algoritmo NI que se baseia em conceitos cosmológicos de interação entre universos a partir de portais, para treinar uma rede neural MLP com uma camada escondida. Cada universo do MVO foi definido por 3 partes: os pesos das conexões da camada de entrada e da camada escondida; os pesos das conexões entre a camada escondida e a camada de saída. Foi utilizada como função objetivo a minimização do valor de MSE obtido pela predição da rede. O método foi testado em 9 bases de dados do repositório UCI e comparado com os algoritmos BP, LM, GA, PSO, DE, FFA e *Cuckoo Search* (CS). O algoritmo MVO obteve a maior acurácia para as bases *Blood*, *Breast cancer*, *Diabete*, *Hepatitis*, *Liver* e *Parkinson*, obtendo 0.796, 0.973, 0.768, 0.894, 0.725 e 0.93 de acurácia respectivamente.

No trabalho de Nayak, Misra e Behera (2017) foi apresentada uma *Artificial Chemical Reaction Neural Network* (ACRNN), uma rede neural MLP treinada pelo algoritmo *Artificial Chemical Reaction Optimization* (ACRO) e aplicada na predição de índices do mercado de ações. O algoritmo ACRO encontrou os valores de pesos ótimos utilizando como função de avaliação a minimização do MAE. O método proposto foi testado em 7 valores do índice de ações e comparado com o MLP, RBFNN e *Multiple Linear Regression* (MLR). Os valores de acurácia e POCID obtidos pelo método mostraram que o modelo proposto possui ganhos significativos em relação a todos os modelos testados em todos os valores de índice de ações considerados.

2.5 ALGORITMOS DE OTIMIZAÇÃO BASEADOS EM COMPORTAMENTOS SOCIAIS HUMANOS

Uma categoria de algoritmos NI que tem sido bastante explorada em trabalhos mais recentes são os algoritmos inspirados em conceitos sociais humanos, como ideias relacionadas à competição ou expansão de ideologias dentro de uma sociedade, ou conceitos políticos como o método ICA, que se baseia na ideia de impérios e colônias para otimizar um determinado problema.

O algoritmo ICA foi aplicado no trabalho de Hajihassani et al. (2015) para a predição de *Peak Particle Velocity* (PPV) resultantes da explosão de pedreiras por meio da otimização de uma rede neural. Para isso, foram monitorados 95 trabalhos de explosões de pedreiras de granito na Malásia, tendo os seus valores de PPV registrados precisamente em cada uma das operações. Utilizaram-se as características mais influentes como entrada de uma rede neural treinada pelo algoritmo ICA, que determinou os pesos ótimos de uma rede MLP com uma única camada escondida, considerando a minimização do valor de RMSE como função objetivo. O modelo proposto foi comparado com outras técnicas de regressão e mostrou-se ser superior à todas elas, predizendo o valor de PPV com uma acurácia alta, obtendo um valor de R^2 de 0.976.

O trabalho de Cao et al. (2015) foi outro exemplo onde um algoritmo NI de otimização inspirado em comportamentos sociais humanos foi utilizado para treinar uma rede neural. Foi proposta uma melhoria do algoritmo *Brain Storm Optimization* (BSO), chamada de BSO-DE, sendo utilizado o algoritmo DE para a criação de um operador de ideias que auxilia o BSO a evitar ótimos locais. Também foi utilizado um método para controlar os passos das novas soluções para melhorar o balanço entre a busca abrangente e a exploração das soluções que já foram encontradas até o momento. A técnica foi aplicada para realizar o treinamento de uma rede neural, otimizando os seus pesos visando minimizar o MSE. O método foi aplicado na predição da série temporal caótica de Lorenz e para modelar uma função não-linear, sendo comparado com outros 5 métodos do estado da arte. Os resultados mostraram que o algoritmo BSO-DE obteve o melhor desempenho na predição da série sem nenhum ruído e foi melhor que os algoritmos BSO e DE isolados para aproximar funções não-lineares.

Os autores Qingyang Zhang et al. (2017) propuseram uma nova técnica meta-heurística NI, baseada em comportamentos sociais humanos, aplicada para o treinamento de redes neurais artificiais. A técnica se chama *Collective Decision Optimization Algorithm* (CDOA) e simula o comportamento social humano baseando-se em características de tomada de decisão em reuniões de grupo que inclui 5 fases, sendo elas: baseada em experiência; baseada na decisão de outros agentes; baseada no pensamento coletivo; baseada em liderança e baseada em inovação. Para esse algoritmo, considera-se a população do algoritmo como o grupo de reunião, os agentes da população como os decisores da reunião, as soluções como os planos do decisor, a função objetivo a qualidade do plano e a solução global como o melhor plano. Ele foi aplicado para otimizar os pesos de uma rede neural, utilizando como função objetivo minimizar o valor de MSE da saída da rede. Foram realizados dois testes. O primeiro teste avaliou a taxa de convergência em 21 funções diferentes, comparando com outros algoritmos de otimização. O segundo experimento avaliou o desempenho do algoritmo para treinamento de uma rede neural *feed-forward* para encontrar duas funções matemáticas não-lineares. Os resultados demonstraram que a técnica proposta executa o treinamento de forma mais precisa e efetiva comparada com outros métodos de otimização atuais da literatura na época.

2.6 ARTIGOS DE REVISÃO

O autor J.Net e J.N (2016) revisou os aprimoramentos de uma rede neural treinada por um algoritmo de otimização NI e aplicada em diversos domínios. Para isso, foram apresentadas diversas técnicas de otimização e separadas por categorias, sendo considerado algoritmos de *swarm intelligence*, bio-inspirado, baseado em processos químicos e físicos, híbridos e outros que não se encaixaram em nenhuma das outras categorias. Em seguida, foram descritos alguns trabalhos que utilizaram essas técnicas. Como conclusão, o autor destacou alguns pontos que devem ser considerados futuramente, sendo os principais deles: (i) a exploração de técnicas de otimização NI em arquiteturas de redes diferentes da *feed-forward*; (ii) a investigação da

aplicabilidade de técnicas meta-heurísticas recentes; (iii) avaliação das vantagens das diferentes categorias das técnicas meta-heurísticas; (iv) explorar a execução paralela desses algoritmos para o treinamento de redes neurais; (v) estudo das vantagens dos algoritmos meta-heurísticos híbridos para otimizar os parâmetros de uma rede neural.

Uma revisão de alguns trabalhos relevantes que aplicaram algoritmos meta-heurísticos na otimização de redes neurais também foi feita no trabalho de Tian e Fong (2016). Os autores assumiram que existe uma grande possibilidade de aplicar algoritmos meta-heurísticos no treinamento de Rede neural profunda ou *Deep Neural Network* (DNN) para agilizar o treinamento sem diminuir o seu desempenho, por causa do seu processo de treinamento ser semelhante ao de redes neurais tradicionais. No entanto, mostraram-se raras as publicações relevantes que apontam para essa direção.

Os autores Ojha, Abraham e Snášel (2017) buscaram sumarizar um espectro amplo das metodologias de otimização de redes neurais, incluindo abordagens convencionais e meta-heurísticas. A revisão feita abordou a importância de redes neurais como aproximadores de funções e seus conceitos preliminares, o papel de algoritmos meta-heurísticos na otimização de redes neurais, o papel de algoritmos meta-heurísticos multi-objetivos e métodos híbridos. Os autores apontaram que os algoritmos de treinamento de redes neurais convencionais, como o BP, são algoritmos de buscas locais que geram novas soluções a partir da solução atual, não possuindo habilidades de exploração e geralmente encontrando ótimos locais para a solução do problema. Sobre os algoritmos meta-heurísticos, como o GA, PSO, ACO, entre outros, foi dito que eles possuem vantagens em sua capacidade de exploração e se adaptam simultaneamente em cada componente da rede neural. No entanto, os autores disseram que não existe um único método que consiga resolver todos os problemas. Por isso, foi destacado a importância da construção de métodos híbridos para generalizar a otimização das redes neurais.

Os autores Devikanniga, Vetrivel e Badrinath (2019) realizaram uma revisão sobre redes neurais híbridas, treinadas por um algoritmo de otimização meta-heurístico, e suas aplicações para classificação de bases de dados de avaliação ou predição de problemas reais. Os autores apresentaram os algoritmos de otimização que são aplicados em *benchmarks* e os algoritmos que são aplicados em problemas reais, dando uma breve explicação de como cada um deles funciona. Foi concluído que para obter soluções ótimas para o problema, o ideal seria combinar técnicas de otimização para gerar algoritmos de otimização híbridos, melhorando a forma como é explorado o espaço de busca da solução.

Os autores AM Hemeida et al. (2020) realizaram uma revisão dos principais algoritmos NI para otimização de redes neurais *feed-forward*, destacando os seus modelos, características, objetivos e restrições para analisar as diferenças e semelhanças entre eles. Sobre os principais algoritmos encontrados na literatura, os autores observaram os seguintes pontos: (i) o algoritmo ACO é viável na otimização dos pesos da rede; (ii) o GA pode aprimorar a velocidade computacional com buscas paralelas; (iii) o CS aprimora a velocidade e a acurácia devido à flexibilização dos seus parâmetros; (iv) o DE possui poucos parâmetros de controle; (v) o BBO é capaz de re-

solver problemas de alta dimensionalidade e evitar a estagnação em mínimos locais; (vi) o BAT é um algoritmo promissor por causa de sua implementação e das comparações feitas com eles; (vii) o PSO PSO possui um cálculo simples e uma busca rápida; (viii) o ABC é uma solução com baixo tempo computacional, devido à sua rápida taxa de convergência.

2.7 ALGORITMOS DE OTIMIZAÇÃO APLICADOS EM APRENDIZADO PROFUNDO

Com o avanço da área de aprendizado de máquina e a demanda por soluções de problemas cada vez mais complexos, um novo ramo da área, chamado aprendizado profundo, apareceu e com isso novos tipos de redes neurais e algoritmos de treinamento surgiram. Um exemplo é a Rede neural convolucional ou *Convolutional Neural Network* (CNN), que é uma classe de rede neural que ganhou bastante destaque graças aos seus resultados positivos em aplicações de processamento e análise de imagens digitais.

Seguindo por essa linha, o trabalho de Bhandare (2017) propôs automatizar o processo de seleção de arquitetura de uma CNN otimizando os seus hiper-parâmetros por meio de computação bio-inspirada. Para isso, foram utilizados os algoritmos GA, PSO e GWO para otimizar os seguintes hiper-parâmetros: n.º de épocas, tamanho do *batch* de treinamento, número de camadas convolucionais e as suas funções de ativação, número e tamanho dos filtros de cada camada convolucional, o tamanho do pool da camada de *Maxpool*, o número de camadas *feed-forward*, o número de neurônios de cada camada *feed-forward* e a sua função de ativação, e o otimizador da CNN. A função de avaliação adotada foi o número de classificações corretas. Os algoritmos foram testados para otimizar a arquitetura de uma CNN para classificar a base de dados MNIST. Os experimentos mostraram que os algoritmos GA e PSO conseguiram uma acurácia mínima maior que 90% e a maior acurácia foi de 99.2% para o GA e 99.36% para o PSO. O algoritmo GWO obteve uma acurácia mínima maior que 85% e a sua maior acurácia foi de 99.4%. Em relação ao tempo de processamento, os algoritmos GA e PSO levaram aproximadamente de 4 a 5 horas enquanto o GWO levou de 5 a 6 horas para otimizar a arquitetura da CNN.

Outro trabalho que propôs uma abordagem parecida foi o de Bin Wang et al. (2018), que utilizou o algoritmo PSO para buscar automaticamente por uma arquitetura ótima de uma CNN. Para isso, foram feitas 3 melhorias no algoritmo PSO tradicional. A primeira é a forma como as camadas da CNN são representadas, sendo adotada uma estrutura de IP onde cada sub-rede dele corresponde a uma camada da rede. A segunda é uma sub-rede adicional, chamada de subnet desabilitada, para simular a variação de tamanho do PSO. A última melhoria é o cálculo da função de avaliação utilizando a base de dados parcialmente. O método foi testado com outros 12 algoritmos existentes em 3 bases de classificação de imagens, mostrando-se páreo aos métodos comparados em termos da taxa de erro de classificação.

No trabalho de Khare et al. (2020) foi feita a combinação do algoritmo *Spider Monkey Optimization* (SMO) com uma DNN. Para isso, as bases de dados NSL-KDD e KDD Cup 99, do repositório *Kaggle*, foram utilizadas. As bases foram limpas utilizando a técnica de norma-

lização *min-max* e passada para o método de codificação 1-N para dar homogeneidade para a base. Em seguida, foi utilizado o algoritmo SMO para redução de dimensionalidade da base, que foi então utilizada como entrada no treinamento de uma rede neural profunda. O método proposto gerou resultados de classificação obtendo 99.4% e 92% de acurácia, 99.5% e 92.7% de precisão, 99.5% e 92.8% de *recall*, 99.6% e 92.7% de *F1-score*.

Um método para determinar os hiper-parâmetros de uma CNN, aplicada em uma base de dados pequena, foi introduzido por Shih et al. (2019). Para isso, foram utilizados os algoritmos HS, *Improved Harmony Search* (IHS), *Self-adaptive Global Best Harmony Search* (SGHS) e *Novel Global Harmony Search* (NGHS) para encontrar os valores ótimos de tamanho do *kernel* da camada convolucional, tamanho do *pool* da camada de *pooling* e números de neurônios da camada densa, utilizando a média da taxa de classificações erradas como função objetivo. O método foi testado para classificar a base MNIST. Os resultados mostraram que os algoritmos HS e IHS conseguem encontrar a melhor estrutura da CNN, reduzindo a taxa de erro da classificação.

No trabalho de Dixit et al. (2019) foi apresentada uma técnica de aprendizado profundo para reconhecimento de textura utilizando uma CNN otimizada pelo algoritmo WOA. O algoritmo WOA foi aplicado na camada convolucional para otimizar os valores dos filtros e na camada densa (totalmente conectada) para otimizar os pesos. O método foi testado em 3 bases de dados diferentes, obtendo resultados superiores aos das técnicas existentes para as bases de dados *Kylberg* (99.7%) e *Outex* (97.7%), e resultados próximos do estado da arte na base *Brodatz* (97.43%).

No trabalho de Serizawa e Fujita (2020) foi proposta a otimização de parâmetros de uma CNN utilizando o algoritmo *Linearly Decreasing Weight Particle Swarm* (LDWPSO) . Foram utilizadas as bases de dados MNIST e CIFAR-10. O algoritmo LDWPSO foi utilizado para encontrar o número de filtros, o tamanho do *kernel*, a função de ativação e o número de neurônios de cada camada de uma LeNet-5. O método conseguiu uma acurácia de 98.95% para a base de dados MNIST e 69.37% na base CIFAR-10. Com esses resultados, os autores concluíram que a técnica LDWPSO consegue encontrar os parâmetros de uma CNN de forma que ela fornece classificações com uma acurácia alta.

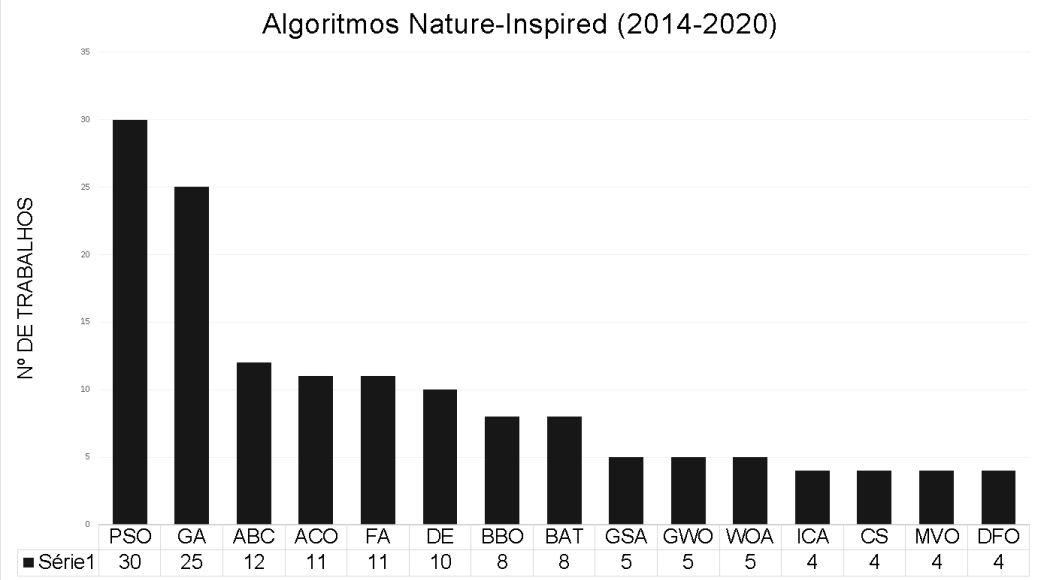
2.8 ANÁLISE DA BIBLIOGRAFIA

Feita essa pesquisa, levando em conta estudos desde os primeiros indícios da otimização de redes neurais por algoritmos NI, até trabalhos publicados atualmente, foram extraídos diversos dados a respeito dos artigos lidos para analisar tendências da área.

A primeira informação analisada é o número de ocorrências dos principais algoritmos de otimização que foram utilizados por trabalhos que foram considerados na pesquisa, dentro do período do ano de 2014 à 2020. Para isso, foi gerado um gráfico de barras, apresentado na Figura 1, onde cada coluna representa o número total de trabalhos que utilizaram um determinado algoritmo. Ao observar o gráfico, nota-se que os principais algoritmos mais utilizados são o

PSO e o GA, dois algoritmos consolidados na literatura e os mais antigos dentre os algoritmos mais utilizados encontrados na revisão feita. Outro aspecto notado é uma aparição maior do FFA em relação ao GSA, sendo o primeiro baseado em inteligência de enxame e o segundo em conceitos físicos newtonianos, ambos desenvolvidos no mesmo ano, sugerindo uma preferência por algoritmos do primeiro tipo somado ao fato da maioria dos algoritmos utilizados serem dessa categoria.

Figura 1 – Gráfico que apresenta a ocorrência dos principais algoritmos NI nos trabalhos pesquisados.



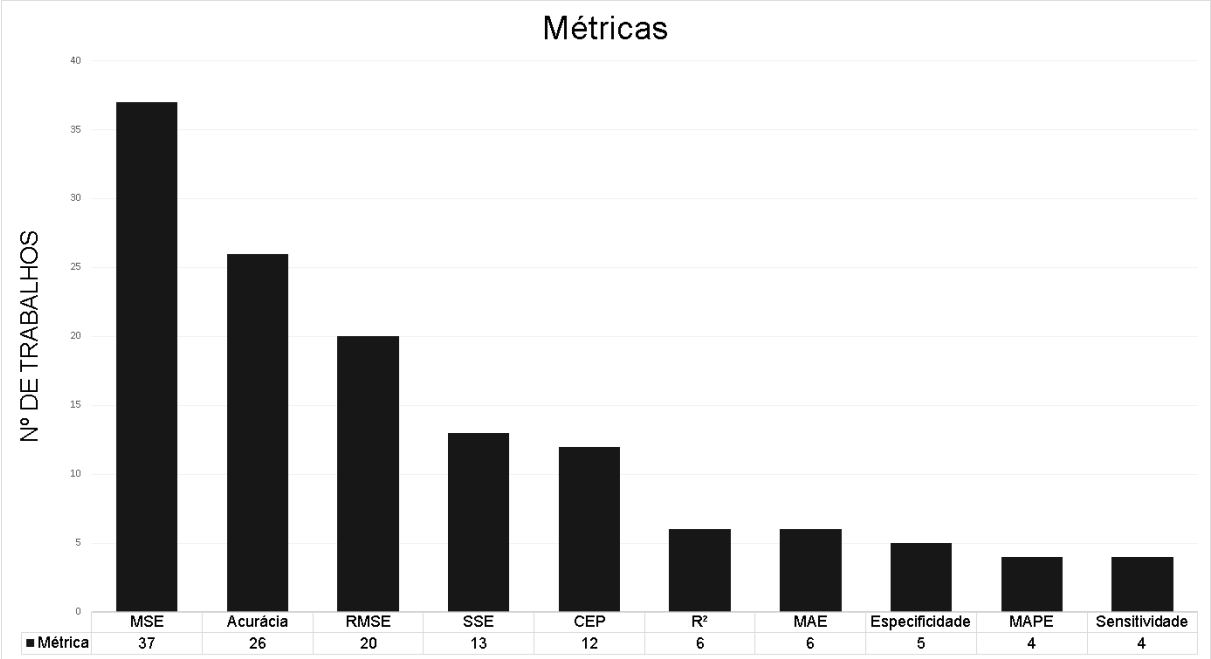
Fonte: Autor.

Em segundo lugar, foram analisadas as métricas mais utilizadas por esses trabalhos a partir de um gráfico do mesmo tipo do apresentado na Figura 1, foi gerado para facilitar essa análise, apresentado na Figura 2. O principal ponto a comentar deste gráfico é em relação às 5 métricas mais utilizadas, que foram utilizadas tanto para avaliar e comparar os resultados dos métodos quanto como função objetivo adota para ser otimizada pelos algoritmos NI utilizados, sendo amplamente utilizadas na avaliação de problemas de classificação e calculadas facilmente. Esses fatores justificam a popularidade dessas métricas nos artigos estudados.

Outra informação analisada foi sobre as bases de dados utilizadas pelos trabalhos descritos. Esses dados são apresentados no gráfico da Figura 3, seguindo o mesmo formato dos gráficos anteriores. As bases mais utilizadas nos trabalhos vistos são bases de dados clássicas de classificação, adotadas há bastante tempo por diversos trabalhos que abordaram problemas de classificação de dados, visto que as mais antigas são as que mais aparecem. A fácil disponibilidade dessas bases também é um dos motivos delas aparecerem na maioria dos trabalhos, estando disponíveis em [1].

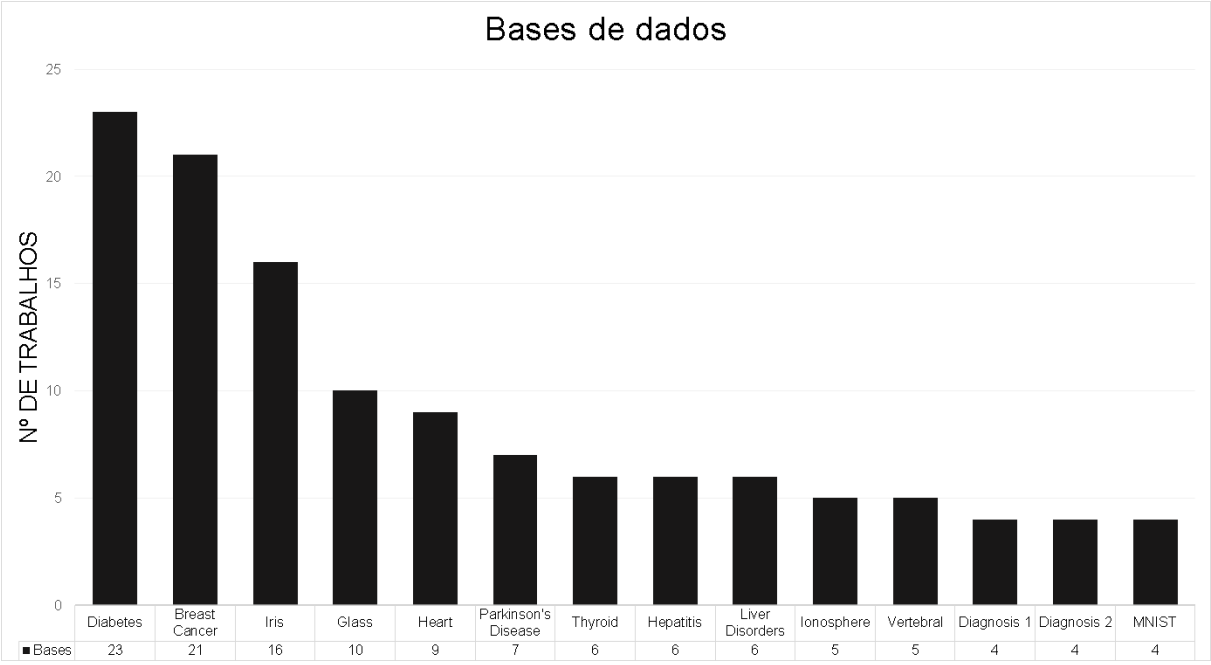
[1] https://archive.ics.uci.edu/ml/index.php

Figura 2 – Gráfico que apresenta a ocorrência das principais métricas adotadas nos trabalhos pesquisados.



Fonte: Autor.

Figura 3 – Gráfico que apresenta a preferência das principais bases utilizadas nos trabalhos pesquisados.

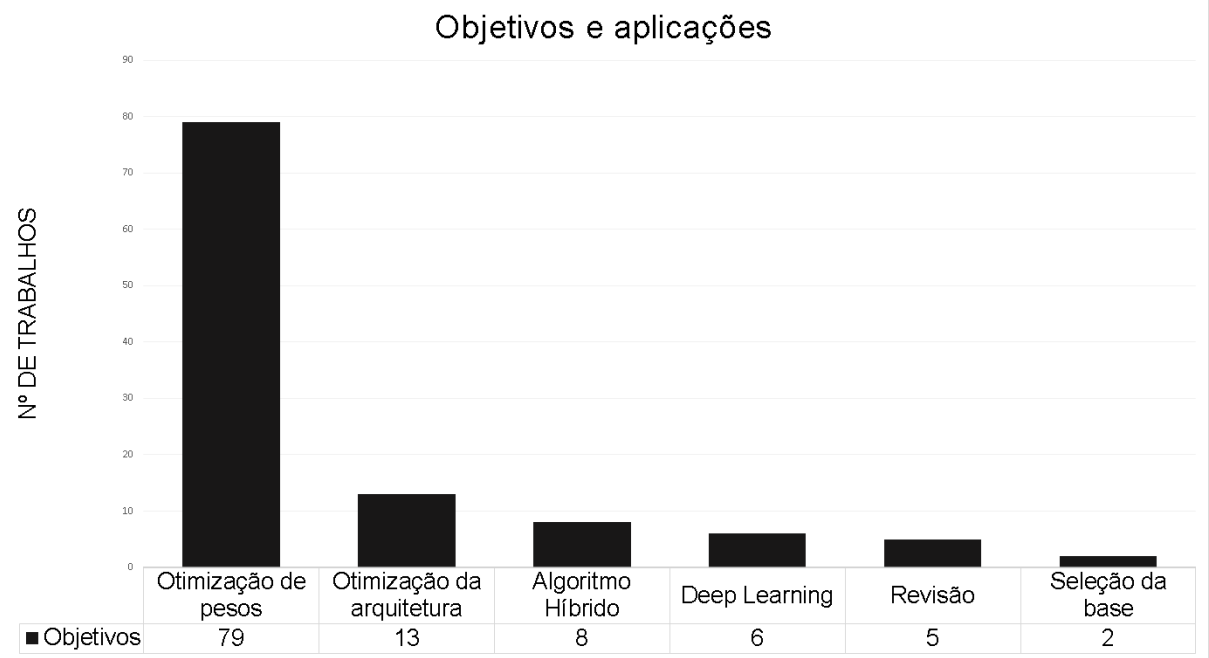


Fonte: Autor.

As últimas características analisadas foram os objetivos da aplicação do algoritmo NI, sendo considerado os seguintes: (i) otimização dos pesos de uma rede neural; (ii) otimização

da arquitetura de uma rede neural; (iii) utilização de algoritmos híbridos que combinam 2 ou mais algoritmos NI; (iv) utilização de algoritmos NI em conjunto com aprendizado profundo; (v) seleção otimizada da base de dados de entrada. Levando isso em conta, foi gerado um gráfico para visualizar a relação dos objetivos considerados nos trabalhos lidos, apresentado na Figura 4. O maior destaque do gráfico é o fato da maioria dos trabalhos terem aplicado os algoritmos NI para otimizarem os pesos de redes neurais, mesmo considerando que os primeiros trabalhos que buscaram otimizar a arquitetura da rede surgiram na mesma época, mostrando que a literatura indica uma necessidade maior da busca dos pesos ótimos ao invés da arquitetura ótima. Por outro lado, nos trabalhos que utilizam técnicas de aprendizado profundo, nota-se uma necessidade maior da otimização da arquitetura da rede devido à utilização de redes neurais mais complexas.

Figura 4 – Gráfico que apresenta a relação dos objetivos considerados nos trabalhos pesquisados.



Fonte: Autor.

2.9 CONCLUSÃO DA REVISÃO BIBLIOGRÁFICA

A partir da leitura dos trabalhos relacionados à aplicação de algoritmos de otimização para treinar RNAs e da análise feita na Seção 2.8, foi possível destacar alguns pontos importantes.

Em primeiro lugar, por mais que o tema pesquisado tenha surgido a mais de três décadas, em 1989, ele ainda é amplamente explorado na literatura, sendo aplicado para resolver problemas diversificados e alcançando ótimos resultados. Além disso, o surgimento de novos algoritmos

de otimização NI ao longo dos anos ajuda com que o tema se mantenha recente, podendo utilizar algoritmos atuais que renovam a área.

Outro ponto observado foi a diversidade dos algoritmos NI utilizados, sendo encontrados dezenas de algoritmos diferentes entre os artigos lidos, desde os primeiros algoritmos NI, até algoritmos mais atuais. Porém, os algoritmos baseado em inteligência de enxame, especificamente aqueles que se inspiram em comportamentos observados no reino animal, são os que aparecem com mais frequência entre os trabalhos, o que indica um interesse maior dos pesquisadores em explorar esse tipo de algoritmo.

Também foi notada a falta de uma análise mais profunda do impacto que as diferentes métricas de avaliação e funções de ativação causam no desempenho de busca da solução ótima e tempo de convergência dos algoritmos NI, mostrando um ponto que pode ser investigado melhor nessa área, podendo ser comparadas as diversas combinações entre funções de ativação e métricas de avaliação para treinar a rede neural.

Por fim, pode-se concluir que o treinamento de redes neurais artificiais por algoritmos de otimização NI é um tópico que tem despertado o interesse dos pesquisadores, sendo aplicado em contextos diferentes nos últimos anos, além de estar sempre se atualizando com o desenvolvimento de novos algoritmos NI, o que indica ser uma área promissora a ser estudada, por ainda ter questões em aberto que podem agregar mais para o seu aprimoramento.

3 CONCEITOS FUNDAMENTAIS

Neste capítulo são apresentados os conceitos fundamentais utilizados na metodologia deste trabalho.

3.1 ALGORITMOS DE OTIMIZAÇÃO INSPIRADOS NA NATUREZA

Algoritmos de otimização NI pertencem a um grupo de algoritmos que simula processos biológicos encontrados na natureza para abordar de forma eficiente problemas complexos de otimização. A aplicação de algoritmos NI para o treinamento de redes neurais não é um tema recente, tendo tido sua primeira aplicação no trabalho de Montana e Davis (1989). No entanto, essa estratégia vem crescendo em interesse sobretudo devido ao aumento das arquiteturas de redes recentes e ao número de novos algoritmos inspirados na natureza. O trabalho de (MOLINA et al., 2020) faz uma revisão recente da área, e aponta pelo menos 6 categorias de algoritmos NI. Dentre as mais populares, podemos citar as inspiradas em otimização evolucionária, algoritmos de otimização baseados em inteligência de enxame, algoritmos de otimização híbridos, algoritmos de otimização baseados em conceitos físicos/químicos e algoritmos de otimização baseados em comportamentos sociais humanos. A Tabela 1 apresenta as características dos algoritmos NI, considerando os algoritmos mais frequentes apresentados na Figura 1. A primeira coluna da tabela é a abreviação do nome do algoritmo, a segunda coluna é o artigo em que o algoritmo foi proposto pela primeira vez, a terceira coluna é a categoria em que o algoritmo se encaixa, a quarta coluna é o número de citações do artigo em que foi proposto, consultadas pelo *Google Scholar*[1], e a última coluna é o número de trabalhos em que o algoritmo apresentou o melhor desempenho para treinar redes neurais, comparados com outros algoritmos, sendo chamada de Frequência de Melhor Desempenho (FMD).

Considerando que há um interesse crescente na utilização de algoritmos nature-inspirados para o treinamento de redes neurais, há então a necessidade de maiores estudos desses algoritmos em diversos aspectos e pouco abordados na literatura, tais como: o impacto da variação da função de ativação, taxa de convergência e tempo de execução.

Assim, os critérios utilizados nessa dissertação para a escolha dos algoritmos que serão testados foram: a popularidade, a recência, a categoria, os resultados promissores recentes, e o número de citações. Assim, foram escolhidos nove algoritmos para testes dessas variáveis, também considerando diferentes bases de dados, e métricas de avaliação. Desta forma, os algoritmos GA e PSO foram escolhidos devido à grade popularidade por décadas, estando também entre os mais antigos e utilizados. Contrariamente, o MVO foi considerado por ser um algoritmo recente, criado em 2016, mas tendo obtido os melhores resultados em trabalhos de treinamento de redes neurais diferentes. Da mesma forma, o GWO foi selecionado por ser um algoritmo conhecido por sua boa capacidade de exploração do espaço de busca, e ser ao mesmo tempo, da

[1] https://scholar.google.com.br/

Tabela 1 – Características dos algoritmos NI mais frequentes encontrados nos trabalhos apresentados no Capítulo 2

Nome	Artigo	Categoria	Citações (Google Scholar)	FMD
GA	Holland (1975)	Evolutivo	67104	0
PSO	Kennedy e Eberhart (1995)	Enxame	65350	2
DE	Storn e Price (1997)	Evolutivo	25653	0
ACO	Dorigo, Birattari e Stutzle (2006)	Enxame	13405	1
ABC	Karaboga e Basturk (2007)	Enxame	6078	0
CS	Yang e Deb (2009)	Enxame	5519	0
GWO	MIRJALILI et al., 2014	Enxame	5071	3
GSA	RASHEDI et al., 2009	Física	4776	0
BAT	Yang (2010)	Enxame	4122	3
FFA	Yang (2009)	Enxame	3517	3
BBO	Simon (2008)	Evolutivo	3014	6
WOA	Mirjalili e Lewis (2016)	Enxame	2935	1
ICA	Atashpaz-Gargari e Lucas (2007)	Social	2413	0
DFO	Mirjalili (2016)	Enxame	1052	0
MVO	Mirjalili, Mirjalili e Hatamlou (2016)	Física	768	2

categoria de inteligência de enxame, além de ter mostrado bons resultados em trabalhos recentes. Da mesma forma, considerando o critério de melhores resultados e popularidade recente, foi selecionado o FFA, que foi utilizado para o treinamento de redes neurais no trabalho de Naganna et al. (2019). Como representante da categoria de algoritmos evolutivos, além do GA, foi selecionado o algoritmo DE, levando em conta também o número de citações em outros trabalhos. Por sua vez, os algoritmos CS e BAT foram considerados principalmente pela sua popularidade crescente, apesar da recência, e também por serem algoritmos da categoria de inteligência de enxame. Finalmente, o algoritmo BBO foi considerado devido aos resultados superiores em outros trabalhos, tendo assim sido considerado entre os melhores algoritmos em alguns casos. Além disso, também foi levada em conta a crescente quantidade de citações desses algoritmos. Assim, os algoritmos de otimização NI utilizados foram o GA, BAT, BBO. CS, CS, DE, FFA, GWO, MVO e PSO, sendo descritos em maiores detalhes a seguir:

3.1.1 Algoritmo Genético

O algoritmo genético GA é um algoritmo de otimização evolutivo baseado nas características de evolução biológica, sendo introduzido por Holland (1975).

O GA resolve problemas seguindo o princípio de "sobrevivência do mais apto", de Charles Darwin, através de várias gerações consecutivas. Cada geração possui uma população de indivíduos chamados de cromossomos. Cada indivíduo representa um ponto no espaço de busca e uma solução candidata, passando por um processo de evolução que resulta na solução ótima para um problema específico.

O algoritmo baseia-se na estrutura genética e no comportamento dos cromossomos, seguindo as seguintes características:

a) Indivíduos da população competem entre si por recursos.
b) Os indivíduos melhores sucedidos em cada competição produzem mais descendentes em relação aos piores indivíduos.
c) Genes dos melhores indivíduos propagam pela população de forma que dois bons ancestrais consigam produzir alguns descendentes a partir dos seus genes.
d) Cada geração sucessiva se tornará mais adequada ao ambiente de busca, se aproximando cada vez mais da solução ótima global.

Cada cromossomo é composto por diversos genes (variáveis do problema), sendo atribuído um valor de aptidão (*fitness*) para cada um deles. O seleciona os indivíduos com melhor aptidão para entrarem no processo de reprodução. A reprodução é feita combinando informações de cromossomos ancestrais para produzir melhores descendentes. Alguns dos cromossomos da população são substituídos por novas soluções a cada geração. Eventualmente, o algoritmo converge para um conjunto de soluções ótimas de um problema específico.

Para otimizar o problema, o GA possui três principais operadores, sendo eles:

a) Seleção - Operador que define o critério de seleção das melhores soluções para as próximas gerações.
b) *Crossover* - Operador que define como as soluções selecionadas são combinadas para gerar novas soluções.
c) Mutação - Operador utilizado para criar a manter a diversidade genética dos cromossomos.

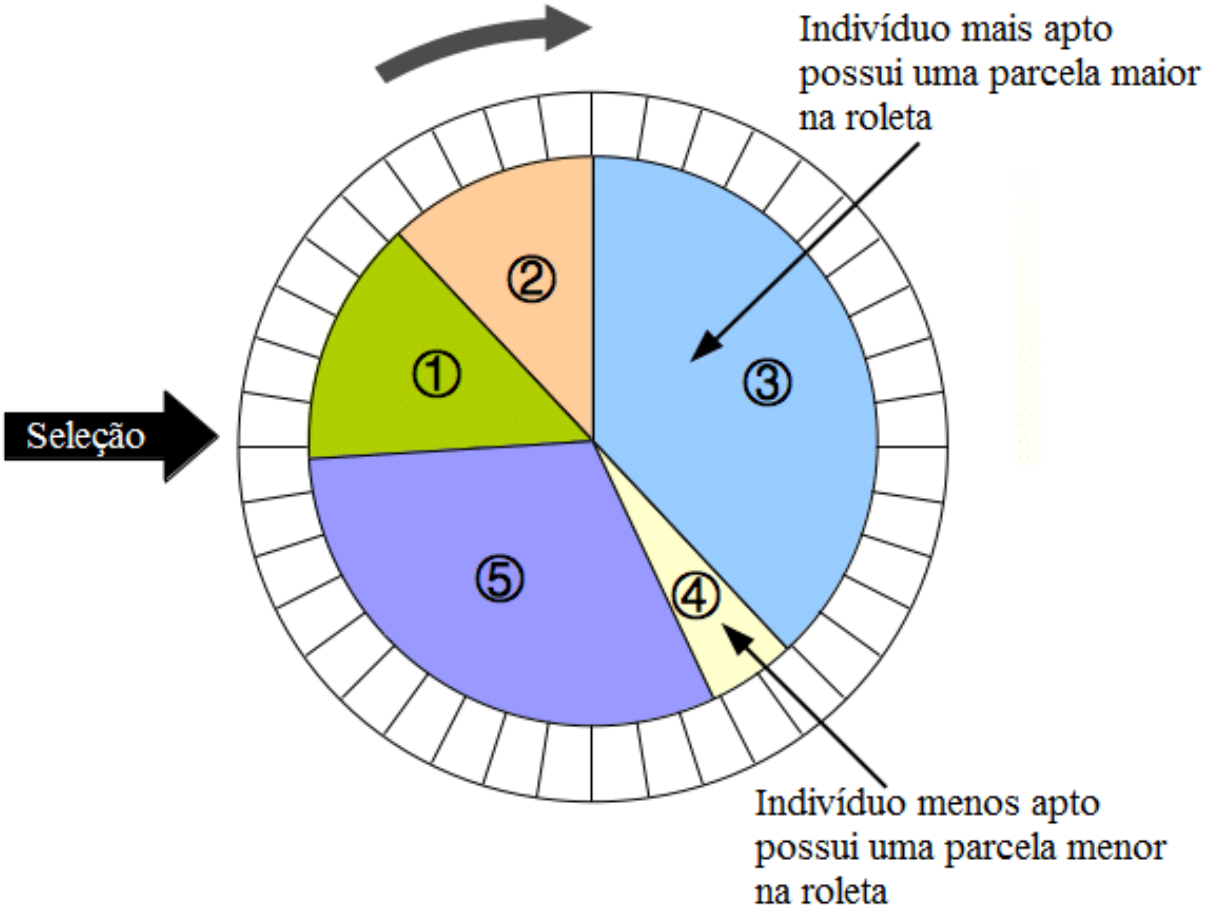
O operador de seleção dá preferência aos melhores cromossomos, que são avaliados por uma função objetivo, para serem passados para a próxima geração. Neste trabalho, a seleção é baseada no Método da Roleta, proposto por Goldberg (1988). O método divide os indivíduos em uma roleta, sendo que os indivíduos com um valor de aptidão maior possuem uma probabilidade maior de serem escolhidos. Em cada rodada, um número aleatório é escolhido e o indivíduo que tiver a área na mesma posição do número escolhido é selecionado. A roleta é girada quantas vezes forem necessárias para conseguir um par distinto de cromossomos para os operadores de *crossover* e mutação. A Figura 5 apresenta uma ilustração do Método da Roleta.

O processo de *crossover* recebe um cromossomo X e um cromossomo Y, selecionados pelo operador de seleção, sendo recombinados para gerar novas soluções descendentes. O *crossover* permite com que as características positivas dos indivíduos mais aptos sejam propagadas pelas próximas gerações.

A forma de realizar o *crossover* adotada neste trabalho é o *crossover* de um único ponto, onde é escolhido um ponto aleatório entre os cromossomos selecionados para os genes desse ponto serem trocados entre eles. A Figura 6 ilustra esse tipo de *crossover*.

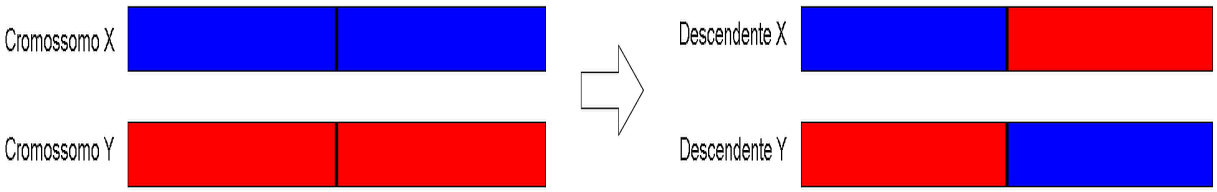
O último operador é a mutação que fornece a diversidade genética entre os cromossomos, alterando ocasionalmente a estrutura de alguns cromossomos e criando indivíduos com

Figura 5 – Ilustração do Método da Roleta com 5 indivíduos.



Fonte: Menezes (2017).

Figura 6 – Ilustração do *crossover* de um único ponto.



Fonte: Autor.

genes não encontrados na maior parte da população. Tem como função evitar com que o algoritmo convirja para um mínimo local, alterando o sentido da busca por meio de perturbações na população. Cada cromossomo da população possui uma chance de ser alterado. Para cada cromossomo que é alterado, é selecionado um gene aleatório para receber um novo valor gerado aleatoriamente a partir de uma distribuição uniforme. A Figura 7 apresenta um exemplo de um cromossomo de dados binários que sofreu mutação em dois genes aleatórios.

Figura 7 – Ilustração da mutação de um cromossomo que possui dados binários.

Fonte: Autor.

3.1.2 *Bat*

Proposto por Yang (2010), o BAT é uma técnica de otimização meta-heurística que baseia-se no comportamento de ecolocalização dos morcegos, sendo determinadas três regras para o funcionamento do algoritmo, sendo elas:

a) Todos os morcegos utilizam ecolocalização para sentir distâncias, e eles conseguem diferenciar comida/presa e barreiras de fundo.
b) Os morcegos voam aleatoriamente na posição x_i com uma velocidade v_i e uma frequência fixa f_{min}, variando o tamanho da onda λ e a sonoridade A_0 para buscar pela presa. Eles conseguem ajustar automaticamente a frequência dos seus pulsos emitidos e ajustar a taxa de emissão de pulsos $r \in [0,1]$, dependendo da proximidade do alvo.
c) Assume-se que a sonoridade varia de um A_0 alto para uma constante mínima A_{min}.

Para simular a movimentação dos morcegos, foi formalizado matematicamente como as posições x_i e velocidade v_i são atualizadas em um espaço de busca de d dimensões. As novas soluções x_i^t e velocidade v_i^t na iteração t são dadas pelas Equações (1)-(3) a seguir:

$$f_i = f_{min} + (f_{max} - f_{min})\beta, \tag{1}$$

$$v_i^t = v_i^{t-1} + (x_i^t - x_*)f_i, \tag{2}$$

$$x_i^t = x_i^{t-1} + v_i^t, \tag{3}$$

onde $\beta \in [0,1]$ é um vetor aleatório gerado a partir de uma distribuição uniforme e x_* é a melhor solução global.

Para busca local, assim que uma solução é selecionada entre as melhores, uma nova solução para cada morcego é gerada localmente utilizando movimentação aleatória de acordo com a Equação (4):

$$x_{new} = x_{old} + \epsilon A^t, \tag{4}$$

onde x_{new} é a nova posição do morcego, x_{old} é a sua posição anterior, $\epsilon \in [-1, 1]$ é um número aleatório e A^t é a média da sonoridade de todos os morcegos.

A sonoridade A_i e a taxa r_i de emissão de pulso são atualizadas conforme o prosseguimento das iterações. A sonoridade diminui quando um morcego encontra a sua presa, enquanto a taxa de emissão de pulsos aumenta. Esse processo é formalizado na Equação (5) dada a seguir:

$$A_i^{t+1} = \alpha A_i^t, \quad r_i^{t+1} = r_i^0[1 - exp(-\gamma t)], \tag{5}$$

onde $0 < \alpha < 1$ e $\gamma > 0$ são constantes.

Os passos do BAT são sumarizados no Algoritmo 1.

Algoritmo 1 – Pseudocódigo do BAT

```
Inicializar a população de morcegos x_i(1, 2, ..., n) e v_i
Definir a frequência de pulso f_i em x_i
Inicializar a taxa de pulso r_i e a sonoridade A_i
while t < número máximo de iterações do
    Gerar novas soluções ajustando a frequência, atualizando as velocidades e
     soluções de acordo com as Equações (1)-(3)
    if rand > r_i then
        Selecionar uma solução entre as melhores
        Gerar uma solução local próxima da melhor solução selecionada
    end
    Gerar uma nova solução por voo aleatório
    if rand < A_i and f(x_i) < f(x_*) then
        Aceitar a nova solução
        Aumentar r_i e reduzir A_i
    end
    Ranquear os morcegos e encontrar o melhor x_*
end
Retornar x_*
```

3.1.3 *Biogeography-Based Optimizer*

Proposto por Simon (2008), o algoritmo BBO se baseia na evolução de áreas biogeográficas, determinada pela migração de espécies entre diferentes *habitats*, para resolver problemas de otimização. No BBO, cada solução possível é chamada de ilha, ou *habitat*, que possuem habitantes que representam as variáveis da solução, também chamada de variável de índice de adequação ou *suitability index variables* (SIV). A função objetivo de cada *habitat* é chamada de índice de adequação do *habitat* ou *habitat suitability index* (HSI). Para a otimização, o algoritmo segue as seguintes regras:

a) Habitantes de qualquer *habitat* sofrem mutações, independente do seu HSI
b) Habitantes em *habitats* com maior HSI são mais prováveis a migração em relação à *habitats* que possuem um HSI menor.

c) Habitantes em *habitats* com menor HSI são mais prováveis a não migrarem.
d) A imigração é sempre de um *habitat* melhor para um pior.
e) Cada *habitat* possui taxa de imigração e emigração que define taxa de imigração de um *habitat* para outro.

A imigração entre os *habitats* é simulada por meio da troca de características entre as soluções. Para a seleção dos *habitats*, é utilizado o método da roleta, explicado na Seção 3.1.1, onde os indivíduos com maior HSI possuem uma maior parte da roleta e os indivíduos com menor HSI possuem a menor parte.

O algoritmo BBO é definido pelas seguintes etapas:

a) Inicializar a população aleatoriamente
b) Calcular as taxas de imigração e emigração, onde as melhores soluções possuem a taxa máxima de emigração e mínima de imigração, enquanto as soluções piores possuem uma taxa de imigração máxima e mínima de emigração.
c) Selecionar os *habitats* de imigração de acordo com as suas taxa de imigração pelo método da roleta.
d) Migrar SIVs aleatórias baseada nas ilhas selecionadas.
e) Realizar a mutação para cada um dos *habitats*.
f) Substituir os *habitats* pelos novos *habitats* gerados.
g) Voltar para a etapa b) enquanto um critério de parada não for atingido.

3.1.4 *Cuckoo Search via Lévy Flights*

O algoritmo de otimização CS foi proposto por Yang e Deb (2009), inspirado no comportamento parasita da ninhada de algumas espécies de cuco combinado com o padrão de voos de Lévy encontrado em alguns pássaros. O algoritmo segue três regras estabelecidas:

a) Cada cuco coloca um ovo de cada vez, depositando-o em um ninho escolhido aleatoriamente.
b) Os melhores ninhos, que possuem os ovos de maior qualidade, são levados para a próxima geração.
c) O número de ninhos disponíveis é fixo e o ovo colocado por um cuco é descoberto pelo pássaro hospedeiro com uma probabilidade $p_a \in [0,1]$. Neste caso, o pássaro hospedeiro pode jogar o ovo fora ou abandonar o ninho e construir um novo ninho.

Baseado nessas três regras, as etapas básicas do CS são descritas no pseudocódigo apresentado no Algoritmo 2.

De uma maneira mais simples, pode-se dizer que cada ovo do ninho representa uma solução candidata e um ovo de cuco representa uma solução nova. O objetivo é utilizar novas possíveis soluções melhores para substituir uma outra solução pior no ninho. É considerado apenas um ovo por ninho.

Algoritmo 2 – Pseudocódigo do algoritmo CS

```
Gerar uma população inicial de n ninhos
while t < número máximo de iterações do
    Selecionar um cuco aleatoriamente pelos voos de Lévy e avaliar a sua aptidão F_i
    Selecionar o ovo de um ninho aleatório e avaliar a sua aptidão F_j
    if F_i > F_j then
        Substituir j pela nova solução i
    end
    Uma fração (p_a) dos piores ninhos são abandonados e novos são construídos
    As soluções são ranqueadas e a melhor solução atual é encontrada
end
Retornar o melhor ninho
```

Novas soluções x^{t+1} para um cuco i são geradas realizando voos de Lévy, de acordo com a Equação (6) mostrada a seguir:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus \text{Lévy}(\lambda), \tag{6}$$

onde $\alpha > 0$ é o tamanho do passo.

O voo de Lévy fornece essencialmente uma caminhada aleatória pelo espaço de busca enquanto o tamanho aleatório do passo é definido pela distribuição de Lévy, mostrada na Equação (7):

$$\text{Lévy } u = t^{-\lambda}, \qquad (1 < \lambda \leq 3), \tag{7}$$

possuindo uma variação infinita com uma média infinita. Algumas das novas soluções geradas pelo voo de Lévy são geradas perto da melhor solução atual, agilizando a busca local. No entanto, uma fração considerável das novas soluções são geradas em locais longe o suficiente da melhor solução, garantindo que o algoritmo não fique preso em ótimos locais.

3.1.5 *Differential Evolution*

DE é um algoritmo de otimização evolutivo proposto por Storn e Price (1997). Esse algoritmo é semelhante ao GA, resolvendo problemas por meio da criação de cromossomos em várias gerações consecutivas, no entanto, os novos cromossomos são criados a partir da diferença entre os melhores cromossomos selecionados. O algoritmo consiste em quatro principais etapas: *inicialização*, *mutação*, *crossover* e *seleção*, sendo explicados a seguir.

Na primeira etapa é inicializada aleatoriamente uma população de soluções de tamanho n dentro de um espaço de busca de d dimensões. Na fase de mutação, o DE gera novas soluções adicionando a diferença ponderada entre duas soluções da população a uma terceira, de acordo com a Equação(8) mostrada a seguir:

$$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}) \tag{8}$$

onde $r1, r2, r3 \in \{1,2,...,n\} \neq i$ são índices aleatórios mutuamente diferentes. $F \in]0, 2]$ é uma constante que controla a variação diferencial $(x_{r_2,G} - x_{r_3,G})$.

Os parâmetros das novas soluções geradas são combinados com os parâmetros de uma solução alvo, gerando uma solução experimental, definida pela Equação (9). Esse processo é chamado de *crossover* e é definido pela Equação (10):

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1},..., u_{di,G+1}) \tag{9}$$

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & \text{if } (randb(j) \leq CR \text{ ou } j = j_{rand} \\ x_{ji,G} & \text{if } (randb(j) > CR \text{ e } j \neq j_{rand} \end{cases} \tag{10}$$

onde $randb(j)$ é um número aleatório $\in [0,1]$, $CR \in [0,1]$ é a constante de *crossover*, j_{rand} é um índice aleatório $\in \{1,2,...,d\}$ que garante que $u_{i,G+1}$ recebe pelo menos um parâmetro de $v_{i,G+1}$.

Na última etapa de *seleção*, a solução experimental $u_{i,G+1}$ é comparada com a solução alvo $x_{i,G}$. Se $u_{i,G+1}$ tiver um valor menor do resultado da função de avaliação, então $x_{i,G+1} = u_{i,G+1}$, caso o contrário, o valor de $x_{i,G}$ é mantido para a próxima geração. Essas etapas se repetem até um critério de parada ser alcançado. O pseudocódigo do algoritmo DE é apresentado no Algoritmo 3.

Algoritmo 3 – Pseudocódigo do algoritmo DE

```
Gerar uma população inicial de tamanho n
for G = 0 até número máximo de iterações do
    for i = 0 até n do
        Gerar três inteiros aleatórios r1, r2, r3 ∈ [1, n], sendo r1 != r2 != r3 != i
        Gerar um inteiro aleatório j_rand ∈ [1, d]
        for para cada parâmetro j do
            randb = rand(0,1)
            v_{i,G+1} = x_{r1,G} + F · (x_{r2,G} − x_{r3,G})
            if randb < CR or j = j_rand then
                u_{ji,G+1} = v_{ji,G+1}
            else
                u_{ji,G+1} = x_{ji,G}
        end
        if u_{i,G+1} < x_{i,G} then
            x_{i,G+1} = u_{i,G+1}
        else
            x_{i,G+1} = x_{i,G}
    end
end
Retornar o melhor indivíduo
```

3.1.6 *Firefly*

O FFA é um algoritmo de otimização inspirado no comportamento de brilho de luz dos vaga-lumes, proposto por Yang (2009). Cada solução candidata é considerada um vaga-lume e a qualidade da solução é representada pela intensidade do brilho do vaga-lume. O algoritmo possui três regras:

a) Todos os vaga-lumes são unissexuais.
b) A atração entre os vaga-lumes é proporcional aos seus brilhos. O vaga-lume menos brilhoso se movimenta em direção ao mais brilhoso. Em casos de empate, o movimento é aleatório.
c) O brilho do vaga-lume é determinado pelo resultado da função objetivo. Quanto mais próximo da solução ótima, mais brilhoso é o vaga-lume.

Para simular a intensidade de luz dos vaga-lumes, foi proposto um modelo matemático dado pela Equação (11):

$$I(r) = I_0 e^{-\gamma r^2}, \tag{11}$$

sendo I_0 a intensidade original da luz, γ é o coeficiente de absorção de luz e r é a distância entre os vaga-lumes.

Com a atração dos vaga-lumes sendo proporcional à intensidade de luz vista pelos vaga-lumes adjacentes, a atratividade β é definida pela Equação (12):

$$\beta(r) = \beta_0 e^{-\gamma r^m} \tag{12}$$

onde β_0 é a atratividade quando $r = 0$.

O movimento de um vaga-lume i que é atraído por um vaga-lume j mais brilhoso no espaço de busca é definido pela Equação (13):

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha \epsilon_i, \tag{13}$$

onde α é um parâmetro de aleatorização e ϵ_i é um vetor de números aleatórios gerados a partir de uma distribuição Gaussiana.

O Algoritmo 4 apresenta um pseudocódigo do FFA.

3.1.7 *Grey Wolf Optimizer*

Inspirado no comportamento de caça de lobos cinzentos, o algoritmo de otimização GWO foi apresentado por Mirjalili, Mirjalili e Lewis (2014a). O algoritmo simula a hierarquia de liderança dos lobos cinzentos na natureza, dividindo-os em quatro grupos, sendo eles:

a) *Alpha*: Líder do grupo.
b) *Beta*: Segundo em comando.
c) *Delta*: Terceiro em comando.

Algoritmo 4 – Pseudocódigo do FFA

```
Função objetivo f(x)
Inicializar a população de vaga-lumes x_i(i = 1,2...n)
Calcular a intensidade de luz I_i de x_i por f(x) while t<número máximo de iterações do
    for i=1 até n do
        for j=1 até n do
            Calcular a atratividade entre x_i e x_j de acordo com a Equação (12) if I_j > I_i then
                Movimentar x_i em direção de x_j de acordo com a Equação (13)
            end
            Calcular a nova solução e atualizar a intensidade de luz
        end
    end
    Ranquear os vaga-lumes e encontrar o mais brilhoso
end
Retornar o vaga-lume mais brilhoso
```

d) *Omega*: Lobos no menor nível da hierarquia.

Além da hierarquia dos lobos, a caça em grupo é outro aspecto que o algoritmo se inspira para a estratégia de otimização, tendo como principais etapas:

a) Busca, rastreamento e aproximação da presa.
b) Perseguição e cerco até a presa parar de se mover.
c) Ataque à presa.

Cada solução candidata é considerada um lobo cinzento, sendo a solução com a melhor aptidão considerada o lobo *alpha* (α), a segunda melhor solução é considerada o lobo *beta* (β) e a terceira melhor solução o lobo *delta* (δ). O resto das soluções candidatas são consideradas como *omega* (ω). A caça (otimização) é conduzida por α, β e δ, enquanto os lobos ω os seguem.

O modelo matemático do cerco da presa pelos lobos durante a caça é dado pelas Equações (14) e (15):

$$\vec{D} = |\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)| \tag{14}$$

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \cdot \vec{D} \tag{15}$$

sendo $\vec{D}$ o vetor de distância do lobo para a presa, t a iteração atual, $\vec{A}$ e $\vec{C}$ são vetores de coeficientes, $\vec{X}_p$ é o vetor de posição da presa e $\vec{X}$ o vetor de posição do lobo cinzento. $\vec{A}$ e $\vec{C}$ são calculados de acordo com as Equações (16) e (17) a seguir:

$$\vec{A} = 2\vec{a} \cdot \vec{r1} - \vec{a} \tag{16}$$

$$\vec{C} = 2 \cdot \vec{r2} \tag{17}$$

onde $\vec{a}$ é um vetor de componentes que decrescem linearmente de 2 até 0 ao longo das iterações e $\vec{r1}$, $\vec{r2}$ são vetores aleatórios dentro do intervalo [0,1].

Para simular matematicamente o comportamento de caça dos lobos cinzentos, é assumido que α, β e δ possuem um conhecimento melhor sobre a possível localização da presa. Para isso, as três melhores soluções até o momento são armazenadas e os outros agentes de busca (os lobos ω) são obrigados a atualizar a sua posição de acordo com os três melhores agentes de busca. Esse processo é formalizado pelas Equações (18)-(20) dadas a seguir.

$$\vec{D}_\alpha = |\vec{C}_1 \cdot X_\alpha \vec{} - \vec{X}|, \vec{D}_\beta = |\vec{C}_2 \cdot X_\beta \vec{} - \vec{X}|, \vec{D}_\delta = |\vec{C}_3 \cdot X_\delta \vec{} - \vec{X}| \tag{18}$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot (\vec{D}_\alpha), \vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot (\vec{D}_\beta), \vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot (\vec{D}_\delta) \tag{19}$$

$$X(\vec{t} + 1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{20}$$

Para modelar matematicamente a aproximação da presa, o alcance de $\vec{A}$ diminui conforme o valor de $\vec{a}$ diminui. Assim, quando $|\vec{A}| < 1$ o lobo é forçado atacar a presa, tendo o seu movimento direcionado à ela. Por outro lado, quando $|\vec{A}| \geq 1$, o lobo é forçado e divergir da presa em questão para buscar uma outra presa que resulte em uma melhor solução. O Algoritmo 5 apresenta um pseudocódigo do GWO.

Algoritmo 5 – Pseudocódigo do algoritmo GWO

```
Inicializar a população de lobos cinzentos
Inicializar a⃗, A⃗ e C⃗
Calcular a aptidão de todos os agentes de busca
X_α = melhor agente de busca
X_β = segundo melhor agente de busca
X_δ = terceiro melhor agente de busca
while t < número máximo de iterações do
    for para cada agente de busca do
        Atualizar a posição do agente de busca atual de acordo com a Equação (20)
    end
    Atualizar a⃗, A⃗ e C⃗
    Calcular a aptidão de todos os agentes de busca
    Atualizar X_α, X_β e X_δ
    t = t+1
end
Retornar X_α
```

3.1.8 *Multi-Verse Optimizer*

Proposto por Mirjalili, Mirjalili e Hatamlou (2016), o algoritmo MVO é um algoritmo de otimização baseado no conceito de múltiplos universos, simulando a interação entre eles através de buracos brancos, buracos negros e buracos de minhoca. Cada solução candidata é considerada um universo e cada objeto do universo é assumido como uma variável do problema a ser otimizado. Além disso, é atribuída uma taxa de inflação para cada universo, correspondendo ao valor obtido pela função de avaliação sobre o mesmo. A ideia principal do algoritmo se origina do fato de grandes universos tenderem a enviar objetos para universos menores para alcançarem estabilidade. Um grande universo é definido baseado na taxa de inflação da teoria de multiversos. Durante a etapa de otimização, o algoritmo se baseia em algumas regras principais, sendo elas:

a) Quanto maior a taxa de inflação, maior é a chance de existir um buraco branco.
b) Quanto maior a taxa de inflação, menor é a chance de existir um buraco negro.
c) Universos com alta taxa de inflação tendem a enviar objetos por buracos brancos.
d) Universos com baixar taxa de inflação tendem a receber objetos por buracos negros.
e) Os objetos de todos os universos podem se movimentar aleatoriamente em direção ao melhor universo através de buracos de minhoca, independente da taxa de inflação.

Para modelar matematicamente os túneis de buracos negros/brancos e a troca de objetos entre os universos, é utilizado um mecanismo de roleta. Em cada iteração, os universos são ordenados de acordo com a sua taxa de inflação e um deles é escolhido pela roleta para ter um buraco branco.

Considera-se que cada universo possui um buraco de minhoca para transportar objetos através do espaço aleatoriamente para manter a diversidade dos universos. Para fornecer mudanças locais para cada universo e ter uma alta probabilidade de aprimorar a sua taxa de inflação utilizando buracos de minhoca, considera-se que exista um buraco de minhoca entre um universo e o melhor universo até o momento. Esse mecanismo é definido pela Equação (21).

$$x_i^j = \begin{cases} \begin{cases} X_j + TDR \times ((ub_j - lb_j) \times r4 + lb_j), & \text{if} \quad r3 < 0.5 \\ X_j - TDR \times ((ub_j - lb_j) \times r4 + lb_j), & \text{if} \quad r3 \geq 0.5 \end{cases}, & \text{if} \quad r2 < WEP \\ x_i^j, & \text{if} \quad r2 \geq WEP \end{cases} \tag{21}$$

onde X_j indica o parâmetro j do melhor universo até o momento, TDR e WEP são coeficientes, lb_j é o limite inferior da variável j, ub_j é o limite superior da variável j, x_i^j indica o parâmetro j do universo i, e $r2, r3, r4$ são número aleatórios dentro do intervalo [0,1].

O coeficiente *Wormhole Existence Probability* (WEP) define a probabilidade de existir um buraco de minhoca no universo, sendo definido pela Equação (22). O *Travelling Distance*

Rate (TDR) define a taxa de distância (variação) que um objeto pode ser teleportado por um buraco de minhoca em volta do melhor universo encontrado até o momento, sendo definido pela Equação (23).

$$WEP = min + l \times (\frac{max - min}{L}) \tag{22}$$

onde l é a iteração atual e L é o número máximo de iterações.

$$TDR = 1 - \frac{l^{\frac{1}{p}}}{L^{\frac{1}{p}}} \tag{23}$$

onde p define a precisão de exploração ao longo das iterações. Quanto maior o valor de p, mais rápido e mais preciso é a busca local.

O processo de otimização do algoritmo MVO começa criando um conjunto de universos aleatórios. Em cada iteração, objetos de universos com altas taxas de inflação tendem a se mover para universos com taxas menores através de buracos negros/brancos. Enquanto isso, ocorrem teletransportes aleatórios nos objetos de todos os universos através de buracos de minhoca em direção ao melhor universo. Esse processo é repetido até atingir-se um critério de parada.

3.1.9 *Particle Swarm Optimization*

O algoritmo PSO foi proposto por Kennedy e Eberhart (1995), sendo um algoritmo de otimização baseado no comportamento de um bando de pássaros em busca de alimentos. É um dos algoritmos de otimização mais populares devido à sua simplicidade para chegar à solução ótima, sendo utilizado amplamente em diversas aplicações (MOLINA et al., 2020). Cada partícula i representa uma solução candidata de um problema específico, possuindo uma posição x_i e movimentando-se no espaço de busca com uma velocidade v_i, de acordo com a posição da melhor partícula, chamada de *Global Best* (gbest) e a sua melhor posição até o momento, conhecida como *Personal Best* (pbest). O algoritmo PSO possui alguns parâmetros de controle, sendo eles:

a) Peso de inércia (w).
b) Número de partículas (m).
c) Constantes de aceleração (c_1 e c_2).
d) Limite máximo de velocidade (v_{max}).

A velocidade das partículas é ajustada dinamicamente de acordo com a Equação (24) e a sua posição é ajustada de acordo com a Equação (25).

$$v_{id}(t+1) = wv_{id}(t) + c_1 r_1(p_{id} - x_{id}(t)) + c_2 r_2(p_{gd} - x_{id}(t)) \tag{24}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{25}$$

sendo $x_{id}(t)$ a posição atual da partícula i, p_{id} a melhor posição da partícula até o momento, p_{gd} a posição da melhor partícula de todo o enxame. r_1 e r_2 são dois números aleatórios dentro do intervalo [0,1].

O peso de inércia representa o quanto a velocidade anterior de uma partícula influencia em sua velocidade atual, permitindo melhorar a habilidade de busca local e global do PSO. A velocidade da partícula depende exclusivamente de Pbest e Gbest, caso $w = 0$. Se $w \neq 0$ a partícula tende a explorar novos espaços. Quanto maior o valor de w, maior é a velocidade da partícula, fazendo com que ela explore com passos maiores. Com valores de w menores, a partícula tende a explorar novos espaços próximos à sua posição atual.

O peso de inércia mais utilizado atualmente é o *Linearly Decreasing Weight* (LDW), proposto por Shi e Eberhart (1999), sendo definido pela Equação (26).

$$w = w_{max} - \frac{w_{max} - w_{min}}{T_{max}} t, \tag{26}$$

onde w_{max} é o peso de inércia máximo, w_{min} é o peso de inercia mínimo, t é o número da iteração atual e T_{max} é o número máximo de iterações. O Algoritmo 6 apresenta um pseudocódigo da técnica PSO.

Algoritmo 6 – Pseudocódigo do algoritmo PSO.

```
Inicializar as partículas aleatoriamente
while t < T_max do
    Calcular a aptidão das partículas
    for i=1 até m do
        Encontrar pbest
        Encontrar gbest
        for d=1 até número de dimensão das partículas do
            Atualizar x_i pela Equação (24) e (25)
        end
    end
    Atualizar w pela Equação (26)
end
Retornar a melhor partícula
```

3.2 REDES NEURAIS ARTIFICIAIS

RNAs são ferramentas da área de aprendizado de máquina utilizadas para a classificação e regressão de dados, inspirando-se em modelos de redes neurais biológicas. As bases do modelo de neurônio artificial surgiram no artigo de McCulloch e Pitts (1943), onde o neurônio possui apenas uma saída gerada pela soma do valor de suas diversas entradas ponderadas por um conjunto de pesos, simulando as sinapses presentes entre os neurônios biológicos.

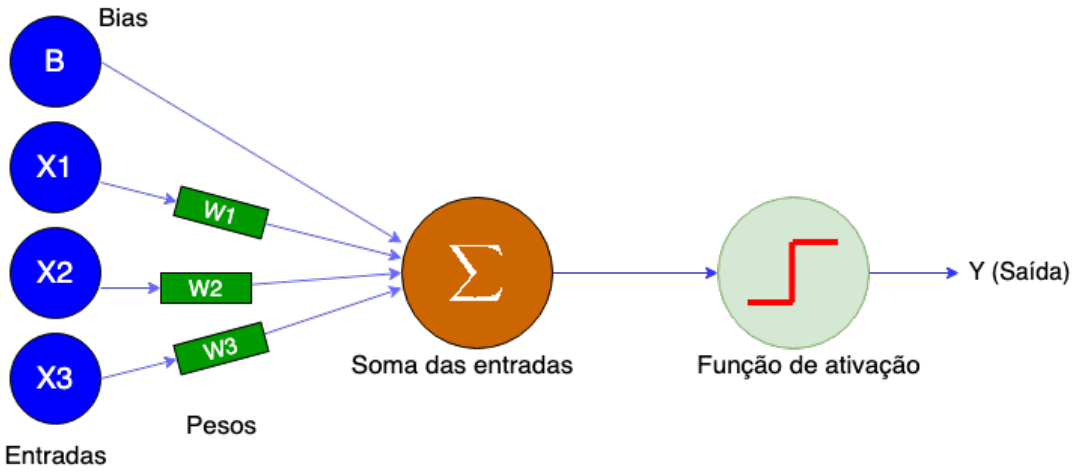
Rosenblatt (1958) construiu o modelo de *perceptron*, sendo um modelo computacional para classificação binária baseado na estrutura do modelo de McCulloch e Pitts (1943), onde

dada uma entrada x, um valor de saída $f(x)$ é retornado de acordo com a função de ativação dada pela Equação (27):

$$f(x) = \begin{cases} 1 \text{ se } w.x + b \geq 0 \\ 0 \text{ caso contrário} \end{cases}, \tag{27}$$

sendo w o vetor de pesos, x as entradas e b o *bias* da função. A Figura 8 ilustra um exemplo de um modelo de Perceptron, possuindo três entradas e uma saída.

Figura 8 – Ilustração de um modelo de Perceptron.



Fonte: Autor.

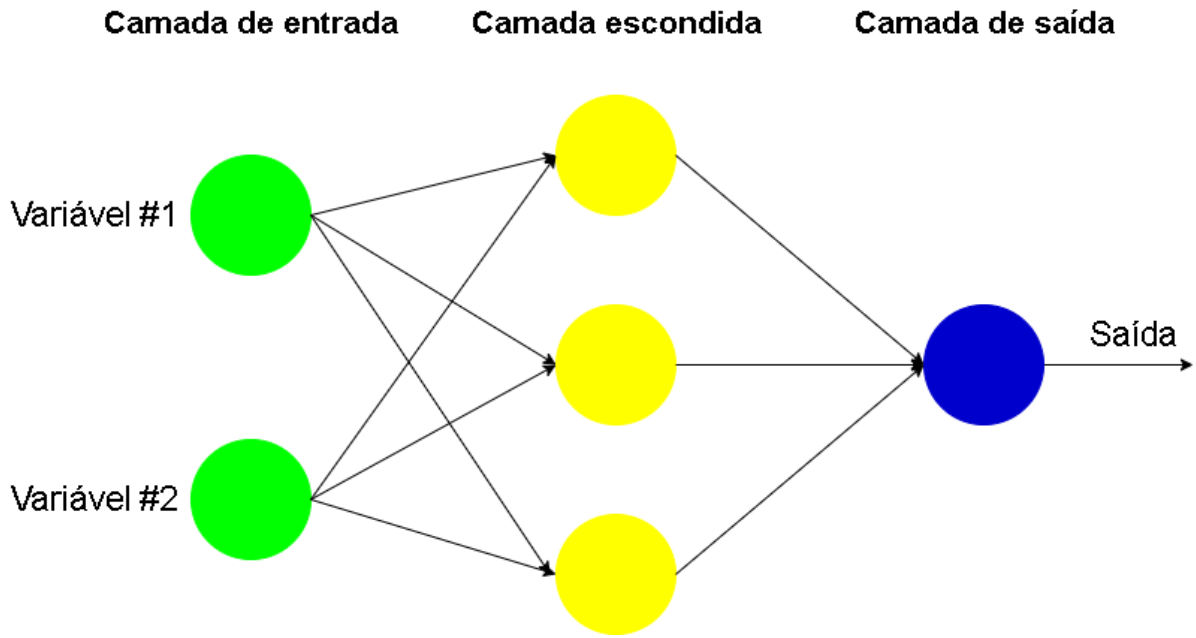
Os autores Minsky e Papert (1969) mostraram que o *perceptron* era limitado apenas a classificar dados linearmente separáveis, sendo incapaz de aprender a função lógica *XOR* (*OU* exclusivo). Além disso, o modelo de *perceptron* também era incapaz de classificar problemas não-binários, ou seja, que possuíam mais de duas classes. Para resolver essas limitações, surgiram as redes neurais MLP.

3.2.1 *Multilayer Perceptron*

As redes MLP foram apresentadas por Rumelhart, Hinton e Williams (1986), sendo uma classe de redes neurais *feedforward*, ou seja, redes que não possuem um ciclo entre os nós e propagam a informação de entrada para apenas uma direção. Elas consistem de uma camada de entrada, N camadas escondidas e uma camada de saída. Cada camada possui M neurônios conectados com os neurônios da camada seguinte. A força da conexão entre os neurônios é representada por pesos. A Figura 9 ilustra um exemplo de uma rede neural MLP.

A camada de entrada recebe um conjunto de dados numéricos. Cada neurônio realiza a soma da entrada ponderada e aplica a sua função de ativação, gerando o resultado para os

Figura 9 – Ilustração de uma rede MLP que possui 2 entradas e 3 neurônios na camada escondida.



Fonte: Autor.

próximos neurônios da rede. A saída obtida pela rede é o resultado gerado pela camada de saída.

Para a rede MLP conseguir classificar corretamente um certo domínio de dados, é necessário encontrar os pesos que otimizam a classificação. Assim, esse tipo de rede faz uso de um algoritmo de aprendizado supervisionado popular, chamado de Retropropagação ou *Backpropagation* (BP), descrito na próxima seção.

3.2.2 *Backpropagation*

O algoritmo BP, apresentado por Rumelhart, Hinton e Williams (1986), é um algoritmo para treinar RNAs através da retro-propagação dos erros da camada de saída para as camadas intermediárias. Os pesos da rede são otimizados de forma que ela aprenda como mapear corretamente as entradas para as saídas.

O processo de aprendizado do algoritmo BP possui dois estágios:

a) *Feed-Forward*: É feita a leitura das entradas, que se propagam pela rede até a camada de saída.
b) *Feed-Backward*: Calcula-se o erro de saída da rede, que é propagado da camada de saída até a primeira camada escondida, atualizando os pesos das conexões entre os neurônios.

Considerando uma rede neural MLP com N camadas, a saída de sua primeira camada escondida é dada pela Equação (28):

$$h^{(0)} = g^{(0)}(W^{(0)T}x + b^{(0)}), \tag{28}$$

e a saída da segunda camada escondida é definida pela Equação (29):

$$h^{(1)} = g^{(1)}(W^{(1)T}h^{(0)} + b^{(1)}), \tag{29}$$

e sucessivamente as camadas seguintes recebem a saída da camada anterior como entrada.

A saída da última camada da rede $\hat{y}$, a camada de saída, é dada pela Equação (30):

$$\hat{y}(x) = W^{(N)T}h^{(N-1)} + b^{(N)}, \tag{30}$$

sendo que $h^{(i)}$ é saída da camada escondida i, $g^{(i)}$ é a função de ativação da camada escondida i, podendo ser a função sigmoide ou a função de ativação ReLU, apresentadas na Seção 3.2.4. $W^{(i)}$ é a matriz de pesos de camada escondida i, x é o vetor de entrada da rede neural, N o número de camadas escondidas da rede e $b^{(i)}$ é a matriz de *bias* da camada i.

Como o aprendizado para a classificação com rede neural é supervisionado, após a entrada ser propagada pela rede neural *feed-forward* e a saída $\hat{y}$ ser gerada, ela é comparada com um vetor de respostas y para calcular uma função de custo $C(y, \hat{y})$, como o MSE, por exemplo, que representa a qualidade da rede em realizar predições. Quanto mais próximo $\hat{y}$ estiver de y, menor será a função de custo da rede. Portanto, a função de custo será alta durante o começo do treinamento de uma rede neural e baixa em um modelo treinado. O objetivo do treinamento é reduzir a função de custo o máximo possível, ou seja, ajustar os pesos da rede gradualmente para encontrar o ponto mínimo de C. Em outras palavras, a derivada de C em relação ao conjunto de pesos e *bias* da rede neural $\frac{\partial C}{\partial \theta}$ deve ser calculada.

Para realizar esse cálculo, é utilizada a regra da cadeia, onde uma função z dependente da função y, que é dependente de x, tem a sua derivada em relação a x dada pela Equação (31):

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} \tag{31}$$

Assim, o algoritmo BP calcula primeiro a derivada na última camada da rede. Esse resultado é passado para o cálculo da derivada da penúltima camada, que é armazenado e reutilizado para antepenúltima rede e assim sucessivamente. Dessa forma, o BP retro-propaga da última camada até a primeira camada da rede, utilizando o último cálculo de derivada para calcular a derivada da camada atual e atualizando os seus pesos por sucessivas iterações, de acordo com a Equação (32):

$$\theta^{t+1} = \theta^t - \alpha\frac{\partial C}{\partial \theta}, \tag{32}$$

onde α é a taxa de aprendizado que define o quanto a função de custo é variada e θ^t é o conjunto de pesos e *bias* na iteração t.

3.2.2.1 Backpropagation com momentum

No trabalho de Rumelhart, Hinton e Williams (1986), foi adicionado um termo de *momentum* no algoritmo BP com o intuito de deixar o conjunto de pesos atual ser influencia pelo conjunto de pesos de iterações passadas, a partir da Equação (33). Nesse trabalho, esse método é chamado de BPMA.

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial C}{\partial \theta} + c\theta^{t-1}, \tag{33}$$

onde c é uma constante de *momentum* que determina a influência da iteração passada na iteração atual.

3.2.3 *Adam*

Um outro algoritmo de otimização estocástica baseado em descida de gradiente é o *Adam*, sendo apresentado por Kingma e Ba (2014). O algoritmo é baseado nos métodos *AdaGrad* (DUCHI; HAZAN; SINGER, 2011) e *RMSProp* (HINTON; SRIVASTAVA; SWERSKY, 2012), mostrando-se efetivo em bases de dados de alta dimensionalidade e que possuem distribuições não-estacionárias, ou seja, possuem média e variância que mudam ao longo do tempo. Esse método possui os seguintes parâmetros:

a) α: taxa de aprendizado.
b) β_1: taxa de decaimento exponencial do vetor de estimativas de momento m_t.
c) β_2:taxa de decaimento exponencial do vetor de estimativas de momento v_t.
d) ϵ: Valor pequeno para evitar divisão por zero.

O Algoritmo 7 apresenta um pseudocódigo do *Adam*.

Algoritmo 7 – Pseudocódigo do algoritmo *Adam*.

Inicializar os vetores de momento m_0 e v_0
while $t <$ *Número de iterações máxima* **do**
    t = t + 1
    $g_t = \nabla_\theta f(\theta^{-1})$ // Calcular o gradiente da função objetivo
    $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ // Atualizar a primeira estimativa de momento
    $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (g_t \odot g_t)$ // Atualizar a segunda estimativa de momento
    $\hat{m}_t = m_t / (1 - \beta_1^t$ // Criar a estimativa $\hat{m}_t$
    $\hat{v}_t = v_t / (1 - \beta_2^t$ // Criar a estimativa $\hat{v}_t$
    $\theta^t = \theta^{t-1} - \alpha \cdot \hat{m}_t / \sqrt{\hat{v}_t} + \epsilon$ // Atualizar os pesos
end
Retornar θ^t

3.2.4 Funções de ativação

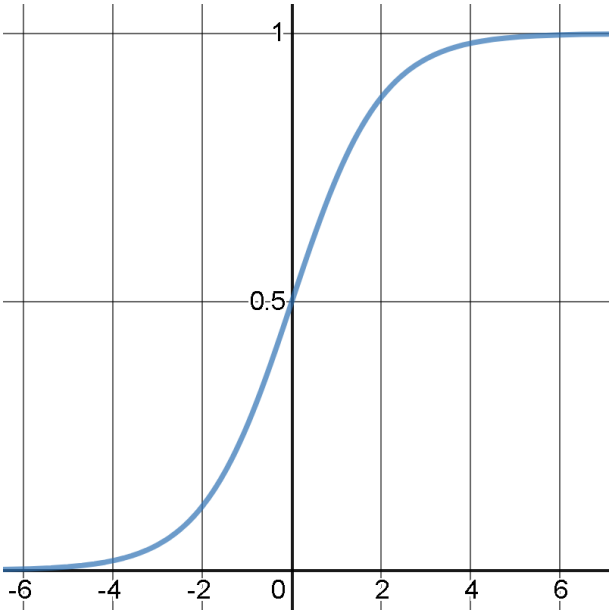
Uma função de ativação é uma função matemática que determina a saída de cada neurônio de uma rede neural. Funções de ativações também ajudam a normalizar valores em intervalos de 0 a 1 ou -1 a 1, por exemplo. Além disso, são responsáveis por dar capacidade representativa às RNAs, introduzindo um componente não linear, que faz com que elas aprendam mais do que relações lineares entre variáveis dependentes e independentes. As funções de ativação utilizadas nesse trabalho são apresentadas a seguir.

3.2.4.1 Sigmoide

A função sigmoide (σ), também conhecida como função logística, é uma das funções mais utilizadas para RNAs, sendo a primeira função sigmoidal a ser utilizada no contexto de RNAs (DUCH; JANKOWSKI, 1999). Além disso, pelo fato da sua ativação ter sido inspirada na neurociência, isso também fez com que ela fosse explorada na área de aprendizado de máquinas (MALMGREN; BORGA; NIKLASSON, 2012). A função σ é definida pela Equação (34) e ilustrada na Figura 10.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{34}$$

Figura 10 – Função σ.



Fonte: Autor.

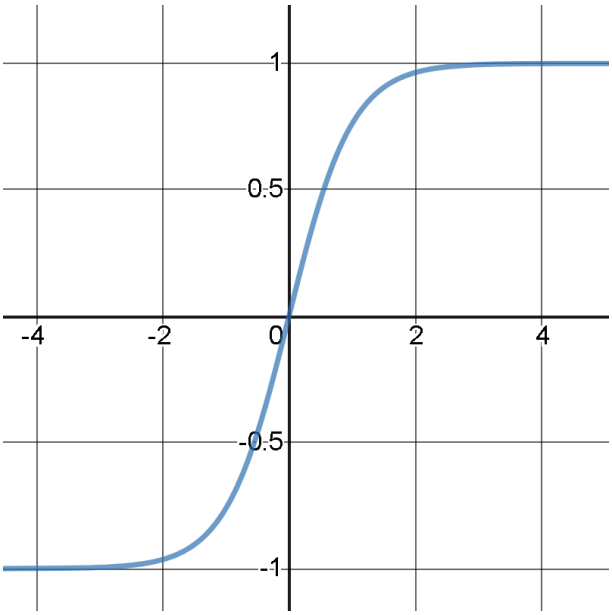
Uma vez que a função σ assume apenas valores entre 0 e 1, ela pode ser uma boa escolha para modelar o comportamento de neurônios biológicos, que funcionam de forma binária.

3.2.4.2 Tangente Hiperbólica

A função Tangente Hiperbólica (Tanh) é uma função similar à sigmoide, possuindo também um formato de 'S', porém ela varia de -1 a 1. Essa flexão da função sigmoide para atuar no intervalo $[-1, 1]$ faz com que a função Tanh se aproxime mais da função identidade e permite uma melhor convergência da rede neural, sendo recomendada como substituta da sigmoide (LECUN et al., 2012). A função Tanh é definida pela Equação (35) e ilustrada na Figura 11.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{35}$$

Figura 11 – Função tangente hiperbólica.



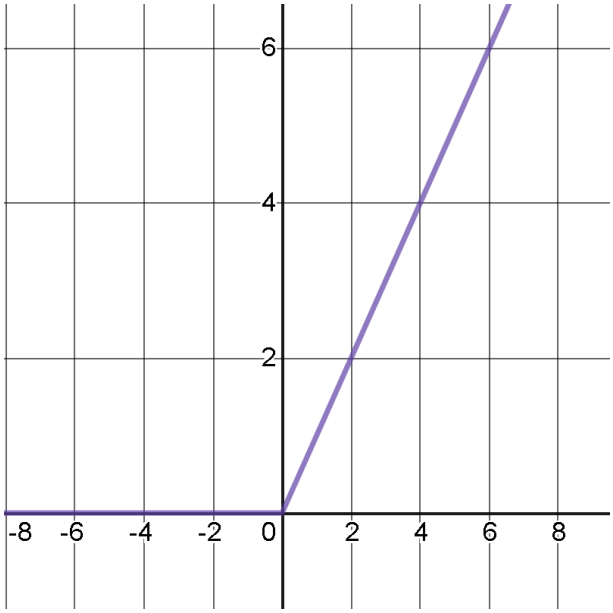
Fonte: Autor.

3.2.4.3 Unidade Linear Retificada

A Unidade Linear Retificada, também conhecida como ReLU é uma função de ativação que ganhou destaque com o seu uso em redes neurais profundas, permitindo com que a rede tenha uma rápida convergência graças à eficiência da sua ativação (NAIR; HINTON, 2010). A função ReLU é definida pela Equação (36) e ilustrada na Figura 12.

$$\text{ReLU}(x) = max(0,x) \tag{36}$$

Figura 12 – Função ReLU



Fonte: Autor.

3.3 TREINAMENTO DE UMA REDE NEURAL UTILIZANDO UM ALGORITMO DE OTIMIZAÇÃO META-HEURÍSTICO

O objetivo de utilizar um algoritmo de otimização meta-heurístico ao invés do BP para treinar uma rede neural é otimizar os pesos de treinamento de forma mais rápida utilizando técnicas de buscas ótimas globais e exploração local para evitar a parada em mínimos locais (ZHANG; SUGANTHAN, 2016). Um dos primeiros trabalhos a aplicar esse conceito com sucesso foi feito por Whitley (1989), utilizando o GA para encontrar os pesos ótimos de uma RNA. Desde então, foram exploradas diversas variações com novos algoritmos que surgiram, mas a ideia geral dessa estratégia manteve-se similar como descrita a seguir.

Algoritmos de otimização meta-heurísticos buscam por soluções ótimas por meio de uma população de soluções candidatas para um problema específico. O tipo de população varia de acordo com o algoritmo de otimização utilizado. Para o problema do treinamento de uma rede neural, considerando uma rede MLP com apenas 1 camada escondida, cada agente de busca P da população é considerado como um vetor de dimensão D, sendo $D = (I * H) + (H * O) + H + O$, onde I, H e O são o número de neurônios da camada de entrada, escondida e de saída, respectivamente. P representa então os pesos e *bias* relacionados às conexões da rede neural.

A primeira etapa é a inicialização aleatória dos pesos e *bias* iniciais da população do algoritmo utilizado. Em seguida, as amostras de treinamento são enviadas para a camada de entrada da rede. Após as entradas passarem pelas camadas escondidas, calcula-se o erro comparando a saída da RNA com a saída esperada da amostra. Com isso, é calculado o valor de aptidão (*fit-*

ness) de cada agente de busca da população, utilizando funções de erro como MSE, SSE, RMSE. Assim, realiza-se a minimização dessa função para encontrar o agente com o melhor valor de aptidão. O algoritmo gera novos pesos e *bias* aleatórios para a população da próxima iteração, que é avaliada novamente pela função objetivo. Essas etapas se repetem enquanto um critério de parada não seja atingido. Após a parada, a solução com o menor erro de todas as iterações é considerada como solução ótima, sendo utilizado os seus pesos e *bias* para classificar ou predizer dados futuros. O Algoritmo 8 apresenta um pseudocódigo do processo de otimização de uma RNA por um algoritmo meta-heurístico.

Algoritmo 8 – Otimização dos pesos de uma RNA

```
Ler a base de treinamento
Inicializar a população do algoritmo de otimização aleatoriamente
while Critério de parada não for alcançado do
    for i=1 até o tamanho da população do
        for j=1 até o tamanho da base de treinamento do
            Calcular a saída da camada escondida
            Calcular a saída da rede atual
            Calcular o erro
        end
        Calcular o valor de aptidão utilizando a função objetivo
    end
    Selecionar o agente de busca com o melhor valor de aptidão
    Atualizar os pesos e bias da população de acordo com o algoritmo de otimização
end
Retornar os valores de pesos e bias ótimos.
```

3.4 MÉTRICAS DE AVALIAÇÃO

Nessa seção serão apresentadas as métricas utilizadas nesse trabalho. Essas métricas são utilizadas como função de custo para a rede neural, função objetivo dos algoritmos de otimização e para avaliar a qualidade do modelo do treinado.

3.4.1 Erro quadrático médio

O MSE, é uma métrica para medir a qualidade da estimativa de um modelo, sendo utilizada como função de custo de uma rede neural ou função de objetivo de um algoritmo de otimização. Quanto menor o valor de MSE, melhor é a predição de um modelo. Ela é definida pela Equação (37).

$$MSE(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} [(\hat{y} - y)^2], \tag{37}$$

onde N é o número de amostras, $\hat{y}$ é a saída estimada e y é a saída esperada.

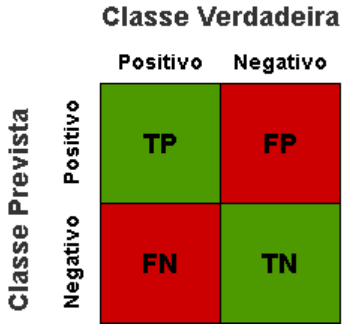
3.4.2 Matriz de Confusão

A matriz de confusão é uma forma de visualizar o desempenho de um modelo de classificação (TING; SAMMUT; WEBB, 2011). Considerando um modelo de classificação binária, que prediz duas classes, chamadas de positiva e negativa, a sua matriz de confusão indica a quantidade de exemplos de cada grupo, chamados de *True Positive* (TP), *True Negative* (TN), *False Positive* (FP) e *False Negative* (FN), sendo descritos a seguir:

a) Verdadeiro positivo (TP): Número de predições que o classificador previu corretamente uma classe positiva como positiva.
b) Verdadeiro negativo (TN): Número de predições que o classificador previu corretamente uma classe negativa como negativa.
c) Falso positivo (FP): Número de predições que o classificador previu uma classe negativa incorretamente como positiva.
d) Falso negativo (FN): Número de predições que o classificador previu uma classe positiva incorretamente como negativa.

A Figura 13 apresenta uma ilustração de uma matriz de confusão binária.

Figura 13 – Ilustração de uma matriz de confusão para um modelo de classificação binária.



Fonte: Autor.

A partir da matriz de confusão, é possível extrair outras métricas para avaliar o desempenho do classificador, sendo elas apresentadas a seguir.

3.4.2.1 Acurácia

A acurácia representa o quanto de todas as amostras que foram classificadas corretamente pelo classificador, sendo definida pela Equação (38).

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN} \tag{38}$$

3.4.2.2 Precisão

A precisão representa o quanto as predições de classe positiva eram realmente positivas. A Equação (39) é utilizada para calcular a precisão.

$$\text{Precisão} = \frac{TP}{TP + FP} \tag{39}$$

3.4.2.3 Sensibilidade

A sensibilidade, também conhecida como *recall* ou *True Positive Rate* (TPR), é uma métrica que representa o quanto as amostras da classe positiva foram classificadas corretamente como positiva pelo classificador. O TPR é definido pela Equação (40).

$$TPR = \frac{TP}{TP + FN} \tag{40}$$

3.4.2.4 F1-Score

O *F1-Score* é uma medida que combina a precisão e a sensibilidade, calculando a média harmônica entre elas. Por ser uma média harmônica, caso a precisão ou a sensibilidade seja baixa, o *F1-Score* também será, ou seja, um modelo com um *F1-Score* alto é um modelo com uma alta qualidade de predição (precisão alta) e capacidade de recuperar corretamente os exemplos de uma classe de interesse (sensibilidade). O *F1-Score* é definido pela Equação (41).

$$F1 - Score = \frac{2TP}{2TP + FP + FN} \tag{41}$$

3.4.3 Entropia Cruzada

A *Cross-entropy*, ou entropia cruzada, é uma métrica que mede a entropia relativa entre duas distribuições de probabilidade. Ela pode ser utilizada para definir uma função de perda em problemas de otimização e de aprendizado de máquina, sendo definida pela Equação (42) (BISHOP, 2006).

$$CE = -\sum_{i}^{N} y \log \hat{y} + (1 - y) \log(1 - \hat{y}), \tag{42}$$

onde y é a saída esperada e $\hat{y}$ é a saída estimada.

3.5 VALIDAÇÃO CRUZADA *K-FOLD*

O método de validação cruzada *k-Fold* é uma técnica utilizada em aprendizado de máquina para avaliar a habilidade de generalização de um modelo de predição, ou seja, o quanto um modelo treinado por uma quantidade limitada de amostras consegue predizer dados que não foram utilizados no treinamento (KOHAVI et al., 1995).

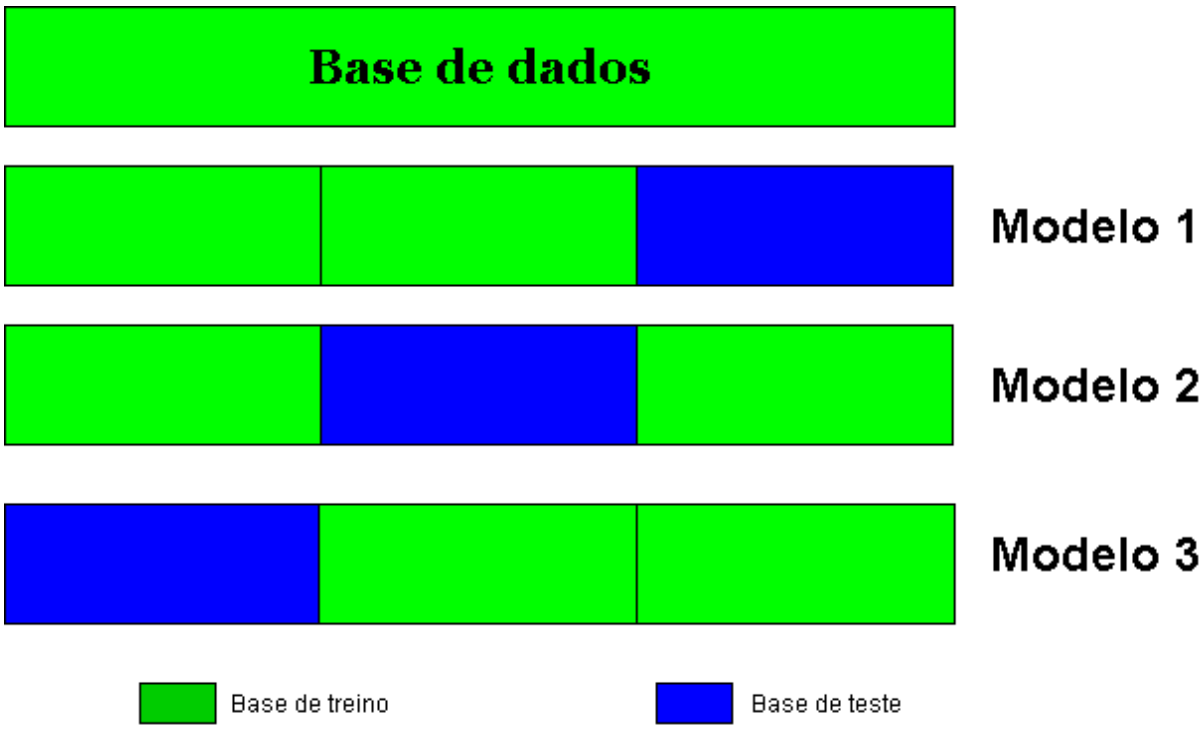
O conceito principal das técnicas de validação cruzada é a divisão da base de dados do problema em subconjuntos mutuamente exclusivos, utilizando alguns dos subconjuntos para treinar o modelo e o restante para a validação do modelo treinado. Por causa de sua simplicidade e por gerar modelos menos enviesados, o *k-Fold* tornou-se um método popular para a validação de modelos de aprendizado de máquina.

O procedimento de validação cruzada *k-fold* é definido pelas seguintes etapas:

a) Embaralhar a base de dados aleatoriamente.
b) Dividir a base de dados em k conjuntos.
c) Para cada conjunto:
   - Selecionar o conjunto atual como base de validação.
   - Selecionar o resto dos conjuntos como base de treinamento.
   - Treinar um modelo com a base de treinamento e avaliá-lo com a base de testes.
   - Armazenar os resultados obtidos pela avaliação do modelo.
d) Calcular a precisão do modelo a partir dos resultados obtidos de cada conjunto.

A Figura 14 apresenta um exemplo da divisão de uma base de dados utilizando o método *k-fold* com o valor de $k = 3$.

Figura 14 – Exemplo da divisão *k-fold* com $k = 3$.



Fonte: Autor.

4 METODOLOGIA

Neste capítulo será apresentada a metodologia desse trabalho para avaliar a utilização de algoritmos de otimização NI para o treinamento de redes neurais artificiais. Inicialmente, será apresentada a ideia geral e, em seguida, cada etapa será descrita com mais detalhes.

A motivação de otimizar os pesos da rede neural veio da revisão feita no Capítulo 2, que mostra, pela Figura 4 (Seção 2.8), a quantidade de trabalhos que tiveram esse objetivo, incluindo trabalhos pioneiros em aplicações com redes neurais até trabalhos recentes, indicando que é uma área que atrai o interesse dos acadêmicos pelas possíveis aplicações e experimentos que ainda podem ser feitos.

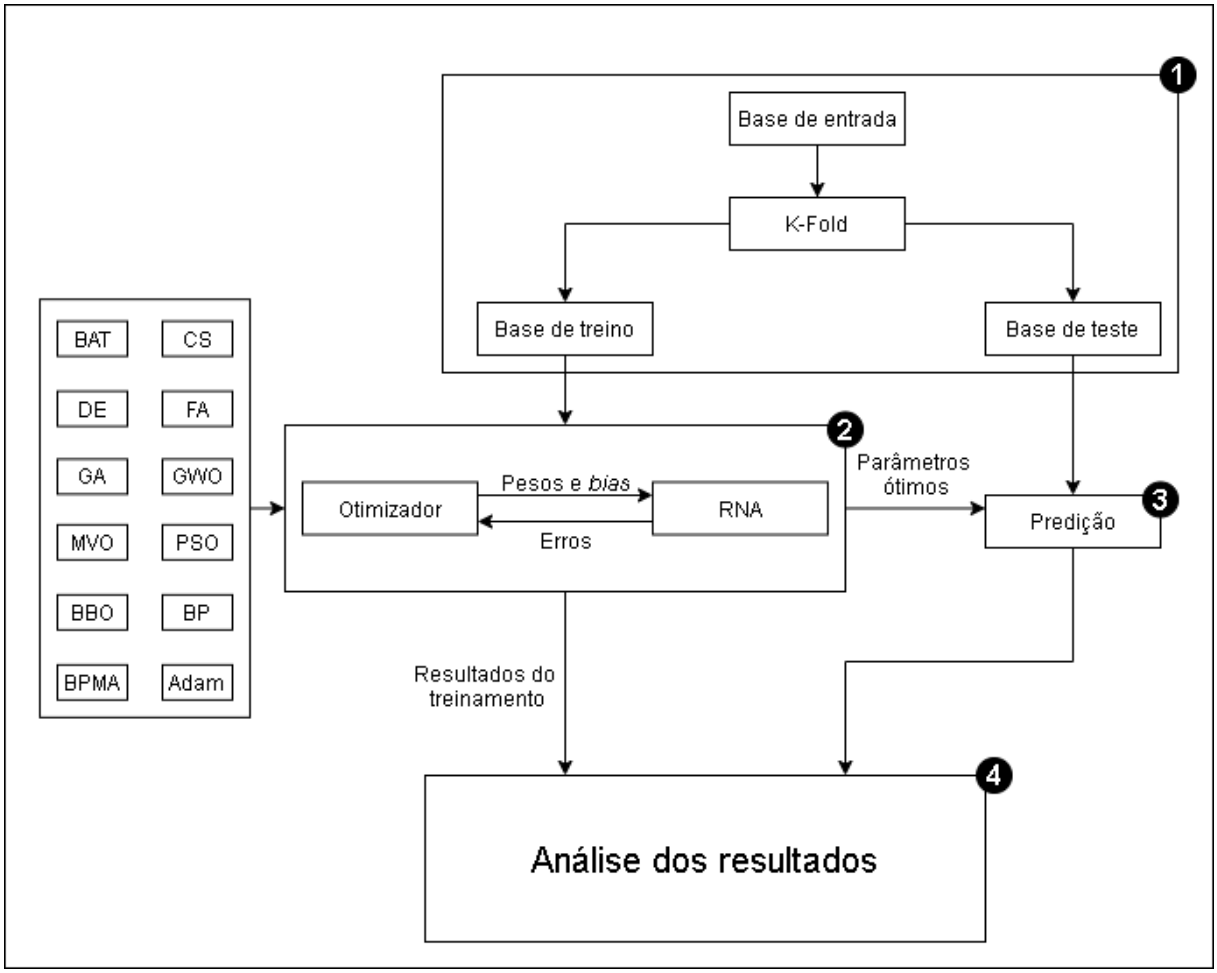
O fluxograma da metodologia é apresentado na Figura 15. Na primeira etapa, é feita a leitura da base de dados dada como entrada, que é separada em amostras de treinamento e amostras de teste. Em seguida, a base de treinamento é inserida na população do algoritmo de otimização selecionado, onde as soluções candidatas do algoritmo são instâncias de redes neurais com diferentes valores de pesos e vieses gerados aleatoriamente pelo algoritmo. Essas redes são avaliadas e devolvem um valor de erro, que representa a aptidão da solução candidata, afim de fazer com que o algoritmo melhore a busca pela solução ótima nas próximas gerações. Esse processo se repete por um número máximo de gerações, retornando os pesos e vieses da solução que possui o melhor valor de aptidão e os resultados obtidos na base de treinamento, para cada um dos algoritmos de otimização utilizados nesse trabalho. As redes neurais com os parâmetros ótimos de cada um dos algoritmos são então testadas, recebendo como entrada as amostras de testes da base de dados para serem classificadas e gerando resultados a partir da classificação feita. Na última etapa é feita a análise dos resultados gerados pelo treinamento da rede e pela predição da base de testes, sendo comparado o desempenho dos algoritmos entre si.

4.1 BASE DE DADOS

Foram selecionadas 9 bases de dados extraídas por outros trabalhos que a utilizaram, estando disponíveis no UCI *Machine Learning Repository*[1]. As bases são referentes a problemas variados de classificação binária, possuindo apenas duas classes, tendo suas características descritas na Tabela 2. Para selecionar as bases, foi considerado o levantamento bibliográfico feito no Capítulo 2 e o gráfico da Figura 3 (Seção 2.8), sendo considerado como critérios: as bases serem de classificação binária; terem sido utilizadas por diversos trabalhos encontrados na pesquisa bibliográfica feita; terem uma pequena quantidade de amostras, com todas elas terem menos de 1000 amostras, e algumas delas possuindo menos de 500 amostras, limitando a quantidade de dados disponíveis para o treinamento do classificador, dificultando o aprendizado final. Além disso, foram consideradas duas bases de dados mais recentes, por possuírem carac-

[1] https://archive.ics.uci.edu/ml/index.php

Figura 15 – Fluxograma da metodologia proposta neste trabalho.



Fonte: Autor.

terísticas assemelham às outras bases selecionadas, sendo elas a *Cervical Cancer* e *Early stage Diabetes Risk*.

Tabela 2 – Bases de dados utilizadas no trabalho

Nome	Amostras	Atributos	Fonte
Blood Transfusion Service Center	748	5	YEH; YANG; TING, 2009
Breast Cancer Wisconsin	569	32	STREET et al., 1993
Cervical Cancer	72	19	MACHMUD; WIJAYA et al., 2016
Early stage Diabetes Risk	520	17	ISLAM et al., 2020
Heart Disease	303	14	DETRANO et al., 1989
Liver Disorders	345	7	MCDERMOTT; FORSYTH, 2016
Parkinsons	197	23	LITTLE et al., 2007
Pima Indians Diabetes	768	8	SMITH et al., 1988
Vertebral Column	310	6	ROCHA NETO et al., 2011

4.1.1 Configuração das bases de dados

Antes de utilizar as bases de dados como entrada da metodologia, é necessário dividi-las em amostras de treinamento e amostras de teste. Para isso, é utilizado o método de validação cruzada *k-Fold*, descrito na Seção 3.5, com o intuito de evitar possíveis vieses que podem acontecer durante a validação, dependendo da forma com que a base é dividida.

O *k-Fold* é aplicado em cada uma das 9 bases de dados selecionadas, separando as amostras em k grupos diferentes, que são embaralhadas entre amostras de treinamento e amostras de testes com sucessivas iterações. Pelo fato das bases possuírem uma amostra pequena, onde nenhuma delas possui mais do que 1000 amostras, é adotado o valor de $k = 3$ para ter amostras de treinamento e de teste o suficiente para que seja possível validar a classificação realizada.

Cada uma das bases apresentadas anteriormente é selecionada como entrada, e então dividida em 3 conjuntos de treinamento e 3 conjuntos de teste. Os 3 conjuntos de treino são utilizados como entrada da rede neural para a etapa de treinamento, enquanto os 3 conjuntos de teste são armazenados para a terceira etapa da metodologia.

4.2 TREINAMENTO DA REDE NEURAL

Após receber a base de treinamento como entrada, é feito o treinamento de uma rede neural MLP. O treinamento da rede é feito utilizando algoritmos de otimização meta-heurísticos NI para encontrar os valores dos pesos e viés que otimizem os resultados da rede neural.

Foram selecionados 9 algoritmos de otimização NI, sendo eles apresentados a seguir:

a) GA - Algoritmo de otimização baseado na evolução biológica.
b) BAT - Algoritmo de otimização baseado no comportamento de localização sonora dos morcego.
c) BBO - Algoritmo de otimização baseado no comportamento de migração de espécies em *habitats* biogeográficos.
d) CS - Algoritmo que se baseia no comportamento do pássaro cuco combinado com vôos de lévy.
e) DE - Algoritmo evolutivo que gera novas soluções a partir da diferença entre soluções existentes.
f) FFA - Algoritmo baseado no comportamento de atração entre os vaga-lumes pela luminosidade.
g) GWO - Algoritmo inspirado no comportamento de caça coletiva de um bando de lobos cinzentos.
h) MVO - Inspirado em conceitos físicos de multiverso que interagem entre si para alcançar a solução ótima.
i) PSO - Algoritmo de otimização baseado no comportamento de pássaros em busca de alimentação.

O funcionamento de cada um dos algoritmos de otimização NI utilizados nesse trabalho estão descritos na Seção 3.1.

Além disso, também foram considerados algoritmos de treinamento tradicionais para serem comparados, sendo eles o BP, BPMA e o *Adam*, apresentados na Seção 3.2. O BP foi selecionado pelo fato de ter sido comparado pela maioria dos trabalhos apresentados no Capítulo 2. O BPMA foi considerado por ser uma melhoria do algoritmo BP e também ter sido utilizado para comparação em alguns trabalhos. No caso do *Adam*, ele foi selecionado por ser um algoritmo de treinamento de rna atual e por mostrar bons resultados em problemas de alta complexidade, como nos trabalhos de An et al. (2020) e Mehta, Paunwala e Vaidya (2019).

Para cada um dos algoritmos, é feita a leitura de cada conjunto de treinamento recebido para ser utilizado como entrada da rede neural. Em seguida, o treinamento é iniciado seguindo o método descrito na Seção 3.3. A cada geração do treinamento, o otimizador retorna um conjunto de pesos e viés que são utilizados para calcular o erro da rede. A rede retorna o erro obtido para o otimizador calcular novos pesos e viés para retornar novamente para a rede, e assim sucessivamente. Como função objetivo dos algoritmos de otimização, são consideradas as funções MSE, Acurácia, *F1-Score* e Entropia Cruzada, apresentadas na Seção 3.4. O MSE e a acurácia foram utilizadas por serem as métricas mais utilizadas de acordo com a pesquisa bibliográfica feita, como mostrado na Figura 2. A Entropia Cruzada foi considerada por ser utilizada para resolver problemas de classificação mais desafiadores, que envolvem dados com mais características, como nos trabalhos de Phan e Yamamoto (2020) e de Takeda, Yoshida e Muneyasu (2020). O *F1-Score* foi considerado por ser uma função não derivável, que indica a precisão do modelo, ou seja, a quantidade de amostras classificadas corretamente, e a robustez, que indica o quanto o modelo deixou de classificar corretamente.

Após a finalização do treinamento, os resultados obtidos em relação à taxa de erro obtido durante o treino e a taxa de convergência são armazenados para serem analisados posteriormente. Os pesos e vieses ótimos são então passados para a próxima etapa da metodologia.

4.2.1 Configuração dos algoritmos NI

Os algoritmos NI selecionados possuem diversos parâmetros que são utilizados para a otimização. A seleção desses parâmetros foi feita baseada na recomendação dos próprios autores dos algoritmos, nos trabalhos da Seção 3.1, e também foi considerado os parâmetros utilizados no trabalho de Hossam Faris et al. (2016a). Os parâmetros dos algoritmos GWO e MVO não foram especificados pois eles seguem as equações detalhadas nas Seções 5 e 3.1.8. Os detalhes dos parâmetros selecionados estão na Tabela 3. Além disso, para cada algoritmo, foram selecionados uma população de tamanho 10 e de tamanho 50, que foram executadas por 150 gerações.

Tabela 3 – Parâmetros dos algoritmos NI.

Algoritmo	Parâmetro	Valor
GA	Seleção	Roleta
	Crossover	Ponto Único (probabilidade=1)
	Probabilidade de mutação	0.01
	Elitismo	2
BAT	A_0	0.5
	r	0.5
	α	0.95
	γ	0.05
BBO	Probabilidade de mutação	0.01
	Elitismo	2
CS	α	0.25
DE	F	0.5
	CR	0.7
FFA	α	0.5
	β_{min}	0.2
	γ	1
PSO	v_{max}	6
	$c1$ e $c2$	2
	w_{min}	0.2
	w_{max}	0.9

4.2.2 Configuração dos algoritmos tradicionais

Os algoritmos de treinamento que não são NI, também possuem parâmetros para realizar a otimização da rede. Para a seleção dos parâmetros, foi considerado os parâmetros indicados no artigo original do algoritmo, apresentados na Seção 3.2. Como função objetivo, foi adotada a função MSE para os algoritmos BP e BPMA por ser a métrica mais utilizada na maioria dos trabalhos apresentados na Seção 2, e para o *Adam* foi adotada a métrica Entropia Cruzada, seguindo o mesmo padrão do artigo em que ele foi apresentado. Os parâmetros são detalhados na Tabela 4. Para ter uma quantidade de iterações equivalente aos algoritmos NI, o número de iterações adotado para os algoritmos tradicionais foram de 1500 e 7500, sendo equivalente ao número de iterações multiplicado pelo tamanho da população adotados para os NI.

4.2.3 Configuração da rede neural

A rede neural utilizada é uma rede *feed-forward* MLP, seguindo o modelo apresentado na Seção 3.2.1. A camada de entrada possui N neurônios, onde N é o número de atributos da base de treinamento. Na camada escondida, quanto menor o número de neurônios, mais difícil é a aproximação da função de classificação, e quanto mais neurônios, maior é a probabilidade da rede não generalizar a classificação para dados não encontrados na base de treinamento. Para mitigar esse problema, é adotada uma camada escondida com $(2N) + 1$ neurônios, seguindo a

Tabela 4 – Parâmetros dos algoritmos tradicionais para o treinamento de uma rede neural.

Algoritmo	Parâmetro	Valor
BP	α	0.01
	Função objetivo	MSE
BPMA	α	0.01
	c	0.9
	Função objetivo	MSE
Adam	α	0.001
	β_1	0.9
	β_2	0.9999
	ϵ	$1e-08$
	Função objetivo	Entropia Cruzada

recomendação dos autores Krková (1991) e Hegazy, Fazio e Moselhi (1994), sendo essa abordagem também adotada em trabalhos semelhantes, como os de Wdaa e Sttar (2008), Mirjalili, Mirjalili e Lewis (2014a), Yamany et al. (2015), Leema, Nehemiah e Kannan (2016) e Aljarah, Faris e Mirjalili (2018). Para a camada de saída é adotado apenas 1 neurônio. Assim, as estruturas das redes MLP utilizadas para a classificação das bases de dados, citadas na Seção 4.1, são apresentadas na Tabela 5. Além disso, também é apresentado um apelido para simplificar a referência às bases de dados.

Tabela 5 – Estruturas das redes para cada base de dados reais.

Nome	Atributos	Estrutura MLP
cancer	32	32-65-1
cervical	19	19-38-1
diabetes	8	8-17-1
diabetes2	17	17-34-1
heart	14	14-29-1
liver	7	7-15-1
parkinsons	23	23-47-1
transfusion	5	5-11-1
vertebral	6	6-13-1

Como função de ativação, são adotadas as funções Tanh, sigmoide e ReLU, descritas na Seção 3.2.4, sendo treinado um modelo de rede para cada uma das funções. Foram selecionadas essas funções pela popularidade de serem utilizadas em RNAs para problemas de classificação.

4.3 PREDIÇÃO

A etapa de Predição recebe os 3 conjuntos de testes selecionados para cada base, seguindo o procedimento descrito na Seção 4.1.1. Cada conjunto é passado como entrada para os modelos de redes que tiveram os seus pesos e vieses otimizados por cada um dos algoritmos NI na etapa anterior. Assim, é feita a avaliação do desempenho dos modelos, gerando a saída

obtida ao receber o conjunto de teste como entrada. A partir das saídas obtidas pelas redes, são calculadas as seguintes métricas de avaliação, formalizadas na Seção 3.4:

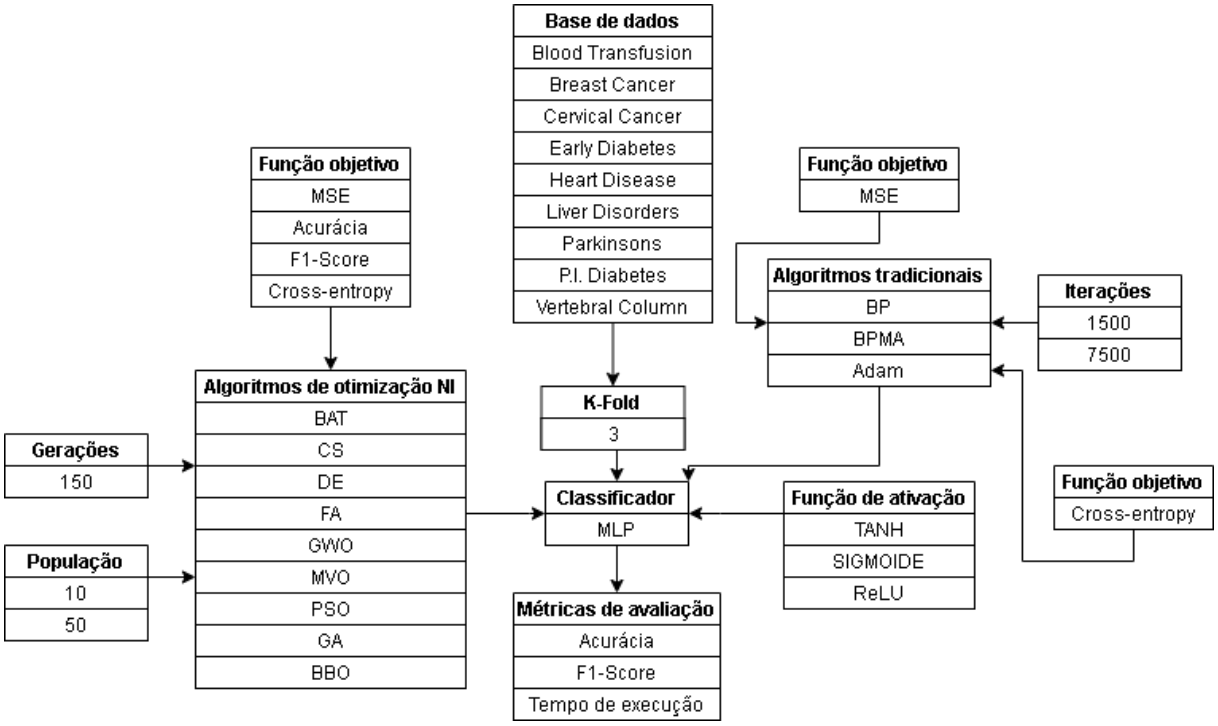
a) Acurácia;
b) F1-Score;
c) Tempo de execução em segundos

Após o cálculo dessas métricas, para cada um dos conjuntos de teste da base, foi calculada a média de cada delas, afim de totalizar o resultado dos modelos treinados por cada um dos algoritmos. Os resultados totalizados são então passados para serem analisados na próxima etapa.

4.4 ANÁLISE DOS RESULTADOS

Para a análise dos resultados, foram consideradas diversas métricas, bases de dados e algoritmos de otimização NI para construir diferentes modelos de rede neural e que são comparados entre si. Esses parâmetros são especificados na Figura 16. A quantidade de parâmetros a serem variados podem então ser sumarizados da seguinte forma:

Figura 16 – Ilustração dos parâmetros específicos levados em consideração na metodologia.



Fonte: Autor.

a) Algoritmos de otimização NI: 9.
b) Algoritmos de otimização tradicionais: 3.
c) Funções objetivo: 4.

d) Funções de ativação: 3.
e) Bases de dados: 9.
f) Métricas de avaliação: 3.

Dessa forma, totalizando o número de combinações de algoritmos de otimização NI, com funções objetivo e funções de ativação diferentes, somado com o número de algoritmos tradicionais utilizando funções de ativação diferentes, obtém-se um total 117 modelos que foram testados em 9 bases de dados diferentes e avaliados por 3 métricas diferentes.

4.5 IMPLEMENTAÇÃO

Todos os experimentos foram implementados utilizando a linguagem *Python* 3.7.2, sendo utilizado o *framework EvoloPY*, apresentado por H. Faris et al. (2016), para aplicar os algoritmos de otimização NI, a biblioteca *SciPy* (JONES; OLIPHANT; PETERSON, 2001) para operações matemáticas e computação científica, e a biblioteca *Scikit-Learn* (PEDREGOSA et al., 2011) para o cálculo de métricas e a utilização do algoritmo *Adam*. Além disso, para a construção de gráficos foi utilizada a biblioteca *Matplotlib*, proposta por Hunter (2007).

5 RESULTADOS E DISCUSSÃO

Nesse capítulo serão apresentados os resultados gerados a partir da metodologia proposta no Capítulo 4 e discutido os pontos importantes que foram observados a partir deles, considerando o impacto do uso das diferentes funções de ativação em conjunto com diferentes funções de custo para cada um dos 9 algoritmos NI utilizados, sendo eles: GA, BAT, BBO, CS, DE, FFA, GWO, MVO e PSO, e para cada um dos 3 algoritmos tradicionais, sendo eles: o BP, o BPMA e o *Adam*, comparando o desempenho entre eles.

Para isso, foram feitas análises de 4 cenários diferentes. No primeiro cenário, foram selecionados os resultados das melhores variações de cada algoritmo, ou seja, o modelo de cada algoritmo com a combinação de diferentes funções de custo, funções objetivo, tamanhos de população que obteve os melhores valores de acurácia, *F1-Score* e tempo de execução, em cada base com o intuito de identificar os algoritmos que se destacaram em bases específicas. No segundo cenário, é calculada a acurácia média, o *F1-Score* médio e o tempo de execução médio global dos algoritmos para serem avaliados de forma geral e descobrir qual deles obteve o melhor desempenho para o problema de treinamento de redes neurais artificiais. Em terceiro, é feita a avaliação das funções de ativação por meio da média global da acurácia, do *F1-Score* e do tempo de execução obtido pelos modelos que utilizaram diferentes funções de ativação para identificar as diferenças entre elas. Por último, é feita uma análise das diferentes funções de custo a partir das taxas de convergência obtidas pelos modelos que utilizaram cada uma delas. Além disso, também foi calculada a acurácia média, o *F1-Score* médio e o tempo de execução médio global dos modelos que utilizaram diferentes funções de custo para verificar o desempenho de cada uma delas. Os resultados foram gerados a partir da execução em CPU, utilizando 1 núcleo.

5.1 MELHORES VARIAÇÕES EM CADA BASE

Para comparar o desempenho dos algoritmos entre si, foram selecionadas as variações dos algoritmos que tiveram a maior acurácia e *F1-Score* para cada uma das 9 bases selecionadas, sendo gerados 3 gráficos de barras para cada uma delas, onde cada barra representa um algoritmo diferente. O primeiro gráfico mede a acurácia obtida por cada algoritmo, o segundo mede o *F1-Score* e o terceiro mede o tempo de execução.

Observando os resultados da base *cancer*, apresentados na Figura 17, nota-se que as melhores variações foram dos algoritmos *Adam* e o BPMA, sendo dois algoritmos de treinamento tradicionais, obtendo resultados de acurácia e *F1-Score* bem próximos dos algoritmos BBO, GWO e MVO, ilustrado nas Figuras 17a e 17b. De forma geral, todos os algoritmos NI, obtiveram uma acurácia e um valor de *F1-Score* acima de 0.9, superando o algoritmo BP nesses termos. No entanto, em relação ao tempo de execução, observado na Figura 17c, os algoritmos NI mostraram-se mais lentos comparado ao *Adam* e o BPMA. Isso aconteceu pois, as melhores

variações do *Adam* e do BPMA obtiveram uma taxa de acurácia e *F1-Score* mais alta utilizando apenas 1500 iterações.

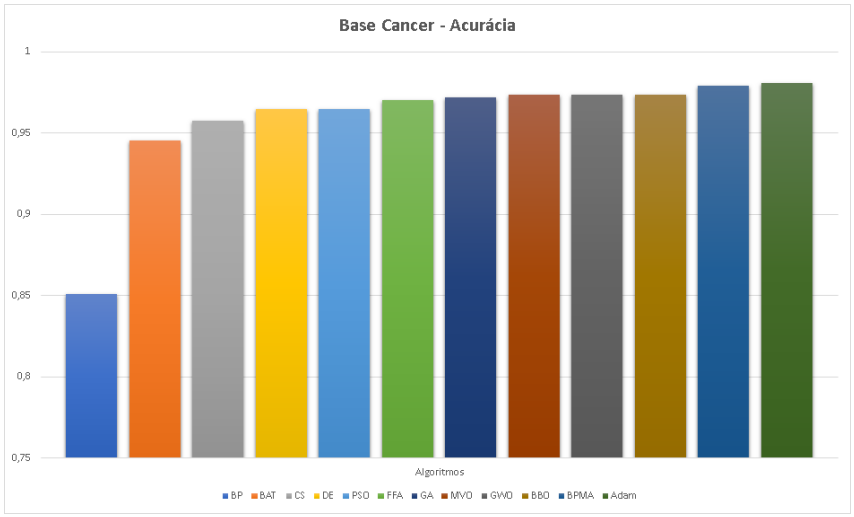
Os resultados da base *cervical*, observados na Figura 18, mostram que o melhor algoritmo em termos de acurácia, *F1-Score* e tempo de execução foi o *Adam*. Um fator que contribuiu para isso foi a quantidade pequena de amostras dessa base, possuindo apenas 72 amostras, mostrando que o algoritmo *Adam* possui vantagem em bases de amostragem limitada. Além disso, ele teve um tempo de execução rápido por utilizar apenas 1500 iterações. Em relação ao restante dos algoritmos NI, pode-se notar que o algoritmo BAT obteve os valores de acurácia e *F1-Score* mais baixos, como mostra as Figuras 18a e 18b, o que indica a sua dificuldade em buscar a solução ótima em relação aos demais algoritmos NI. Além disso, o segundo melhor algoritmo em termos de acurácia e *F1-Score* foi o BBO, superando os algoritmos GWO, DE e GA com um tempo de execução menor, ilustrado na Figura 18c, o que indica que o BBO otimizou melhor os pesos utilizando apenas 10 indivíduos, enquanto os outros algoritmos citados anteriormente utilizaram 50 indivíduos.

Em termos de acurácia, o algoritmo que mostrou-se superior na base *diabetes*, ilustrado na Figura 19a foi o BP, no entanto, considerando o *F1-Score*, observado na Figura 19b, os algoritmos GWO e GA tiveram os maiores valores. Isso significa que o GWO e GA classificam mais corretamente a classe positiva comparados com o BP. No entanto, o GA é mais rápido comparado ao GWO, como mostra a Figura 19c, devido ao fato da sua variação com o maior valor de *F1-Score* ter utilizado uma população de tamanho 10, enquanto a melhor variação do GWO utilizou uma população de tamanho 50. Outro ponto a ser destacado é os resultados obtidos pelo BPMA, onde ele obteve a acurácia mais baixa, o *F1-Score* mais baixo e o tempo de execução mais alto, mostrando que mesmo utilizando 7500 iterações, ele não convergiu e acabou estagnando em um mínimo local.

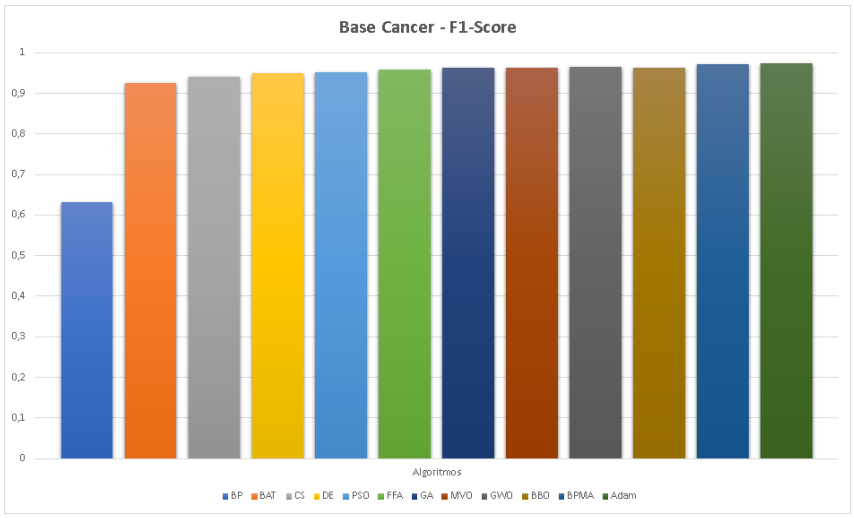
Os resultados da base *diabetes2*, apresentados na Figura 20, indicam que o GWO foi o melhor algoritmo em termos de acurácia e *F1-Score*, ilustrado nas Figuras 20a e 20b, seguido dos três algoritmos de treinamento tradicionais, BP, BPMA, *Adam*. Todavia, em relação ao tempo de execução, o GWO demorou 150 segundos, enquanto o BP levou 20 segundos, o BPMA 4 segundos e o *Adam* precisou de 1 segundo. Nesses casos onde o tempo dos algoritmos tradicionais ficaram bem abaixo do GWO, a justificativa é devido à quantidade de iterações necessárias para cada algoritmo. Enquanto o GWO demandou de 50 indivíduos, com cada um deles executando por 150 iterações, os algoritmos tradicionais executaram apenas 1500 iterações para encontrar os seus valores ótimos. Além disso, o BAT obteve a quinta posição em termos de acurácia e *F1-Score*, sendo o melhor em relação ao restante dos algoritmos NI nesses termos, com um tempo de execução menor mas utilizando o mesmo tamanho da população que eles, indicando que o BAT alcançou uma taxa de convergência melhor na base *diabetes2*.

Observando os resultados da base *heart*, apresentados na Figura 21, nota-se que quase todos os algoritmos tiveram uma acurácia próxima de 0.8, com exceção do BPMA, que obteve uma acurácia próxima de 0.65 e um valor de *F1-Score* de aproximadamente 0.25, indicando uma

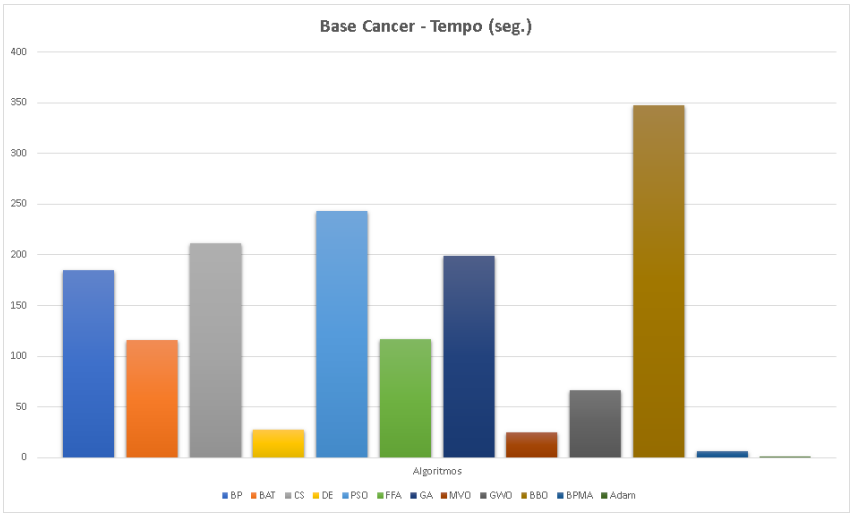
Figura 17 – Gráficos de acurácia, tempo de execução e F1-Score obtidos na base *cancer* - 17a Acurácia obtida na base *cancer* pelos 12 algoritmos de otimização. 17b - F1-Score obtido na base *cancer* pelos 12 algoritmos de otimização. 17c - Tempo de execução obtido na base *cancer* pelos 12 algoritmos de otimização.



(a)



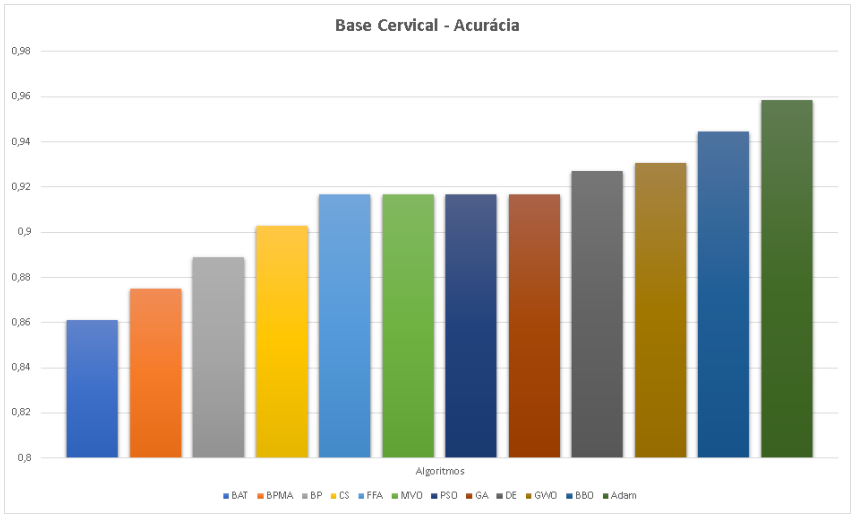
(b)



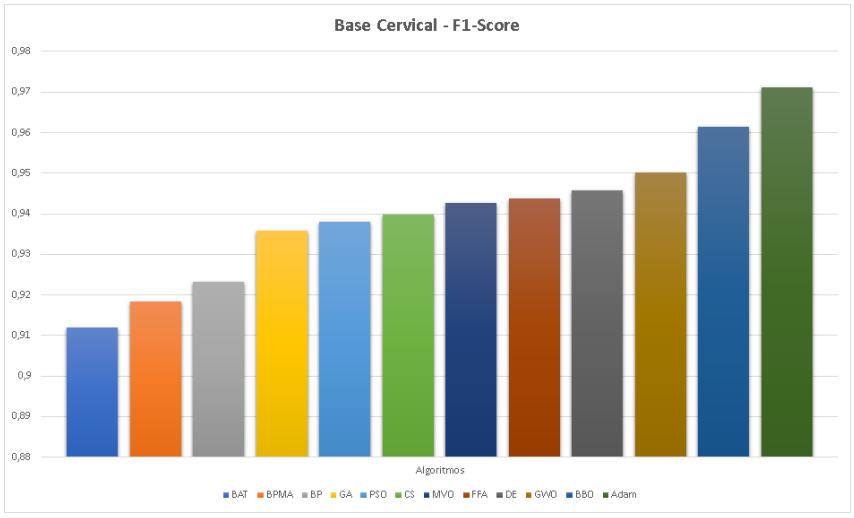
(c)

Fonte: Autor.

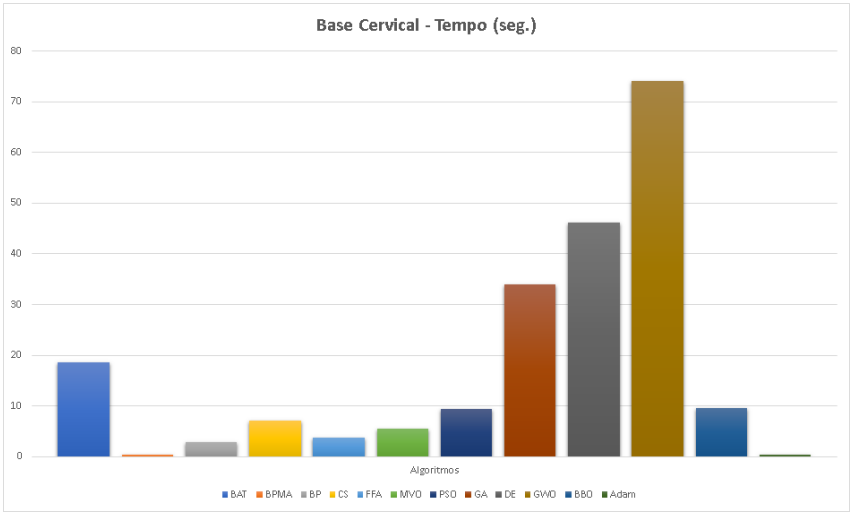
Figura 18 – Gráficos de acurácia, tempo de execução e F1-Score obtidos na base *cervical* - 18a Acurácia obtida na base *cervical* pelos 12 algoritmos de otimização. 18b - F1-Score obtido na base *cervical* pelos 12 algoritmos de otimização. 18c - Tempo de execução obtido na base *cervical* pelos 12 algoritmos de otimização.



(a)



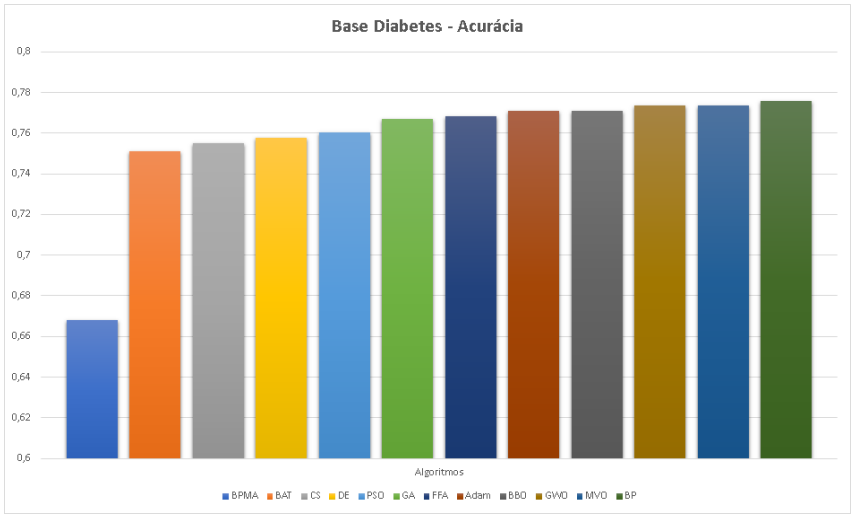
(b)



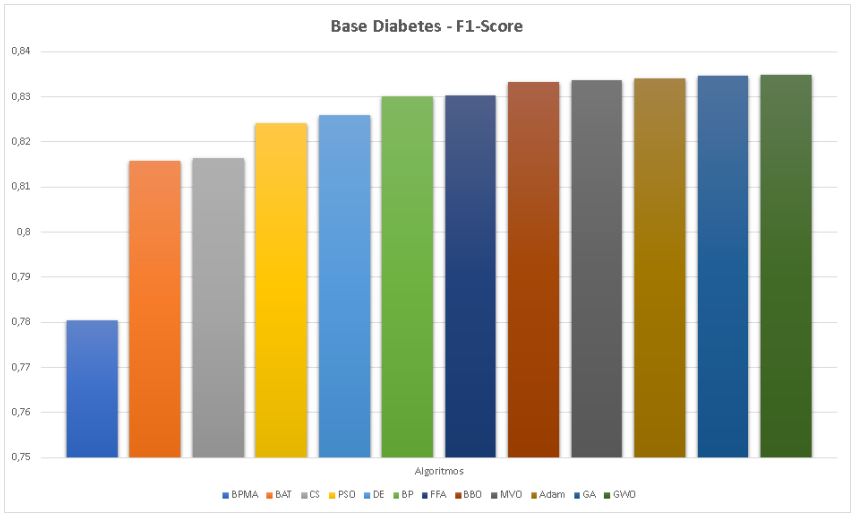
(c)

Fonte: Autor.

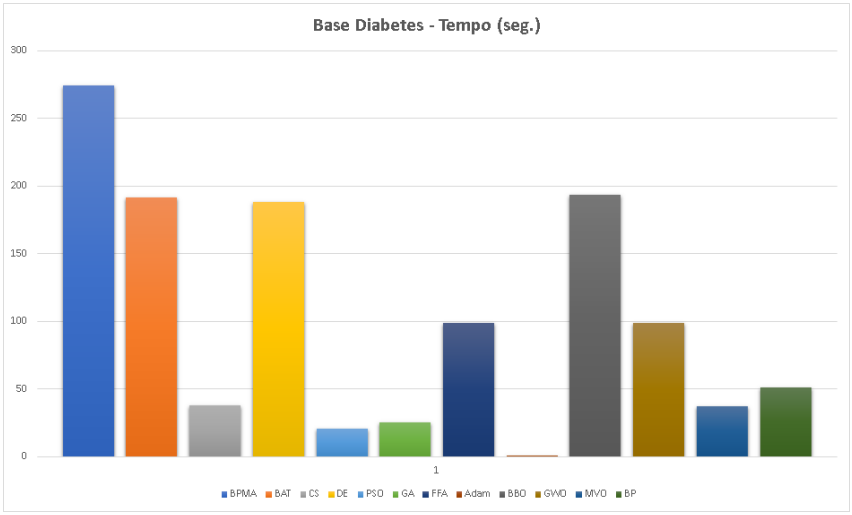
Figura 19 – Gráficos de acurácia, tempo de execução e F1-Score obtidos na base *diabetes* - 19a Acurácia obtida na base *diabetes* pelos 12 algoritmos de otimização. 19b - F1-Score obtido na base *diabetes* pelos 12 algoritmos de otimização. 19c - Tempo de execução obtido na base *diabetes* pelos 12 algoritmos de otimização.



(a)



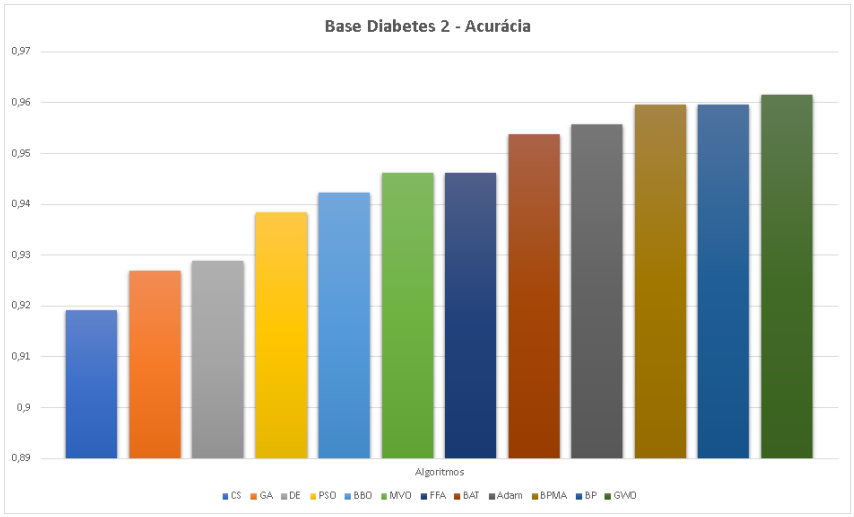
(b)



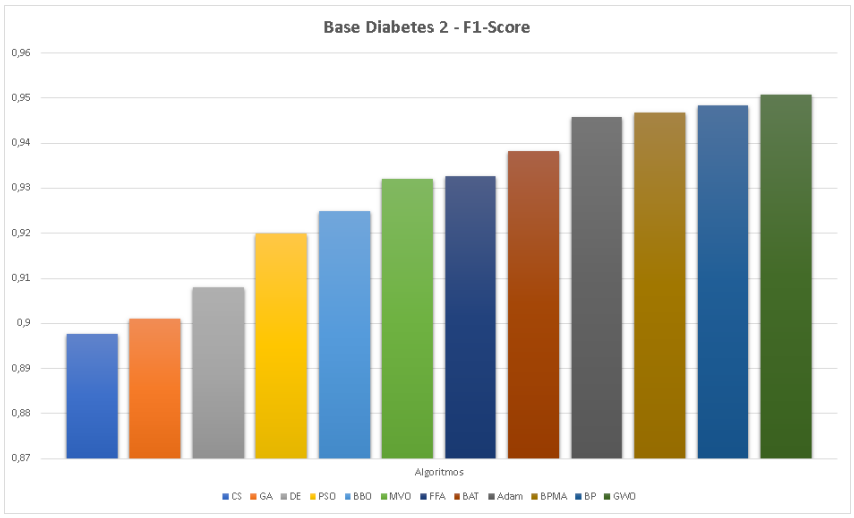
(c)

Fonte: Autor.

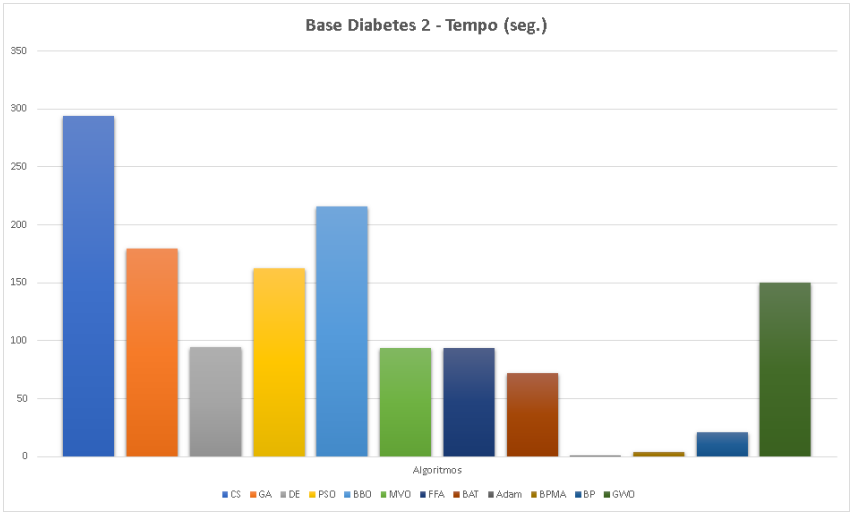
Figura 20 – Gráficos de acurácia, tempo de execução e F1-Score obtidos na base *diabetes2* - 20a Acurácia obtida na base *diabetes* pelos 12 algoritmos de otimização. 20b - F1-Score obtido na base *diabetes2* pelos 12 algoritmos de otimização. 20c - Tempo de execução obtido na base *diabetes2* pelos 12 algoritmos de otimização.



(a)



(b)



(c)

Fonte: Autor.

dificuldade do algoritmo em classificar corretamente amostras da classe positiva. Além disso, o algoritmo CS obteve o maior valor de acurácia e de *F1-Score*, observado nas Figuras 21a e 21b, superando o algoritmo *Adam* por uma diferença de aproximadamente 0.01. Além disso, o CS foi 50 segundos mais rápido que o *Adam* como observado na Figura 21c. O algoritmo *Adam* foi mais lento nesse caso pelo fato da sua melhor variação ter encontrado os valores ótimos com 7500 iterações.

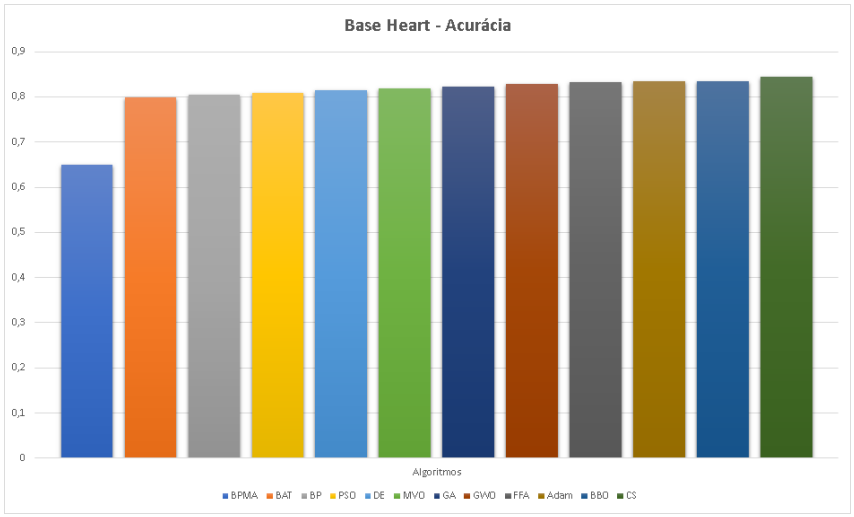
Analisando os resultados da base *liver*, o primeiro ponto a ser destacado é em relação à acurácia, ilustrada na Figura 22a, onde a acurácia geral dos algoritmos foi baixa em relação às outras bases, com a maior acurácia obtida foi de aproximadamente 0.75 pelo algoritmo MVO. Baseado nisso, pode-se dizer que a base *liver* é uma das bases de maior dificuldade de classificação. Dito isso, é interessante notar que o algoritmo *Adam* ficou entre os 4 piores algoritmos em termos de acurácia e *F1-Score*, tendo uma diferença de aproximadamente 0.06 e 0.1 da acurácia e *F1-Score* do algoritmo MVO, como mostra a Figura 22b. Além disso, em relação ao tempo de execução, o MVO foi o segundo algoritmo mais rápido, observado na Figura 22c, mesmo utilizando 50 indivíduos para a otimização.

Os resultados obtidos na base *parkinsons* mostraram que o algoritmo com maior acurácia e um tempo de execução de aproximadamente 5 segundos foi o BPMA, observado nas Figuras 23a e 23c, no entanto, o algoritmo MVO superou o BPMA em termos de *F1-Score* por uma pequena diferença de 0.015, ilustrado na Figura 23b, porém, o seu tempo foi 10 vezes mais lento. O BAT foi o pior dos algoritmos em relação à acurácia e *F1-Score*, com um valor de *F1-Score* de aproximadamente 0.45, indicando uma dificuldade do BAT para classificar corretamente as amostras da classe positiva na base *parkinsons*. Por outro lado, o restante dos algoritmos obteve uma acurácia próxima de 0.88 e um *F1-Score* próximo de 0.7, mostrando que a maioria dos algoritmos de otimização encontraram pontos semelhantes no espaço de busca do problema.

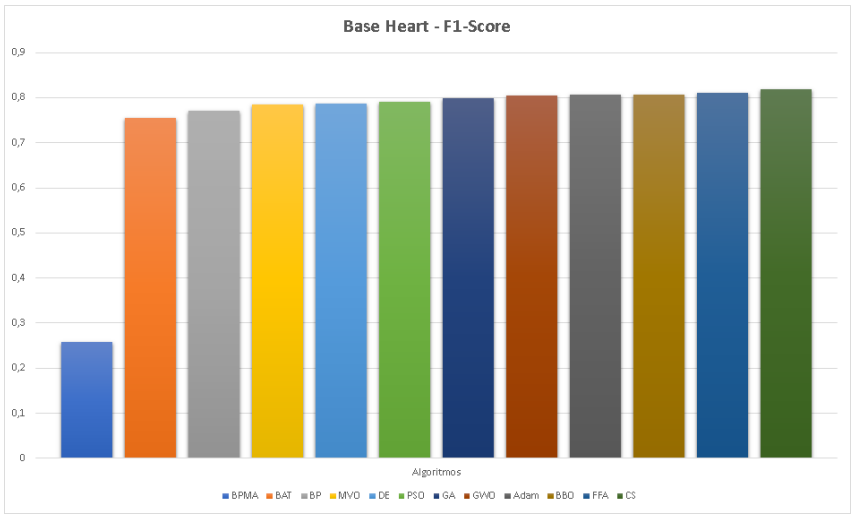
Para a base *transfusion*, foi notado que o algoritmo PSO obteve a melhor acurácia, seguido do GA, como mostra a Figura 24a, enquanto o GA obteve o maior valor de *F1-Score*, seguido do algoritmo PSO, observado na Figura 24b, no entanto, o GA teve um tempo de execução mais rápido, mostrado no gráfico da Figura 24c. Também foi observado que os algoritmos GWO, MVO e CS obtiveram praticamente o mesmo valor de acurácia e um valor de *F1-Score* próximos entre si, porém o GWO foi mais rápido, seguido do MVO em termos de velocidade de execução, com ambos sendo mais rápido do que o algoritmo *Adam*, que apresentou dificuldades em outra base onde a acurácia geral foi abaixo de 0.8, como na base *liver*.

Na base *vertebral*, os algoritmos FFA, BBO, *Adam* e BP obtiveram a maior taxa de acurácia, com um valor de aproximadamente 0.85, como observado na Figura 25a, no entanto, o algoritmo FFA obteve um tempo de execução menor entre os quatro melhores, como mostra o gráfico da Figura 25c, utilizando uma população de tamanho 50, assim como o BBO. Porém, o BBO e o *Adam* obtiveram um valor de *F1-Score* maior comparado ao FFA, ilustrado na Figura 25b. Além disso, o BAT mostrou uma diferença maior de acurácia e *F1-Score* em relação aos outros algoritmos, apresentando dificuldades para convergir para uma solução ótima. Outro

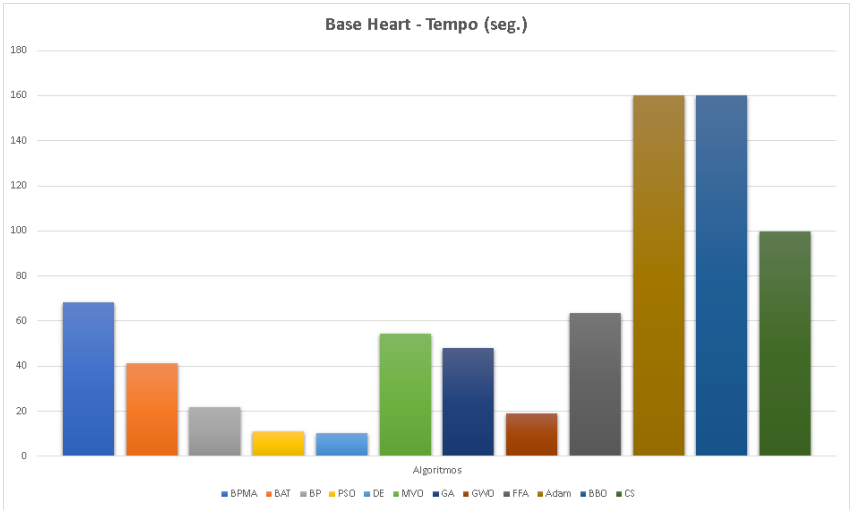
Figura 21 – Gráficos de acurácia, tempo de execução e F1-Score obtidos na base *heart* - 21a Acurácia obtida na base *heart* pelos 12 algoritmos de otimização. 21b - F1-Score obtido na base *heart* pelos 12 algoritmos de otimização. 21c - Tempo de execução obtido na base *heart* pelos 12 algoritmos de otimização.



(a)



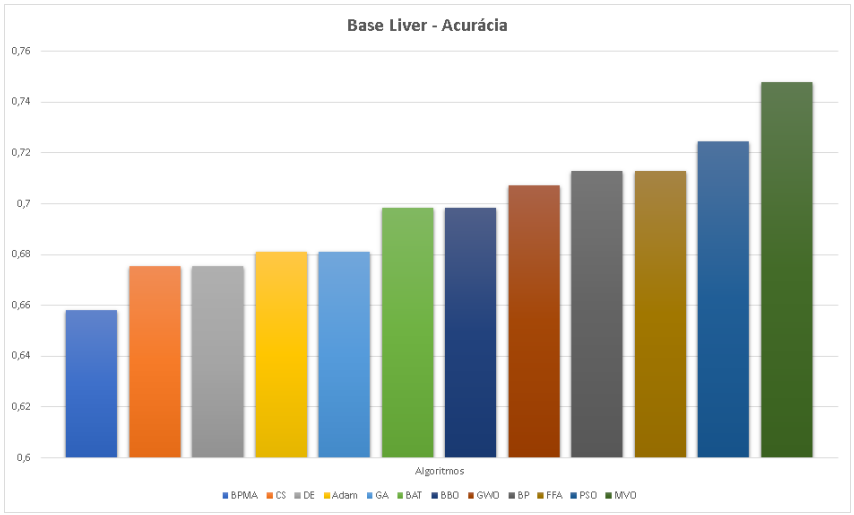
(b)



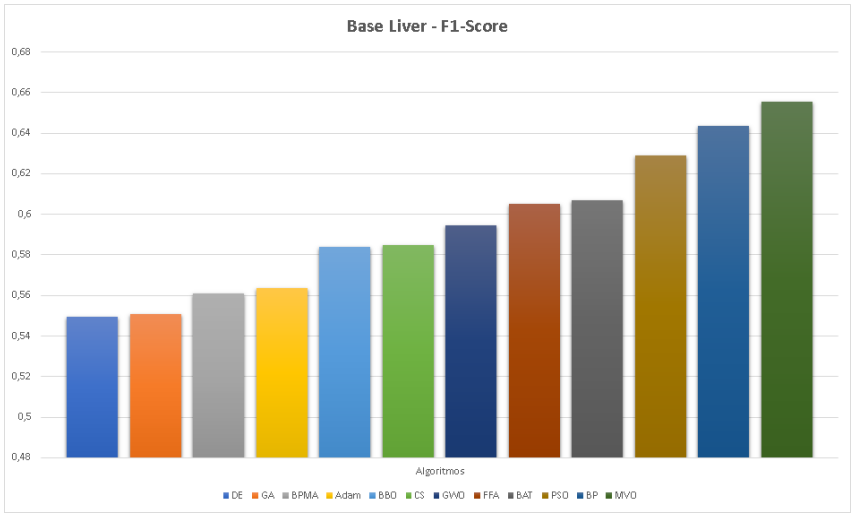
(c)

Fonte: Autor.

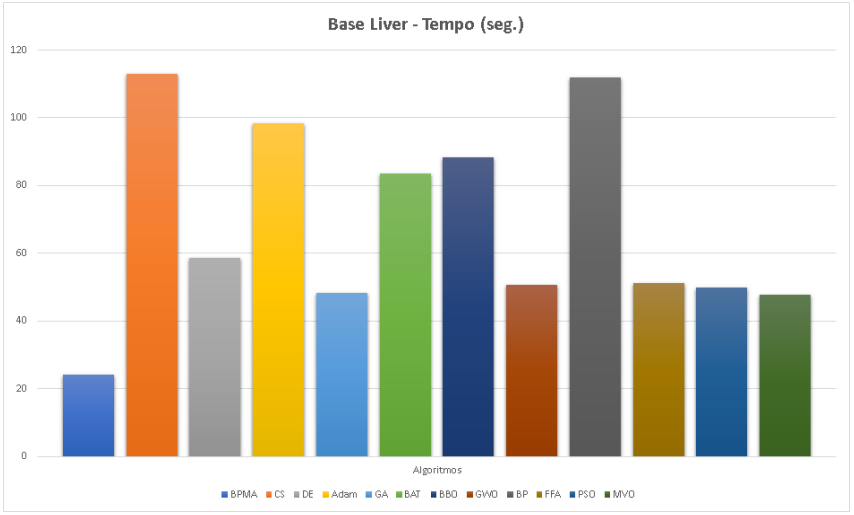
Figura 22 – Gráficos de acurácia, tempo de execução e F1-Score obtidos na base *liver* - 22a Acurácia obtida na base *liver* pelos 12 algoritmos de otimização. 22b - F1-Score obtido na base *liver* pelos 12 algoritmos de otimização. 22c - Tempo de execução obtido na base *liver* pelos 12 algoritmos de otimização.



(a)



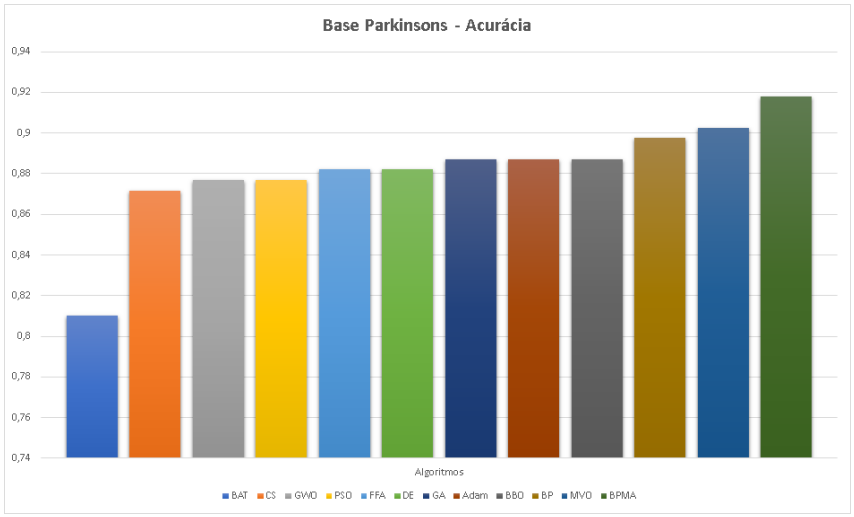
(b)



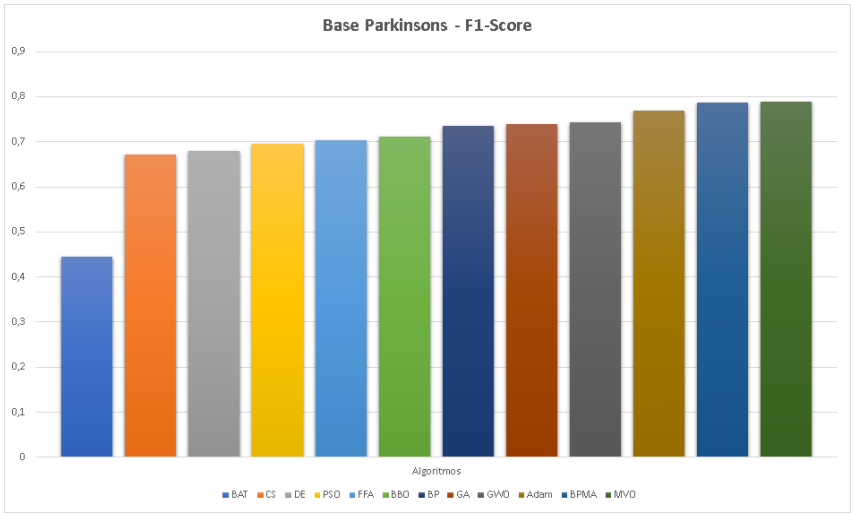
(c)

Fonte: Autor.

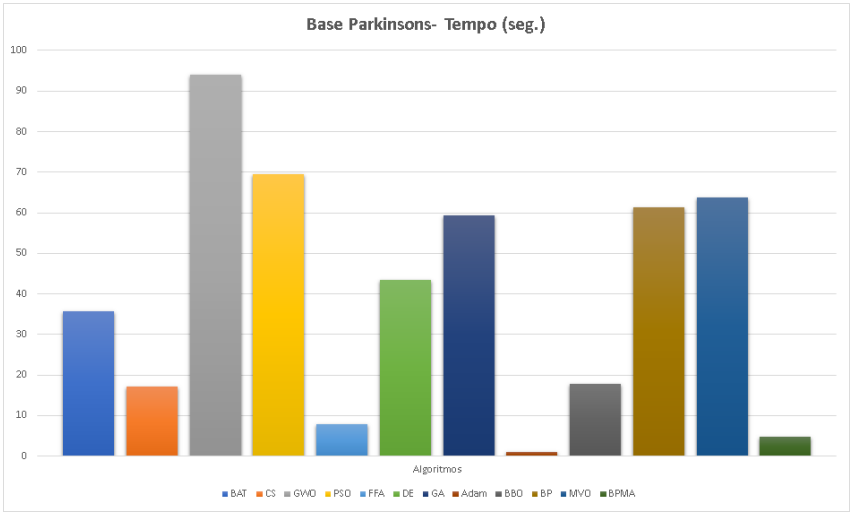
Figura 23 – Gráficos de acurácia, tempo de execução e F1-Score obtidos na base *parkinsons* - 23a Acurácia obtida na base *parkinsons* pelos 12 algoritmos de otimização. 23b - F1-Score obtido na base *parkinsons* pelos 12 algoritmos de otimização. 19c - Tempo de execução obtido na base *parkinsons* pelos 12 algoritmos de otimização.



(a)



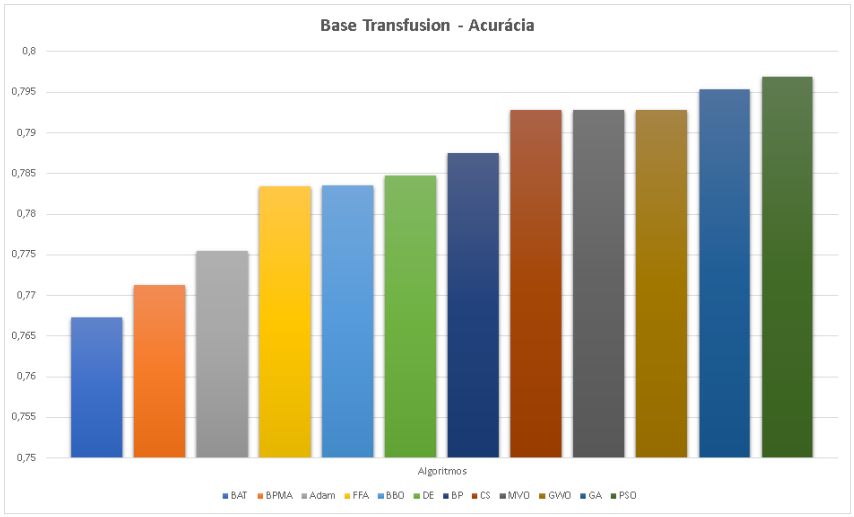
(b)



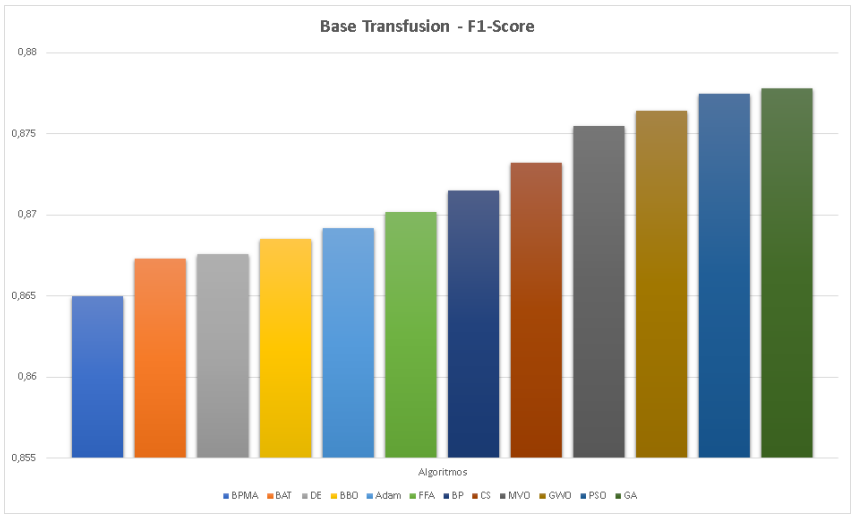
(c)

Fonte: Autor.

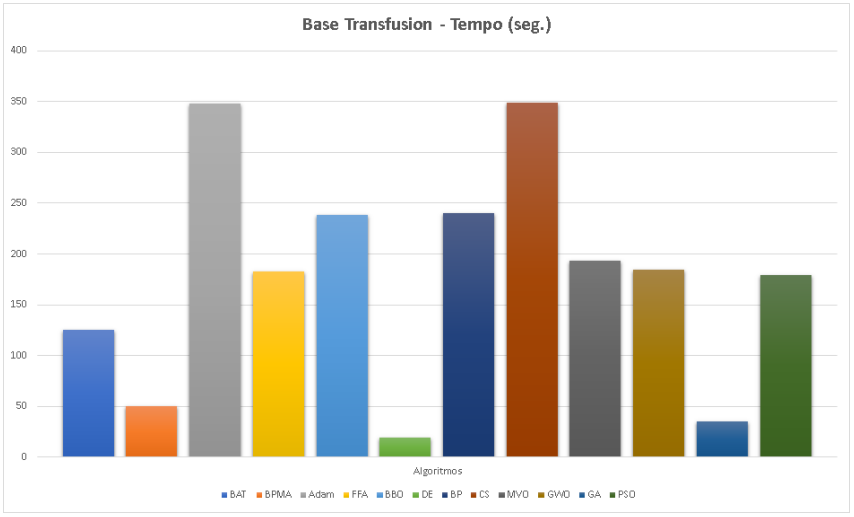
Figura 24 – Gráficos de acurácia, tempo de execução e F1-Score obtidos na base *transfusion* - 24a Acurácia obtida na base *transfusion* pelos 12 algoritmos de otimização. 23b - F1-Score obtido na base *transfusion* pelos 12 algoritmos de otimização. 24c - Tempo de execução obtido na base *transfusion* pelos 12 algoritmos de otimização.



(a)



(b)



(c)

Fonte: Autor.

algoritmo que também se destacou foi o PSO, que obteve uma acurácia semelhante do MVO e teve o menor tempo de execução dentre todos os algoritmos.

5.2 ALGORITMOS INSPIRADOS NA NATUREZA X ALGORITMOS TRADICIONAIS

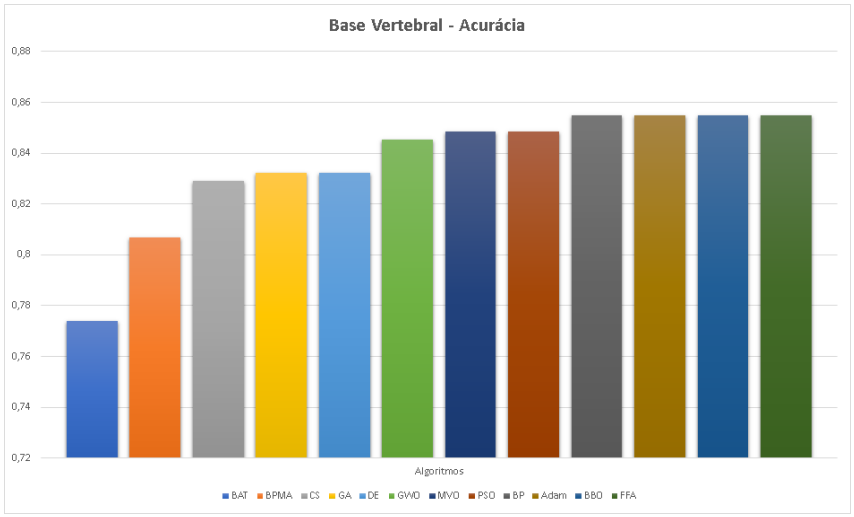
Para comparar o desempenho entre os algoritmos NI e tradicionais, foi calculada a média da acurácia e do tempo de execução para cada um dos algoritmos utilizados, considerando todas as suas variações em todas as bases aplicadas. Além disso, também foi levado em conta os dois tamanhos de população definidos no Capítulo 4. Com essas informações, foi construído um gráfico de dispersão, onde o eixo *x* representa a acurácia média, o eixo *y* o tempo de execução médio e cada marcador representa um algoritmo diferente, para cada um dos tamanhos de população diferentes, sendo apresentados na Figura 26.

Considerando o gráfico da média geral da acurácia e do tempo de execução de cada algoritmo utilizando uma população de tamanho 10 ou 1500 iterações, apresentado na Figura 26a, nota-se que os algoritmos NI superam os algoritmos BP e BPMA em termos de acurácia e tempo de execução, com exceção do BAT, que mostrou-se o pior algoritmo em termos de acurácia nesse cenário. Nota-se também que o algoritmo *Adam* foi o algoritmo mais rápido e com uma acurácia média equivalente aos valores obtidos pelos algoritmos GWO e BBO, com um valor de aproximadamente 0.81, porém o algoritmo BBO foi um dos algoritmos mais lentos, levando aproximadamente 35 segundos, e o algoritmo GWO obteve um tempo de execução intermediário, próximo de 20 segundos. Além disso, os algoritmos MVO e DE também obtiveram uma acurácia próxima do algoritmo *Adam*, com um valor de 0.8 e um tempo de aproximadamente 20 segundos.

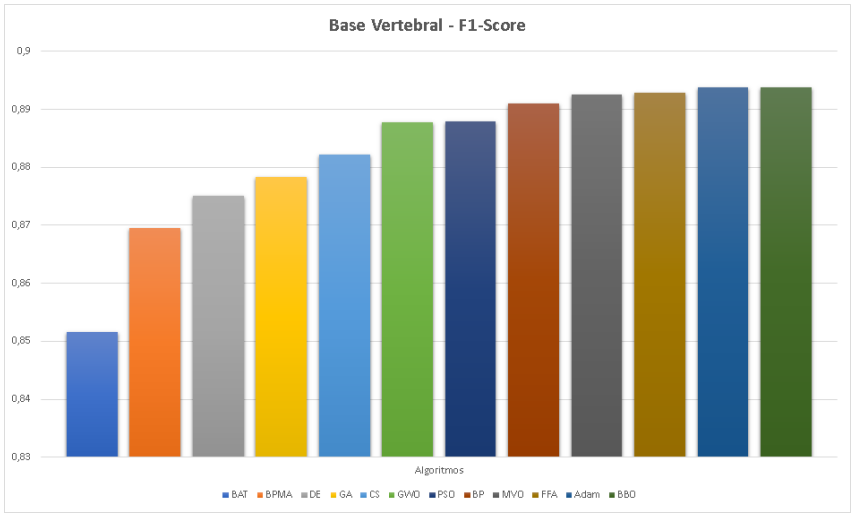
Observando o gráfico da Figura 26b, que representa a acurácia média e o tempo de execução médio de cada algoritmo utilizando 50 indivíduos na população ou 7500 iterações, percebe-se que o BAT continua sendo o pior algoritmo, porém, com uma acurácia média mais próxima da acurácia média obtida pelos algoritmos BP e BPMA, que continuaram pior em relação ao resto dos algoritmos NI em relação à acurácia. Além disso, o algoritmo *Adam* obteve uma acurácia média de 0.82, com uma pequena diferença de 0.002 em relação aos algoritmos GWO e BBO, porém, com um tempo de execução maior, com uma diferença de aproximadamente 60 segundos em relação ao GWO e sendo mais lento do que o restante dos algoritmos com exceção do CS. É importante destacar que o tempo de execução do *Adam* com 7500 iterações teve um grande aumento em relação ao *Adam* com 1500 iterações, mas a sua acurácia aumentou aproximadamente 0.01, o que indica que um aumento no número de iterações acaba não aprimorando tanto o resultado final e causando apenas um aumento de tempo de execução. O CS também foi outro algoritmo que teve um pequeno aumento na acurácia e passou a ser o algoritmo mais lento com uma população de tamanho 50.

Foram considerados outros gráficos de dispersão, um para cada tamanho de população, no entanto, o seu eixo *x* é o valor *F1-Score* médio, ao invés da acurácia média, enquanto o tempo

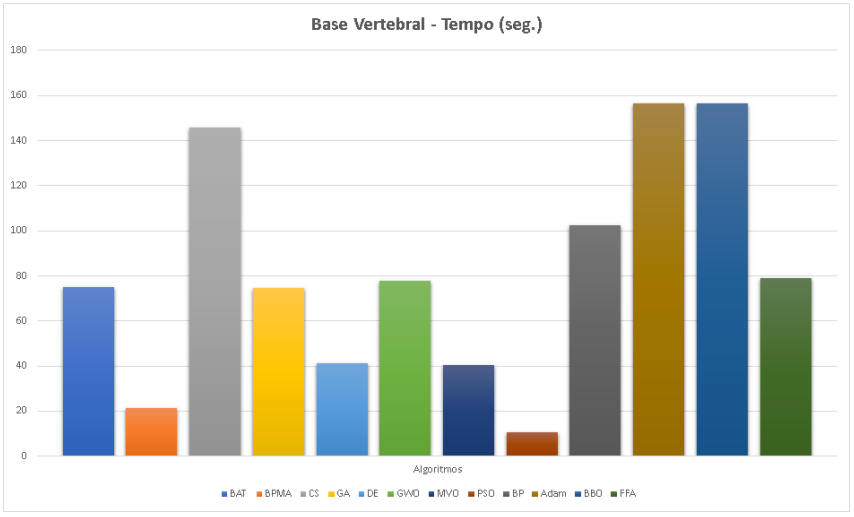
Figura 25 – Gráficos de acurácia, tempo de execução e F1-Score obtidos na base *vertebral* - 24a Acurácia obtida na base *vertebral* pelos 12 algoritmos de otimização. 23b - F1-Score obtido na base *vertebral* pelos 12 algoritmos de otimização. 25c - Tempo de execução obtido na base *vertebral* pelos 12 algoritmos de otimização.



(a)



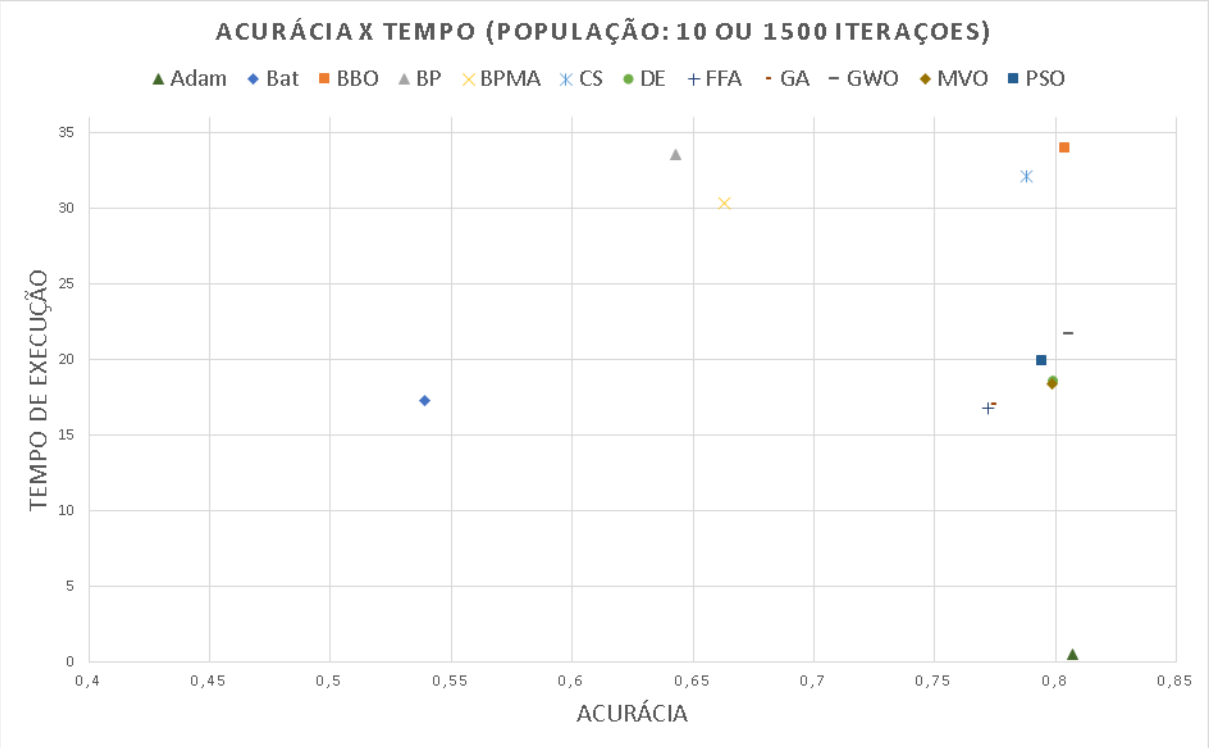
(b)



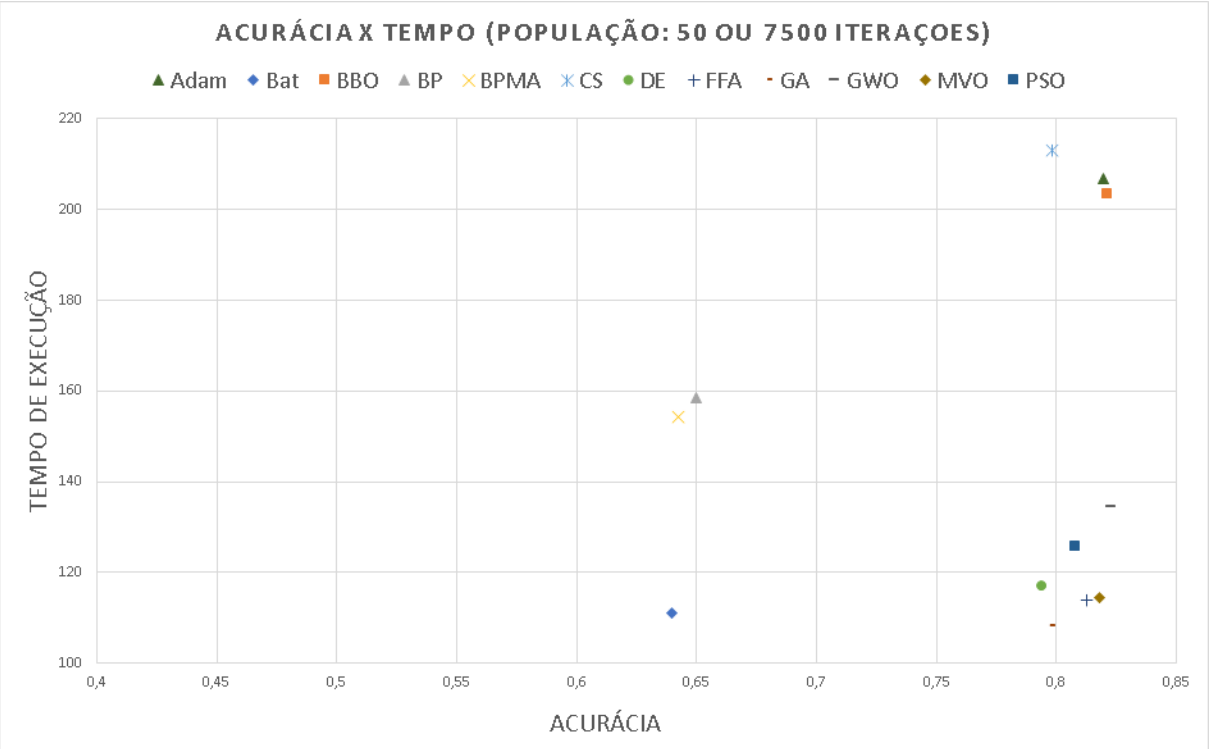
(c)

Fonte: Autor.

Figura 26 – Gráficos de acurácia média x tempo de execução médio das variações de cada um dos algoritmos. 26a - Acurácia x Tempo médio de cada algoritmo com população de tamanho 10 ou 1500 iterações. 26b - Acurácia x Tempo de cada modelo que possui diferentes funções de custo com população de tamanho 50 ou 7500 iterações.



(a)



(b)

Fonte: Autor.

de execução médio permanece representando o eixo *y*, apresentados na Figura 27. Para o gráfico da Figura 27a, que apresenta o *F1-Score* médio de cada algoritmo que utilizou uma população de tamanho 10 ou 1500 iterações, foi notado que o BAT obteve um valor próximo do BP com um tempo de execução menor, porém obteve um *F1-Score* menor do que o algoritmo BPMA. O algoritmo *Adam* obteve um *F1-Score* menor em relação ao PSO, DE, GWO e BBO. No entanto, em relação ao tempo de execução, o *Adam* foi mais rápido, com um tempo de aproximadamente 1 segundo.

A Figura 27b apresenta o gráfico com o *F1-Score* médio de cada algoritmo com a população de tamanho 50 ou 7500 iterações. Nesse gráfico, o BAT alcançou um valor de *F1-Score* médio maior em relação ao BP e BPMA. Além disso, o BBO e o GWO obtiveram um valor de *F1-Score* de aproximadamente 0.82, com uma diferença de aproximadamente 0.01 melhor em relação ao *Adam*, apesar do GWO ter obtido um tempo de execução aproximadamente duas vezes menor. Os algoritmos MVO, FFA, PSO e CS também obtiveram um valor de *F1-Score* próximo de 0.8, mostrando uma boa capacidade média para classificar amostras da classe positiva corretamente.

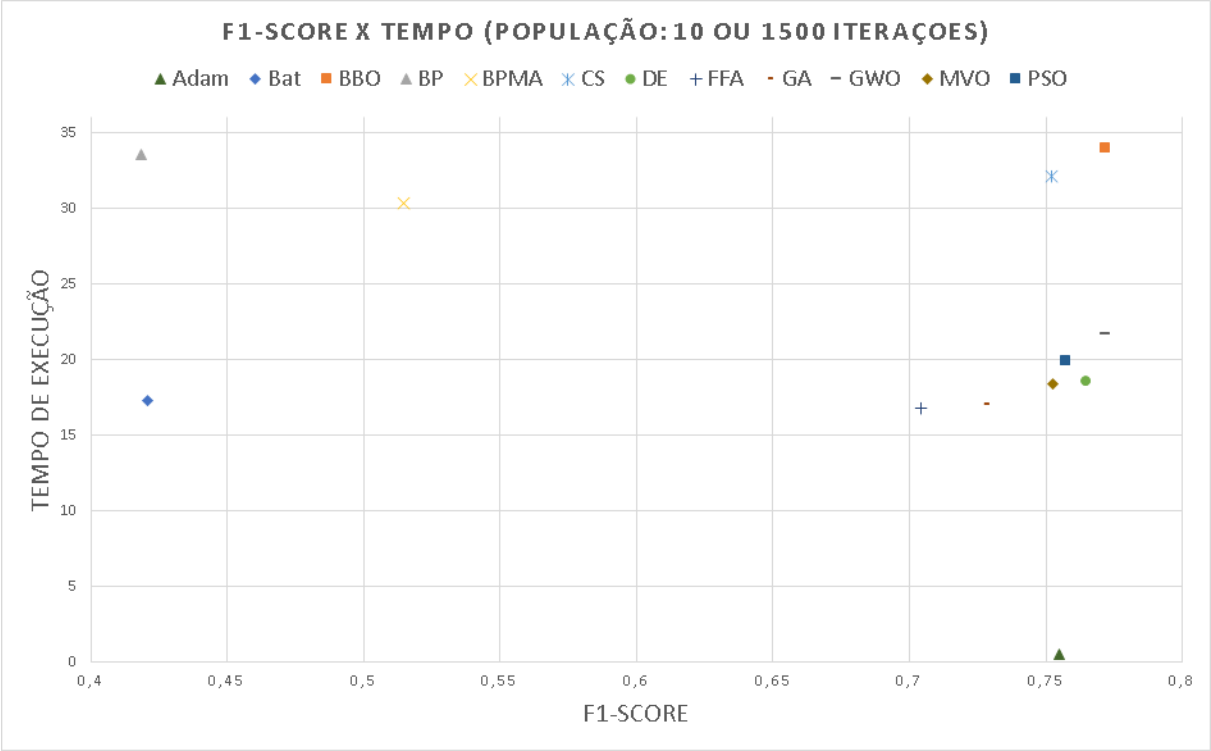
A partir desse resultados, foram observados alguns pontos importantes. Em primeiro lugar, o BAT apresentou dificuldades para alcançar valores de acurácia média e *F1-Score* próximos do restante dos algoritmos NI, mostrando um aprimoramento com uma população maior, o que indica a necessidade de ter uma população maior para realizar o treinamento e convergir para uma solução ótima global. Por outro lado, de forma geral, o restante dos algoritmos NI superou os algoritmos tradicionais BP e BPMA, em termos de acurácia, *F1-score* e tempo de execução, mostrando-se serem bons substitutos para esses algoritmos em diferentes problemas. Em relação ao algoritmo *Adam*, os algoritmos BBO, GWO e MVO obtiveram uma taxa de acurácia e *F1-Score* equivalentes, sendo os modelos que utilizam população de tamanho 10 mais lentos em relação ao *Adam*, porém, com uma população de tamanho 50, o algoritmo GWO obteve um tempo de execução mais rápido ao tempo do *Adam*. Isso mostra que esses algoritmos NI se aproximam do estado da arte, ao menos em termos de acurácia obtida e *F1-Score*, sendo o *Adam* um dos algoritmos para otimização de redes MLP com melhores resultados utilizados atualmente na literatura.

5.3 ANÁLISE DAS DIFERENTES FUNÇÕES DE ATIVAÇÃO

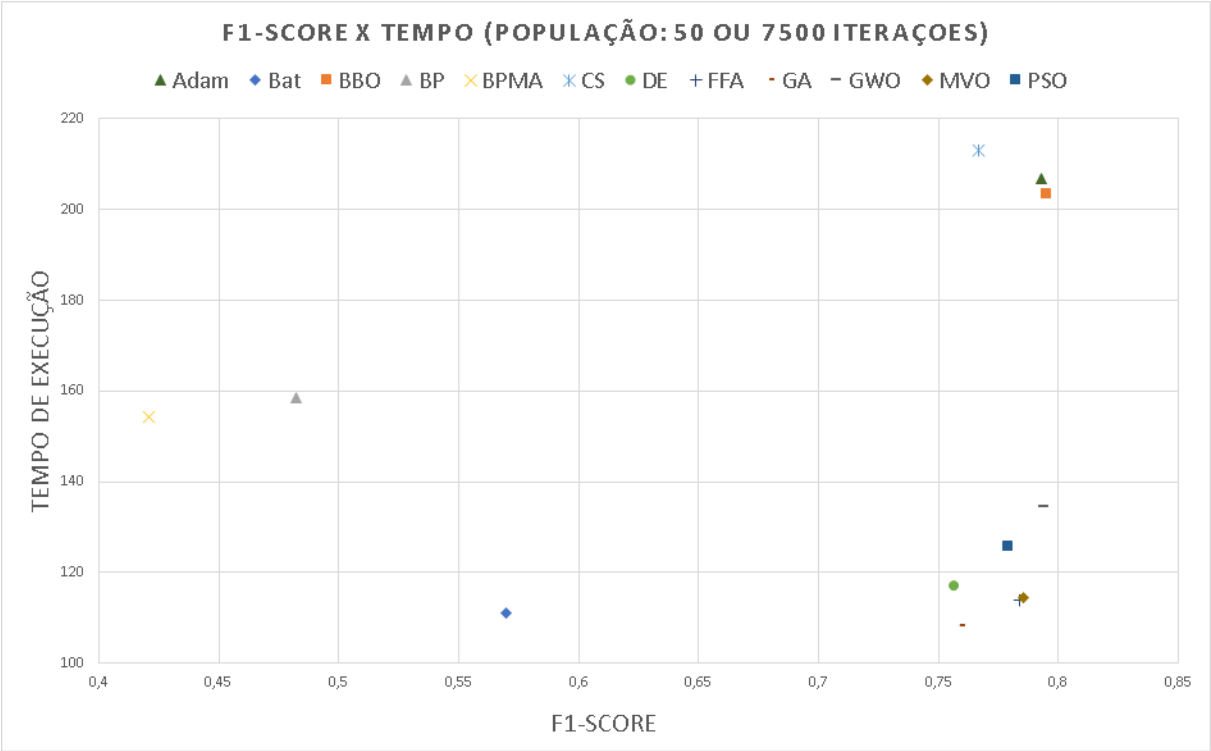
Foram adotadas a função Sigmoide, ReLU e Tanh como função de ativação para serem testadas em diversos modelos de redes neurais nesse trabalho.

Para comparar o desempenho entre as diferentes funções de ativação, foi calculada a média da acurácia e do tempo de execução para os modelos que utilizaram cada uma das funções de ativação, considerando todas as suas variações em todas as bases aplicadas. Com essas informações, foi construído um gráfico de dispersão, onde o eixo *x* representa a acurácia média, o

Figura 27 – Gráficos de *F1-Score* médio x tempo de execução médio das variações de cada um dos algoritmos. 27a - *F1-Score* x Tempo de cada modelo que possui diferentes funções de custo com população de tamanho 10 ou 1500 iterações. 27b - *F1-Score* x Tempo de cada modelo que possui diferentes funções de custo com população de tamanho 50 ou 7500 iterações.



(a)



(b)

Fonte: Autor.

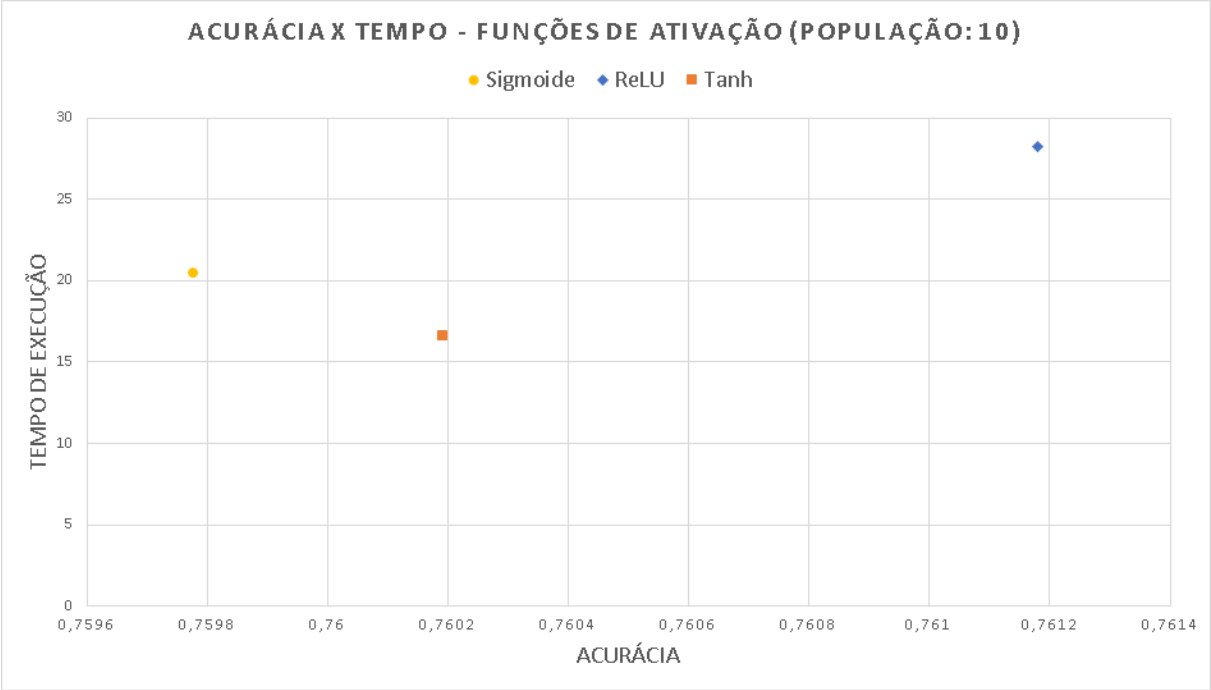
eixo *y* o tempo de execução médio e cada marcador representa uma função de ativação diferente, para cada um dos tamanhos de população diferentes, sendo apresentados na Figura 28.

Analisando as funções de ativação de forma geral, considerando a média dos modelos que utilizaram funções de ativação diferentes, foi observado que para os modelos que utilizaram uma população de tamanho 10 ou 1500 iterações, ilustrado na Figura 28a, a função ReLU obteve a maior taxa de acurácia média, com um valor de aproximadamente 0.761, enquanto as funções Tanh e Sigmoide obtiveram uma acurácia média de 0.76, com a função ReLU tendo apenas uma pequena diferença de acurácia média para um tempo de execução de aproximadamente 10 segundos adicionais. Para a média dos modelos com população de tamanho 50 ou 7500 iterações, que utilizaram funções de ativação diferentes, ilustrado na Figura 28b, foi observado que a função Sigmoide obteve a maior acurácia média, com um valor de aproximadamente 0.787 e um tempo de execução de 130 segundos, superando a acurácia média de aproximadamente 0.784 obtido pelos modelos que utilizaram a função Tanh e de aproximadamente 0.78 dos que utilizaram a função ReLU.

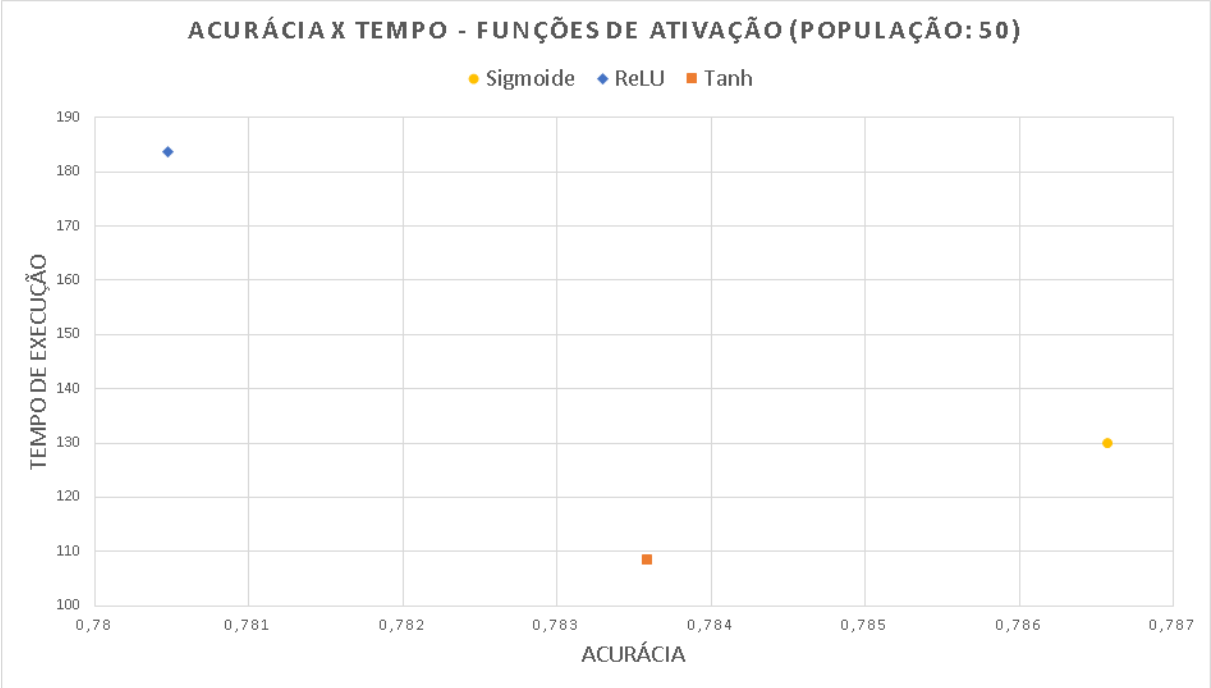
Em relação ao *F1-Score* médio, foi observado na Figura 29a que os modelos que utilizaram a função ReLU tiveram um valor médio de aproximadamente 0.71 para uma população de tamanho 10 ou 1500 iterações, superando por uma pequena diferença o *F1-Score* médio de aproximadamente 0.7 obtido pelos modelos que utilizaram a função Tanh e de aproximadamente 0.695 dos modelos que utilizaram a função Sigmoide. Para os modelos com uma população de tamanho 50 ou 7500 iterações, apresentados na Figura 29b, o maior *F1-Score* médio também foi obtido pelos modelos que utilizaram a função ReLU, com um valor de aproximadamente 0.744, estando próximo do *F1-Score* médio obtido pelos modelos com a função Sigmoide, sendo 0.743, enquanto os modelos que utilizaram a função Tanh tiveram um *F1-Score* médio de aproximadamente 0.732.

Com esses dados apresentados, pode-se concluir os seguintes pontos a respeito das diferentes funções de ativação em relação aos algoritmos de treinamento de RNA: Para ambos os tamanhos de população, a função ReLU mostrou-se ser, em média, a mais custosa, levando mais tempo do que as outras funções para realizar o treinamento, sendo a função Sigmoide a segunda mais custosa e a função Tanh a menos custosa. Isso se deve ao fato da função ReLU ter que fazer uma operação de comparação, sendo mais custosa computacionalmente. Em termos de acurácia, para os modelos com uma população pequena, de tamanho 10, as três funções de ativação obtiveram resultados bem próximos, com uma diferença de aproximadamente 0.01 entre elas, porém com a função ReLU sendo a mais lenta e a função Tanh a mais rápida. Além disso, com um tamanho de população maior, de tamanho 50, a função Sigmoide se destacou entre as demais, obtendo a maior acurácia, mostrando que com um número de iterações maior, os modelos treinados utilizando a função Sigmoide, em média, podem obter uma acurácia maior. Para o valor médio de *F1-Score*, a função ReLU obteve o melhor valor para os modelos com população de tamanho 10, enquanto que com os modelos com população de tamanho 50, o *F1-Score* médio

Figura 28 – Gráficos de acurácia média x tempo de execução médio dos modelos que utilizam diferentes funções de ativação. 28a - Acurácia x Tempo de cada modelo que possui diferentes funções de ativação com população de tamanho 10 ou 1500 iterações. 28b - Acurácia x Tempo de cada modelo que possui diferentes funções de ativação com população de tamanho 50 ou 7500 iterações.



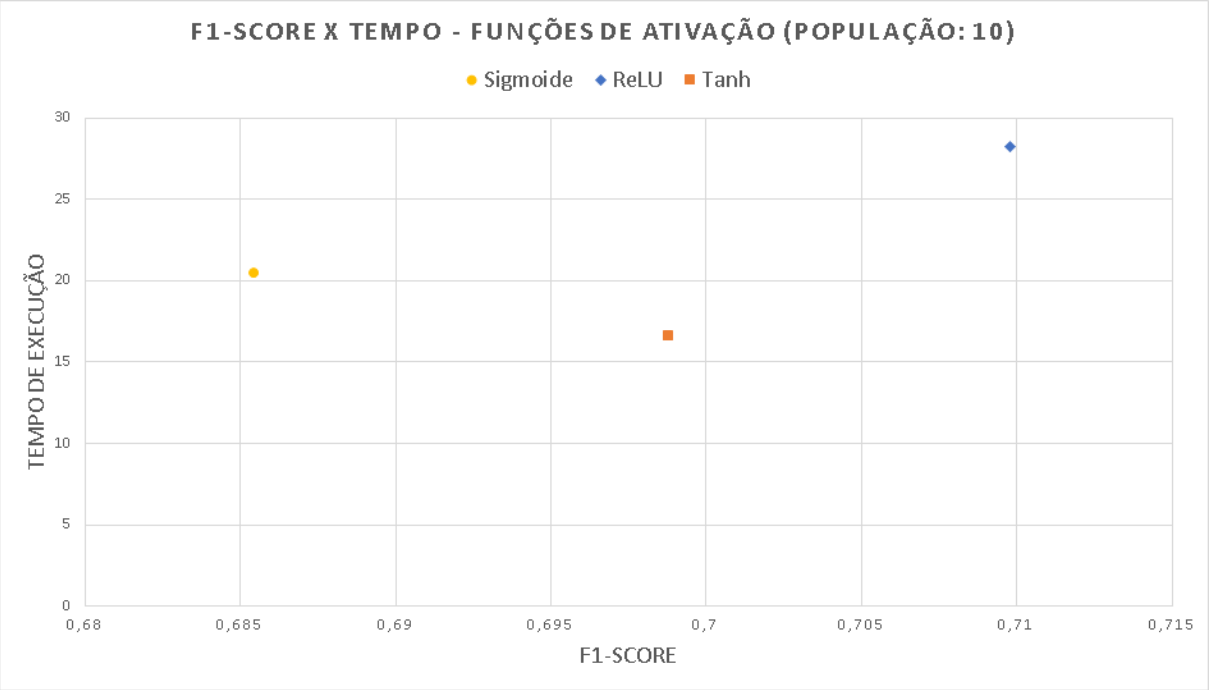
(a)



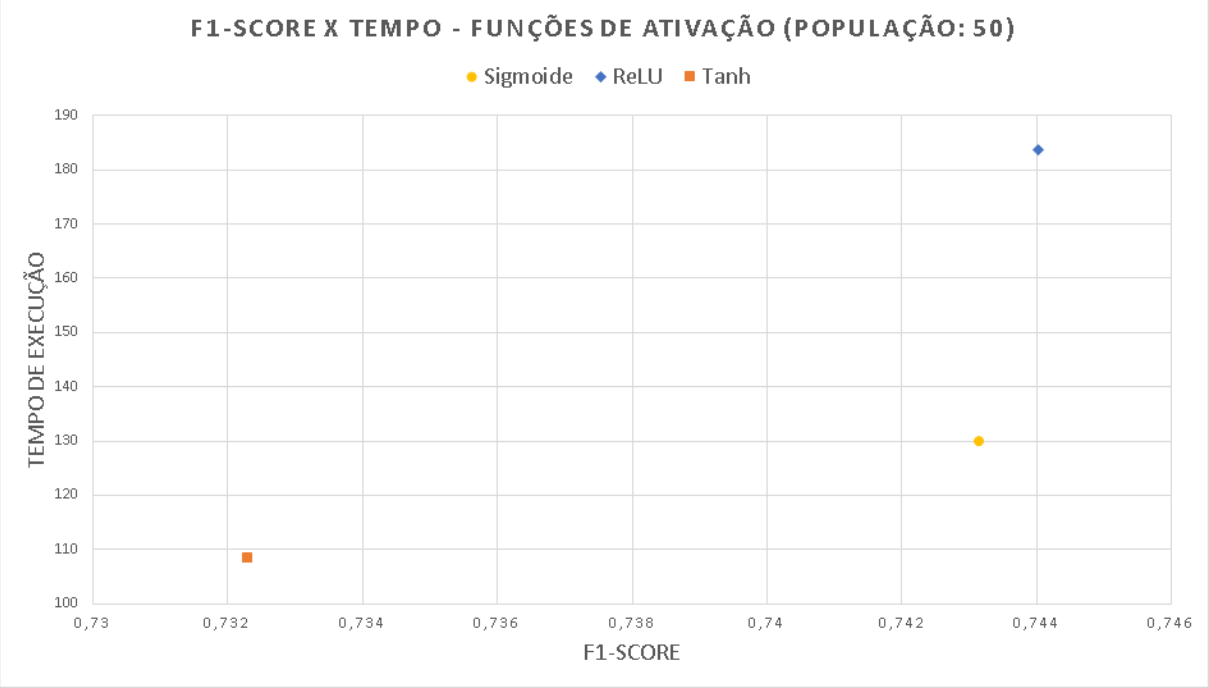
(b)

Fonte: Autor.

Figura 29 – Gráficos de *F1-Score* médio x tempo de execução médio dos modelos que utilizam diferentes funções de ativação. 29a - *F1-Score* x Tempo de cada modelo que possui diferentes funções de ativação com população de tamanho 10 ou 1500 iterações. 29b - *F1-Score* x Tempo de cada modelo que possui diferentes funções de ativação com população de tamanho 50 ou 7500 iterações.



(a)



(b)

Fonte: Autor.

obtido pela função ReLU teve uma diferença de apenas 0.001 da função Sigmoide, porém com um tempo de execução maior.

5.4 COMPARAÇÃO ENTRE AS DIFERENTES FUNÇÕES DE CUSTO

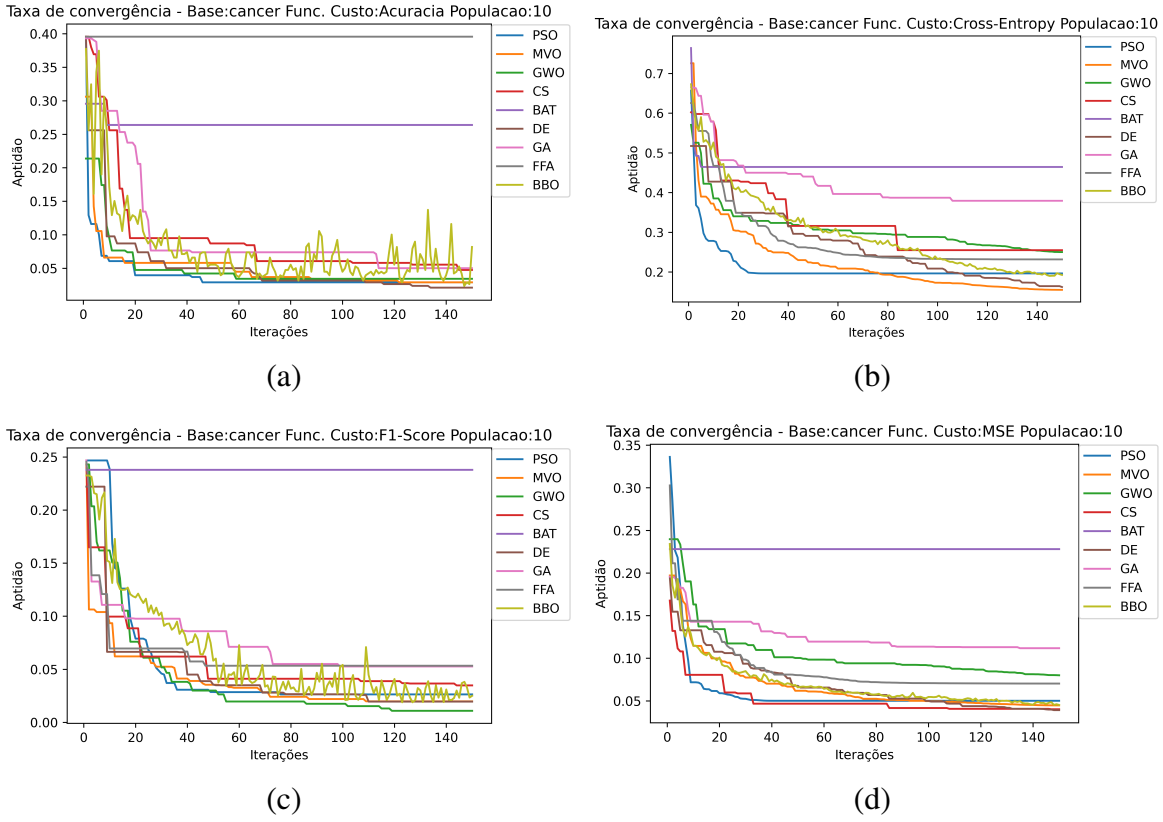
Nesse trabalho, foram consideradas as métricas MSE, acurácia, *F1-Score* e entropia cruzada como função de custo a ser otimizada pelos algoritmos de otimização, sendo as funções acurácia e *F1-Score* funções não deriváveis adotadas apenas pelos algoritmos NI. Para comparar o desempenho entre elas, será levado em conta primeiramente o impacto das diferentes funções na convergência dos algoritmos.

Para comparar a convergência dos algoritmos NI utilizando diferentes funções de custo, foram gerados gráficos que apresentam a aptidão obtida no treinamento ao longo das iterações. Os algoritmos tradicionais não foram considerados por não terem a mesma quantidade de iterações dos algoritmos NI. Pelo fato de terem sido gerados muitos gráficos com informações semelhantes, foram selecionados apenas os gráficos que apresentam as taxas de convergência obtidas nas bases *cancer* e *liver*, sendo as bases em que foram obtidas a maior e a menor taxa de acurácia média (0.958 e 0.697, respectivamente). Por ilustrar diferenças perceptíveis entre as funções de custo diferentes, foi considerada apenas a função de ativação sigmoide. Além disso, foram considerados os dois valores de tamanho da população, sendo 10 e 50. Todos os gráficos de convergência podem ser encontrados em `https://github.com/MFBouzon/Resultados_NNNI`.

Os gráficos das Figuras 30a e 30c mostram que as funções acurácia e *F1-Score* permitem com que os algoritmos com uma população de treinamento de tamanho 10 cheguem em pontos mínimos com menos iterações em relação às métricas entropia cruzada e MSE, encontrando outros pontos mínimos apenas em algumas iterações específicas. Por outro lado, a entropia cruzada e MSE permitem com que os algoritmos tenham uma convergência de forma com que sejam encontrados pontos mínimos a cada iteração e formando uma curva de convergência, como mostra as Figura 30d e 30b. Em relação às funções não deriváveis, é notado que o algoritmo BBO teve dificuldades em convergir, oscilando entre pontos mais altos em relação à iterações anteriores. Além disso, é observado que o BAT estagnou em mínimos locais para todas as métricas nesse caso.

Em relação a taxa de convergência obtidas na base *cancer*, utilizando uma população de tamanho 50, foi observado que os algoritmos que utilizaram a acurácia como função de custo, mostrado na Figura 31a, encontrando pontos mínimos em poucas iterações, sendo aprimorado também a convergência do BAT. Para a função de custo *F1-Score*, apresentada na Figura 31c, os algoritmos tiveram uma dificuldade maior de convergir, principalmente o BAT, que estagnou em um mínimo local, e o BBO que oscilou entre pontos maiores e menores. Utilizando a função entropia cruzada, observada na Figura 31b, o BAT e o GA tiveram dificuldades em convergir, enquanto o FFA convergiu em aproximadamente 60 iterações. O restante dos algoritmos tiveram

Figura 30 – Gráficos da taxa de convergência dos algoritmos NI na base *cancer* com uma população de tamanho 10. 30a - Taxas de convergência obtidas na base *cancer* utilizando a métrica acurácia como função de custo. 30b - Taxas de convergência obtidas na base *cancer* utilizando a métrica entropia cruzada como função de custo. 30c - Taxas de convergência obtidas na base *cancer* utilizando a métrica *F1-Score* como função de custo. 30d - Taxas de convergência obtidas na base *cancer* utilizando a métrica MSE como função de custo.



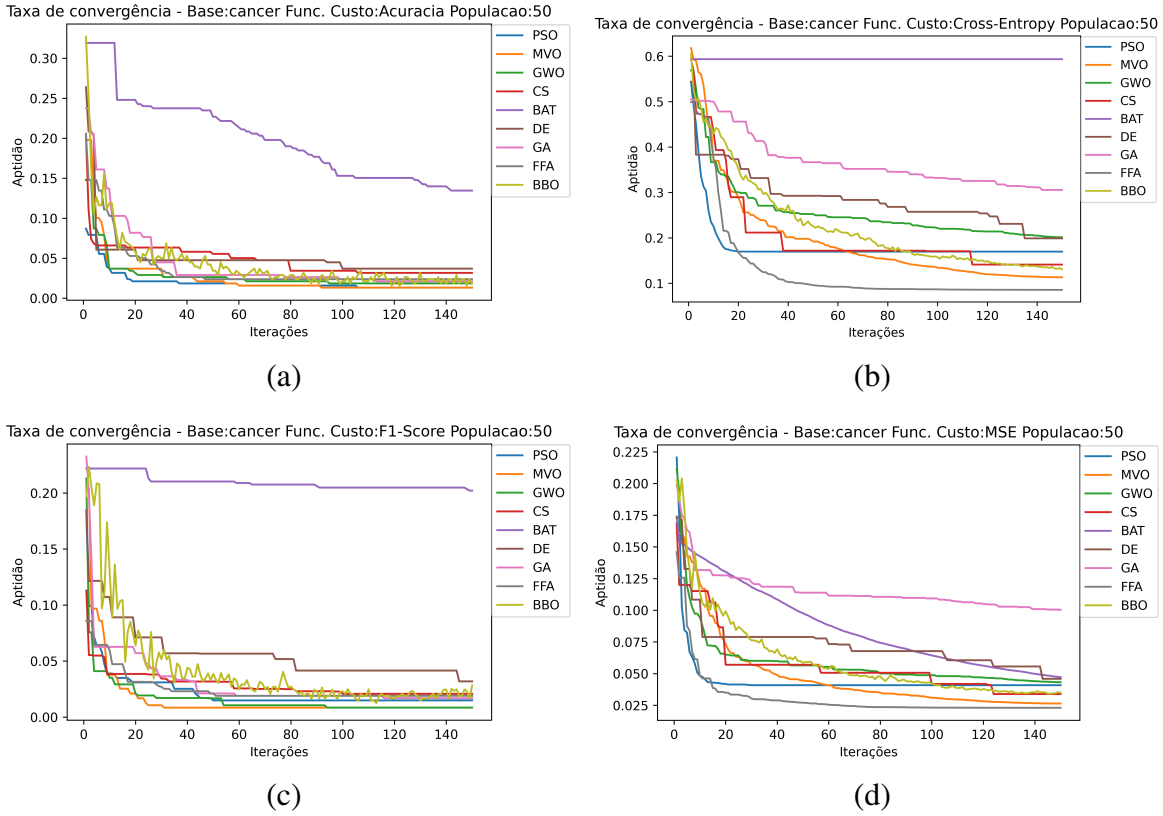
Fonte: Autor.

um valor de aptidão final pior comparado ao obtido pelo FFA. A função MSE mostrou ser a mais adequada para o BAT, formando uma curva de convergência que diminui o erro a cada iteração. Além disso, também mostrou uma convergência em poucas iterações para o FFA e o MVO.

Ao observar os gráficos de convergência da base *liver*, utilizando uma população de tamanho 10, na Figura 32, nota-se que as funções não deriváveis mostraram uma dificuldade maior nessa base, que é considerada mais desafiadora, ilustrado nas Figuras 32a e 32c, sendo notado que a maioria dos algoritmos encontrou poucos pontos mínimos no espaço de busca ao longo das iterações. Nessa base, as funções entropia cruzada e MSE também mostraram dificuldades em convergir, ilustrado nas Figuras 32b e 32d, podendo ser visto que houve uma pequena variação da aptidão dos algoritmos ao longo das iterações. Isso se deu pelo fato dessa base ser considerada difícil de ser classificada.

Para as taxas de convergência da base *liver*, utilizando uma população de tamanho 50, pode-se observar que para a acurácia como função de custo, mostrada na Figura 33a, o GWO e o

Figura 31 – Gráficos da taxa de convergência dos algoritmos NI na base *cancer* com uma população de tamanho 50. 31a - Taxas de convergência obtidas na base *cancer* utilizando a métrica acurácia como função de custo. 31b - Taxas de convergência obtidas na base *cancer* utilizando a métrica entropia cruzada como função de custo. 31c - Taxas de convergência obtidas na base *cancer* utilizando a métrica *F1-Score* como função de custo. 31d - Taxas de convergência obtidas na base *cancer* utilizando a métrica MSE como função de custo.

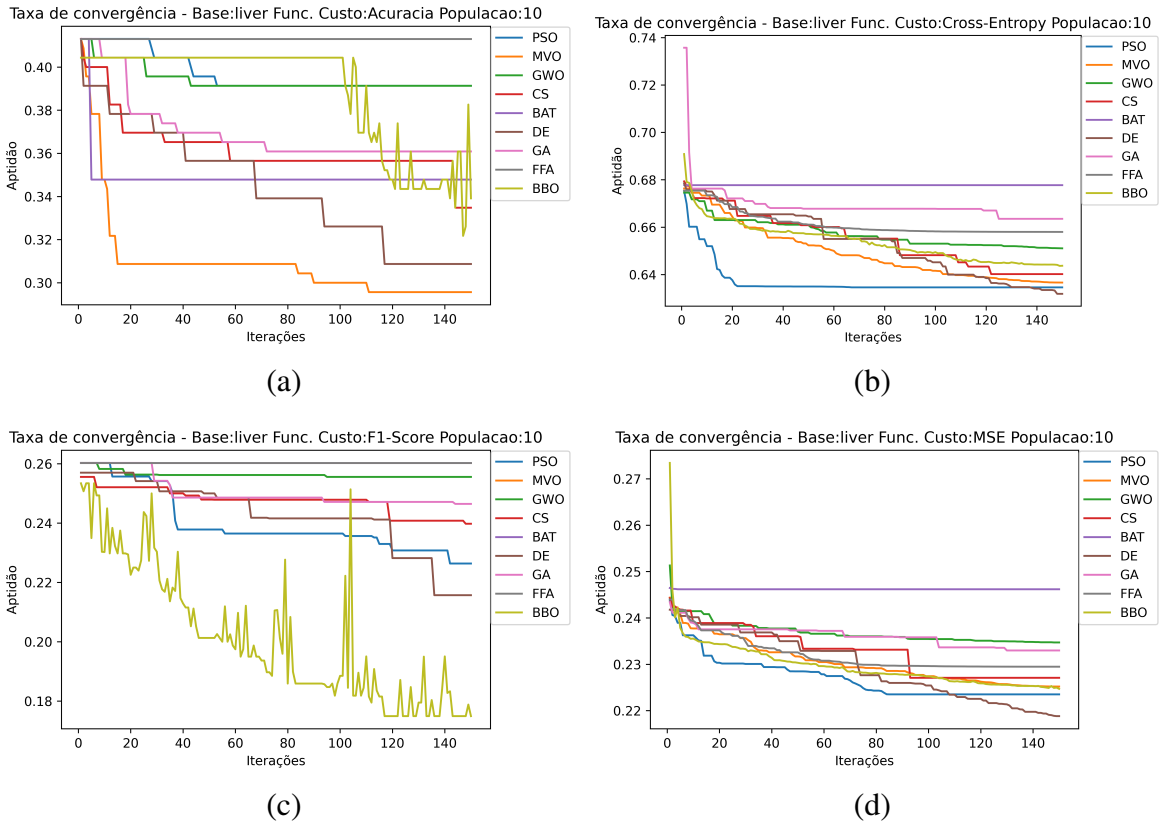


Fonte: Autor.

BBO diminuíram o seu valor de aptidão ao longo das iterações, enquanto o resto dos algoritmos se mantiveram estagnados em ótimos locais. Para a função *F1-Score*, mostrada na Figura 33c, os algoritmos se mantiveram estagnados, não alcançaram valores menores de aptidão ao longo das iterações. Para as funções entropia cruzada e MSE, observadas nas Figuras 33b e 33d, os algoritmos diminuíram o valor de aptidão ao longo das iterações, o que indica que com mais iterações, os resultados obtidos para a base *liver* poderiam ser aprimorados.

Para comparar o desempenho entre as diferentes métricas adotadas como função de custo, foi calculada a média da acurácia e do tempo de execução para cada um delas, considerando todas as variações dos modelos que as utilizaram em todas as bases aplicadas. Com essas informações, foi construído um gráfico de dispersão, onde o eixo x representa a acurácia média, o eixo y o tempo de execução médio e cada marcador representa uma função de custo diferente, para cada um dos tamanhos de população diferentes.

Figura 32 – Gráficos da taxa de convergência dos algoritmos NI na base *liver* com uma população de tamanho 10. 32a - Taxas de convergência obtidas na base *liver* utilizando a métrica acurácia como função de custo. 32b - Taxas de convergência obtidas na base *liver* utilizando a métrica entropia cruzada como função de custo. 32c - Taxas de convergência obtidas na base *liver* utilizando a métrica *F1-Score* como função de custo. 32d - Taxas de convergência obtidas na base *liver* utilizando a métrica MSE como função de custo.

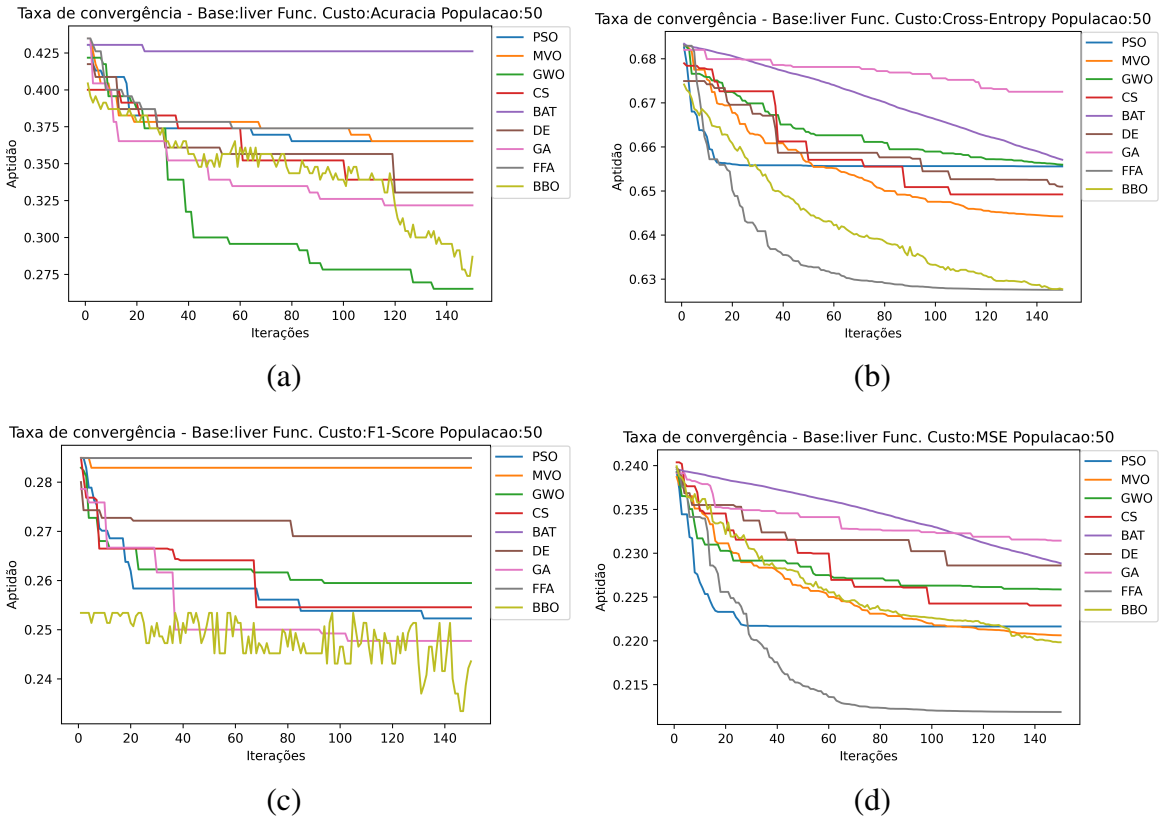


Fonte: Autor.

Avaliando as métricas de forma geral, calculando a média da acurácia e tempo de execução de todos os modelos que usaram cada uma das métricas, a partir da Figura 34a, foi observado que para uma população de tamanho 10, os modelos que utilizaram a acurácia como função objetivo obtiveram uma acurácia média de aproximadamente 0.775, tendo uma diferença de aproximadamente 0.01 em relação aos modelos que utilizaram as métricas MSE e entropia cruzada, enquanto os modelos que utilizaram o *F1-Score* como função de custo obtiveram uma acurácia média de aproximadamente 0.75. No caso dos modelos com uma população de tamanho 50, ilustrados na Figura 34b, a métrica MSE como função objetivo superou a métrica acurácia por uma diferença de 0.02, com um valor de aproximadamente 0.798 de acurácia média, enquanto os modelos que utilizaram as métricas entropia cruzada e *F1-Score* obtiveram uma acurácia média de aproximadamente 0.786 e 0.782 respectivamente.

Os gráficos da Figura 29 mostram que os modelos que utilizaram como função objetivo a maximização do *F1-Score* tiveram os valores de *F1-Score* médio mais baixos dentre todas as

Figura 33 – Gráficos da taxa de convergência dos algoritmos NI na base *liver* com uma população de tamanho 50. 33a - Taxas de convergência obtidas na base *liver* utilizando a métrica acurácia como função de custo. 33b - Taxas de convergência obtidas na base *liver* utilizando a métrica entropia cruzada como função de custo. 33c - Taxas de convergência obtidas na base *liver* utilizando a métrica *F1-Score* como função de custo. 33d - Taxas de convergência obtidas na base *liver* utilizando a métrica MSE como função de custo.



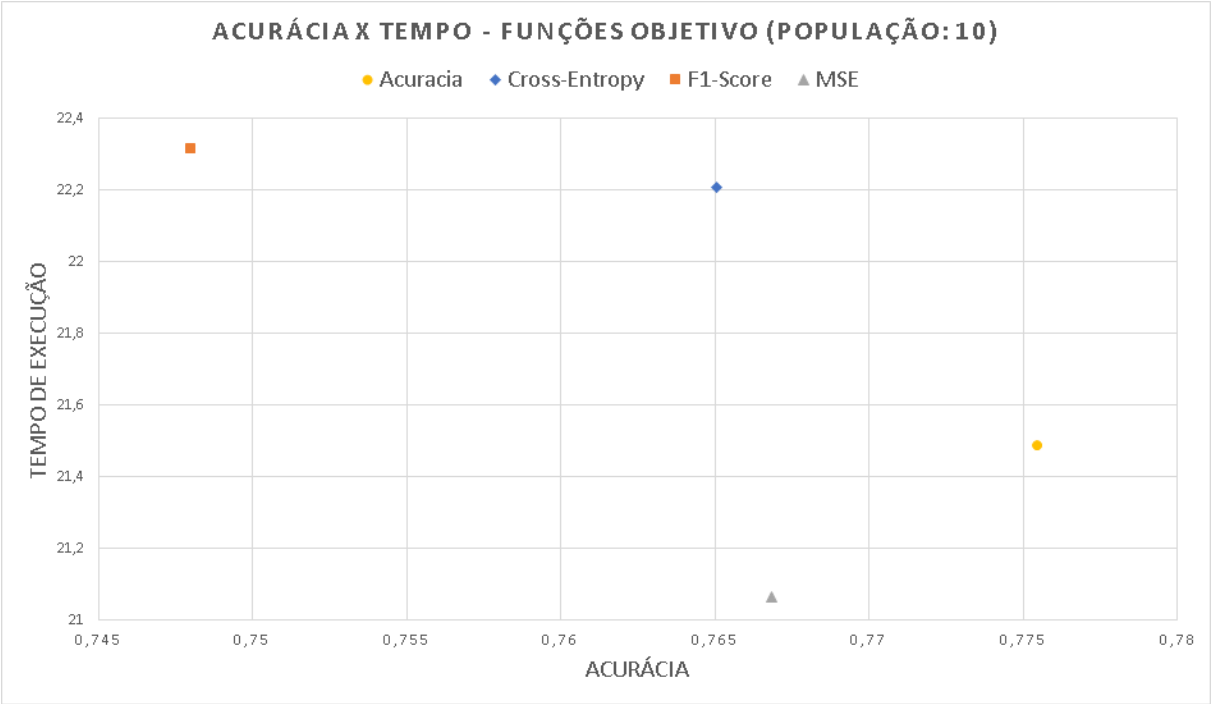
Fonte: Autor.

métricas utilizadas como função objetivo. Para os modelos com um tamanho 10 da população, apresentados na Figura 35a, a função de custo com o maior valor de *F1-Score* foi a acurácia, seguido da métrica entropia cruzada e do MSE, com um valor de *F1-Score* médio de aproximadamente 0.73. No caso dos modelos com uma população de tamanho 50, a métrica MSE obteve um valor médio de *F1-Score* de aproximadamente 0.77, sendo o maior valor entre todas as métricas, com uma diferença próxima de 0.1 dos valores da acurácia e da entropia cruzada.

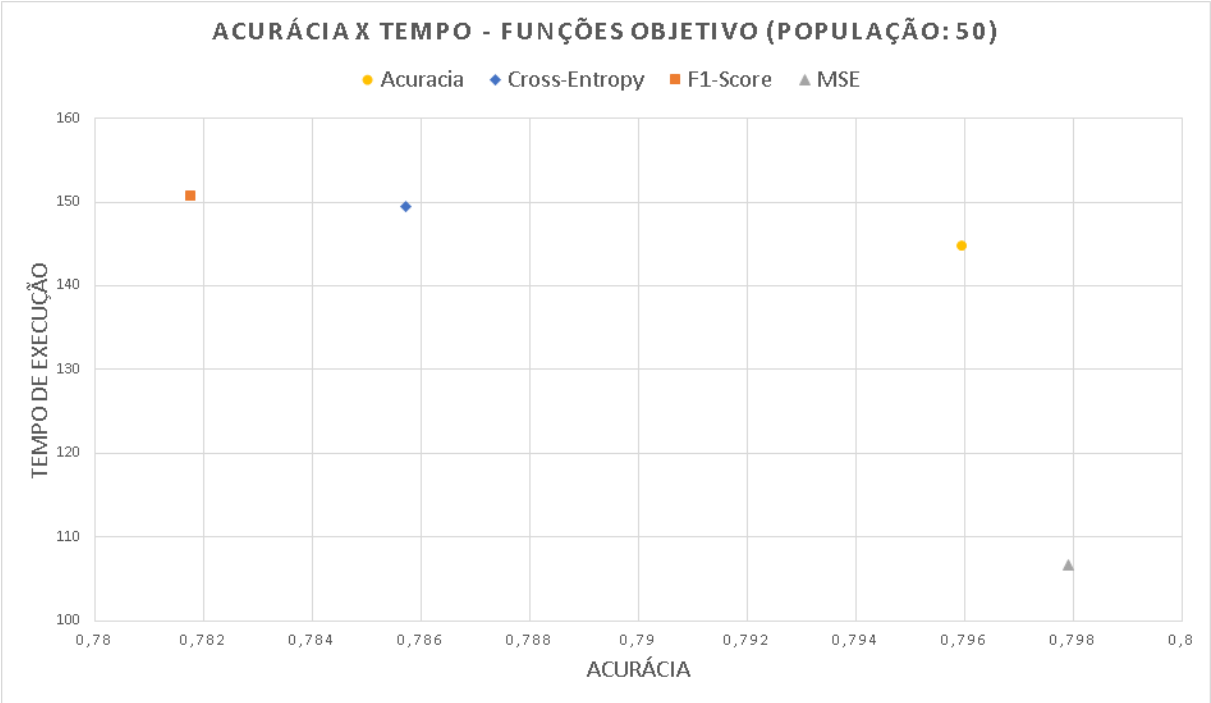
Em relação ao tempo de execução, os resultados indicam que a métrica MSE, em média, é a função de custo mais rápida, mostrando uma diferença de tempo maior quando o tamanho da população é maior. Além disso, o MSE também obteve valores de acurácia e *F1-Score* maiores nos modelos em que foram adotadas uma população maior.

Sendo assim, pode-se sumarizar os seguintes pontos encontrados ao analisar as diferentes funções de custo:

Figura 34 – Gráficos de acurácia média x tempo de execução médio dos modelos que utilizam diferentes funções de custo. 34a - Acurácia x Tempo de cada modelo que possui diferentes funções de custo com população de tamanho 10 ou 1500 iterações. 34b - Acurácia x Tempo de cada modelo que possui diferentes funções de custo com população de tamanho 50 ou 7500 iterações.



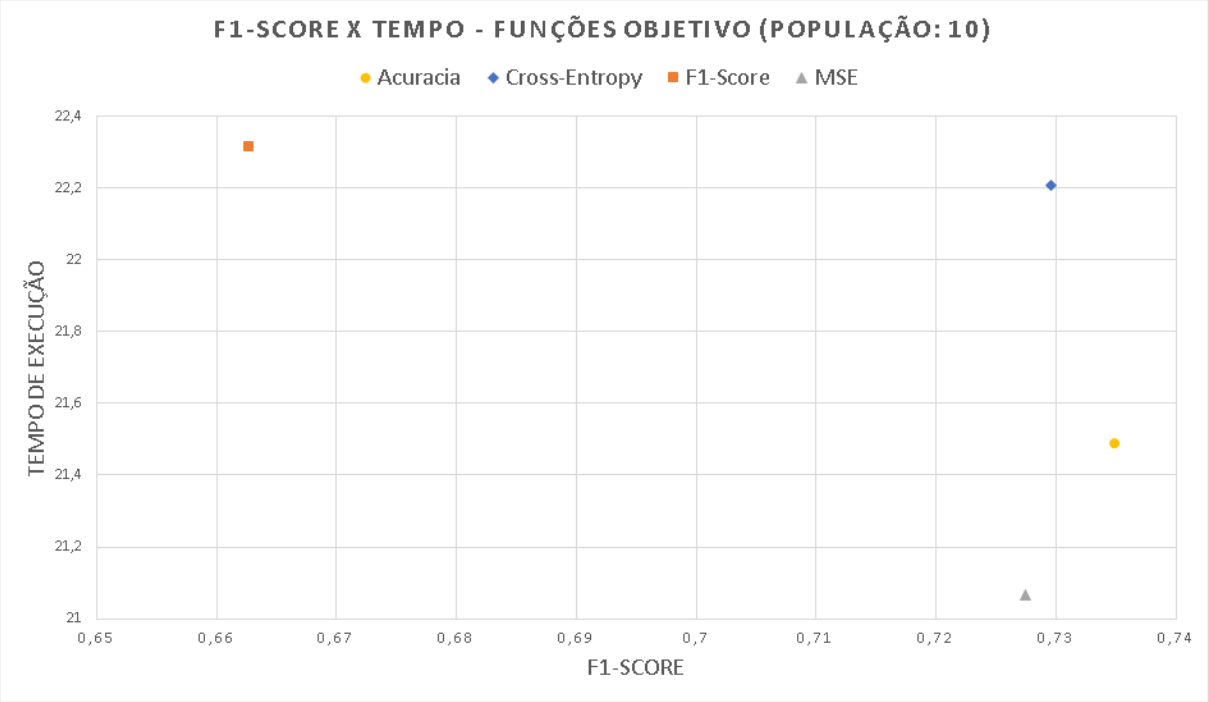
(a)



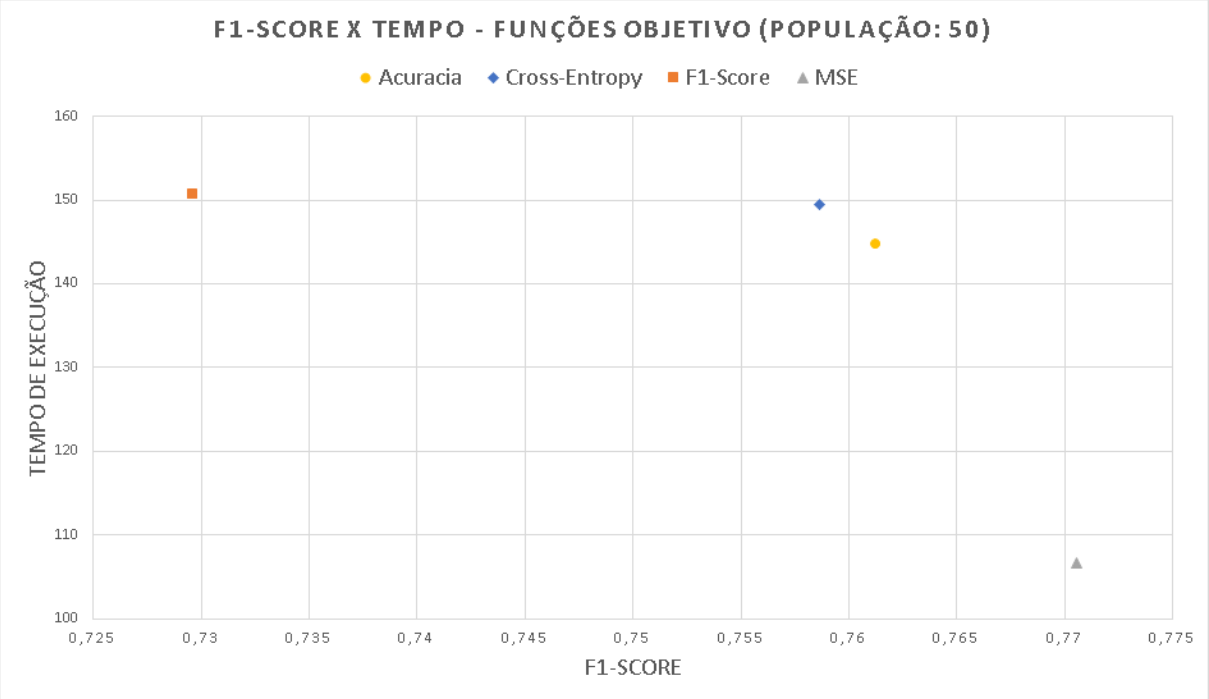
(b)

Fonte: Autor.

Figura 35 – Gráficos de *F1-Score* médio x tempo de execução médio dos modelos que utilizam diferentes funções de custo. 35a - *F1-Score* x Tempo de cada modelo que possui diferentes funções de custo com população de tamanho 10 ou 1500 iterações. 35b - *F1-Score* x Tempo de cada modelo que possui diferentes funções de custo com população de tamanho 50 ou 7500 iterações.



(a)



(b)

Fonte: Autor.

a) As funções não deriváveis apresentaram uma taxa de convergência que chega ao ponto mínimo com menos iterações na base *cancer*. No entanto, para uma base mais desafiadora, como a base *liver*, essas funções apresentaram dificuldades em busca de mínimos globais. Por outro lado, as funções MSE e entropia cruzada mostraram uma convergência melhor quando o tamanho da população é maior.
b) A métrica *F1-Score* foi a pior função em termos de acurácia média, *F1-Score* médio e tempo de execução.
c) A métrica MSE mostrou ser a mais rápida e aquela com o maior valor de acurácia média e *F1-Score* médio em modelos com uma população maior.
d) Foi observado também que para o treinamento de alguns algoritmos, como o BAT, é necessário uma população de tamanho maior, obtendo uma taxa de convergência ótima utilizando o MSE

5.5 CONTRIBUIÇÕES

Após a análise dos resultados, pode-se destacar as principais contribuições desse trabalho para a área de treinamento de RNAs, sendo elas:

a) A comparação de diversos algoritmos NI, incluindo desde algoritmos clássicos, como o PSO e o GA, até algoritmos mais recentes, como o GWO e o MVO. Além disso, foi feita a comparação desses algoritmos com o algoritmo *Adam*, popular pela qualidade dos resultados obtidos em treinamento de RNAs. Os resultados mostraram que alguns dos algoritmos NI obtiveram valores semelhantes de acurácia e *F1-Score*, com um tempo de execução mais rápido em alguns casos.
b) O impacto de diferentes funções de ativação para cada algoritmo, sendo observado que a função Sigmoide obteve um equilíbrio entre tempo de execução e acurácia média.
c) A comparação de diferentes funções de custo, algo que foi pouco explorado na pesquisa bibliográfica feita, sendo um diferencial em relação aos outros trabalhos já publicados. De forma geral, foi observado que os algoritmos NI possuem a vantagem de utilizarem funções não deriváveis para resolverem problemas de otimização, no entanto, em alguns casos, essas funções apresentaram dificuldades para o algoritmo convergir, resultando em um tempo de execução alto com valores de acurácia e *F1-Score* menores em relação às outras funções.
d) A variação do tamanho da população do algoritmo, mostrando como alguns algoritmos necessitam de uma população maior para ter resultados melhores, como por exemplo o BAT, enquanto outros algoritmos, como o GWO e o MVO, obtiveram resultados melhores com uma população menor.

6 CONCLUSÃO

Neste trabalho foi proposta uma metodologia para avaliar o uso de algoritmos de otimização NI para treinar redes neurais de classificação.

Para isso, foram selecionados 9 algoritmos NI, utilizando como critérios a frequência de uso e desempenho em outros trabalhos, a recência, a tradição, a categoria e o número de citações. Também foram selecionados 3 algoritmos tradicionais para comparação, sendo eles o BP, o BPMA e o algoritmo *Adam*, conhecido pela qualidade dos seus resultados obtidos a partir do treinamento de redes neurais. Esses 12 algoritmos foram utilizados para classificar 9 bases de dados, utilizando 3 funções de ativação, 4 funções de custo e variando o tamanho da população para 10 e 50, ou 1500 e 7500 iterações para os algoritmos tradicionais.

Para avaliar os algoritmos, foram extraídas a acurácia, *F1-Score* e tempo de execução em segundos de cada um deles. Os resultados mostraram que os algoritmos BBO, GWO e MVO foram os algoritmos NI que mais se destacaram em acurácia média e *F1-Score* médio, com valores próximos de 0.82 e 0.8, respectivamente. Além disso, eles obtiveram resultados próximos ao algoritmo *Adam*, tendo uma diferença próxima de 0.02, com o algoritmo GWO obtendo um tempo de execução 60 segundos mais rápido em relação ao *Adam*, utilizando uma população de tamanho 50.

Em relação às funções de ativação, a função ReLU foi a que obteve o maior tempo de execução, obtendo uma diferença de acurácia média de 0.01 em relação às funções Sigmoide e Tanh. Com uma população de tamanho 50, a função Sigmoide se destacou, obtendo uma acurácia média maior e um valor de *F1-Score* equivalente à função ReLU, porém, obteve um tempo de execução menor.

Para as diferentes funções de custo, as funções não deriváveis (acurácia e *F1-Score*) atingiram pontos mínimos em menos iterações para bases menos complexas, no entanto, para bases de dificuldade de classificação maior, elas apresentaram dificuldades em sua busca por pontos mínimos globais. No caso das funções MSE e *Cross-Entropy*, com um tamanho de população maior, a taxa de convergência tende a melhorar, minimizando a busca a cada iteração. Dessa forma, os resultados indicaram que a função MSE, em média, alcançaram os maiores valores de acurácia e *F1-Score* em menor tempo, para os modelos que utilizaram uma população de tamanho 50.

Por fim, pode-se dizer que o uso de algoritmos de otimização NI é uma boa opção para o treinamento de RNAs, trazendo resultados equivalentes a métodos do estado da arte, com um tempo de execução menor em alguns casos. No entanto, ainda existem muitos fatores que podem ser explorados por trabalhos futuros. Alguns exemplos são: A exploração de paralelismo e otimização dos algoritmos NI para aprimorar o tempo de execução; abordar o problema de aprendizado profundo com algoritmos NI.

REFERÊNCIAS

ABIODUN, Oludare Isaac et al. State-of-the-art in artificial neural network applications: A survey. **Heliyon**, v. 4, 2018.

AL NUAIMI, Zakaria Noor Aldeen Mahmood; ABDULLAH, Rosni. Neural network training using hybrid particlemove artificial bee colony algorithm for pattern classification. **Journal of Information and Communication Technology**, v. 16, n. 2, p. 314–334, 2017.

ALJARAH, Ibrahim; FARIS, Hossam; MIRJALILI, Seyedali. Optimizing connection weights in neural networks using the whale optimization algorithm. **Soft Computing**, Springer, v. 22, n. 1, p. 1–15, 2018.

ALJARAH, Ibrahim et al. Training radial basis function networks using biogeography-based optimizer. **Neural Computing and Applications**, Springer, v. 29, n. 7, p. 529–553, 2018.

AN, Yang et al. Event classification for natural gas pipeline safety monitoring based on long short-term memory network and Adam algorithm. **Structural Health Monitoring**, v. 19, p. 1151–1159, 2020.

ASHRAF, Adnan et al. Training of Artificial Neural Network Using New Initialization Approach of Particle Swarm Optimization for Data Classification. In: IEEE. 2020 International Conference on Emerging Trends in Smart Technologies (ICETST). [S.l.: s.n.], 2020. P. 1–6.

ATASHPAZ-GARGARI, Esmaeil; LUCAS, Caro. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: IEEE. 2007 IEEE congress on evolutionary computation. [S.l.: s.n.], 2007. P. 4661–4667.

BAHADORI, M. et al. A supplier selection model for hospitals using a combination of artificial neural network and fuzzy VIKOR. **International Journal of Healthcare Management**, v. 13, p. 286–294, 2020.

BEHESHTI, Zahra et al. Enhancement of artificial neural network learning using centripetal accelerated particle swarm optimization for medical diseases diagnosis. **Soft Computing**, Springer, v. 18, n. 11, p. 2253–2270, 2014.

BHANDARE, Ashray Sadashiv. **Bio-inspired Algorithms for Evolving the Architecture of Convolutional Neural**. Dez. 2017. Diss. (Mestrado) – University of Toledo.

BISHOP, Christopher M. **Pattern recognition and machine learning**. [S.l.]: springer, 2006.

BOSIRE, Adrian. Recurrent Neural Network Training using ABC Algorithm for Traffic Volume Prediction. **Informatica**, v. 43, n. 4, 2019.

BRAJEVIC, Ivona; TUBA, Milan. Training feed-forward neural networks using firefly algorithm. **Recent advances in knowledge engineering and systems science**, 2013.

BUI, Quang-Thanh. Metaheuristic algorithms in optimizing neural network: a comparative study for forest fire susceptibility mapping in Dak Nong, Vietnam. **Geomatics, Natural Hazards and Risk**, Taylor & Francis, v. 10, n. 1, p. 136–150, 2019.

CAO, Zijian et al. An improved brain storm optimization with differential evolution strategy for applications of ANNs. **Mathematical Problems in Engineering**, Hindawi, v. 2015, 2015.

CHEN, Huadong et al. A hybrid of artificial fish swarm algorithm and particle swarm optimization for feedforward neural network training. In: ATLANTIS PRESS. INTERNATIONAL Conference on Intelligent Systems and Knowledge Engineering 2007. [S.l.: s.n.], 2007.

CHHACHHIYA, Devika; SHARMA, Amita; GUPTA, Manish. Designing optimal architecture of neural network with particle swarm optimization techniques specifically for educational dataset. In: IEEE. 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence. [S.l.: s.n.], 2017. P. 52–57.

DETRANO, Robert et al. International application of a new probability algorithm for the diagnosis of coronary artery disease. **The American journal of cardiology**, Elsevier, v. 64, n. 5, p. 304–310, 1989.

DEVIKANNIGA, D; RAJ, R Joshua Samuel. Classification of osteoporosis by artificial neural network based on monarch butterfly optimisation algorithm. **Healthcare technology letters**, IET, v. 5, n. 2, p. 70–75, 2018.

DEVIKANNIGA, D; VETRIVEL, K; BADRINATH, N. Review of Meta-Heuristic Optimization based Artificial Neural Networks and its Applications. In: IOP PUBLISHING, 1. JOURNAL of Physics: Conference Series. [S.l.: s.n.], 2019. v. 1362, p. 012074.

DIXIT, Ujjawal et al. Texture classification using convolutional neural network optimized with whale optimization algorithm. **SN Applied Sciences**, Springer, v. 1, n. 6, p. 655, 2019.

DORADO-MORENO, Manuel et al. Dynamically weighted evolutionary ordinal neural network for solving an imbalanced liver transplantation problem. **Artificial Intelligence in Medicine**, Elsevier, v. 77, p. 1–11, 2017.

DORIGO, Marco; BIRATTARI, Mauro; STUTZLE, Thomas. Ant colony optimization. **IEEE computational intelligence magazine**, IEEE, v. 1, n. 4, p. 28–39, 2006.

DUCH, Wlodzislaw; JANKOWSKI, Norbert. Survey of neural transfer functions. **Neural Computing Surveys**, v. 2, n. 1, p. 163–212, 1999.

DUCHI, John; HAZAN, Elad; SINGER, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of machine learning research**, v. 12, n. 7, 2011.

ENGELBRECHT, Andries P; ENGELBRECHT, Ap; ISMAIL, A. Training product unit neural networks. Citeseer, 1999.

FARIS, H. et al. EvoloPy: An Open-source Nature-inspired Optimization Framework in Python. In: IJCCI. [S.l.: s.n.], 2016.

FARIS, Hossam; ALJARAH, Ibrahim; MIRJALILI, Seyedali. Improved monarch butterfly optimization for unconstrained global search and neural network training. **Applied Intelligence**, Springer, v. 48, n. 2, p. 445–464, 2018.

_____. Training feedforward neural networks using multi-verse optimizer for binary classification problems. **Applied Intelligence**, Springer, v. 45, n. 2, p. 322–332, 2016.

FARIS, Hossam et al. EvoloPy: An Open-source Nature-inspired Optimization Framework in Python. In: IJCCI (ECTA). [S.l.: s.n.], 2016. P. 171–177.

FARIS, Hossam et al. Optimizing the learning process of feedforward neural networks using lightning search algorithm. **International Journal on Artificial Intelligence Tools**, World Scientific, v. 25, n. 06, p. 1650033, 2016.

FELDMAN, Sarah. **Machine Learning Tops AI Dollars**. [S.l.: s.n.], mai. 2019. https://www.statista.com/chart/17966/worldwide-artificial-intelligence-funding/.

GARRO, Beatriz A; VÁZQUEZ, Roberto A. Designing artificial neural networks using particle swarm optimization algorithms. **Computational intelligence and neuroscience**, Hindawi, v. 2015, 2015.

GOLDBERG, D. Genetic Algorithms in Search Optimization and Machine Learning. In:

HAJIHASSANI, Mohsen et al. Ground vibration prediction in quarry blasting through an artificial neural network optimized by imperialist competitive algorithm. **Bulletin of Engineering Geology and the Environment**, Springer, v. 74, n. 3, p. 873–886, 2015.

HARIFI, Sasan et al. Optimizing a Neuro-Fuzzy System based on nature inspired Emperor Penguins Colony optimization algorithm. **IEEE Transactions on Fuzzy Systems**, IEEE, 2020.

HEGAZY, T_; FAZIO, P; MOSELHI, O. Developing practical neural network applications using back-propagation. **Computer-Aided Civil and Infrastructure Engineering**, Wiley Online Library, v. 9, n. 2, p. 145–159, 1994.

HEMEIDA, AM et al. Implementation of nature-inspired optimization algorithms in some data mining tasks. **Ain Shams Engineering Journal**, Elsevier, v. 11, n. 2, p. 309–318, 2020.

HEMEIDA, Ashraf Mohamed et al. Nature-inspired algorithms for feed-forward neural network classifiers: A survey of one decade of research. **Ain Shams Engineering Journal**, Elsevier, 2020.

HINTON, Geoffrey; SRIVASTAVA, Nitsh; SWERSKY, Kevin. Neural networks for machine learning. **Coursera, video lectures**, v. 264, n. 1, 2012.

HOLLAND, J. Adaptation in natural and artificial systems. In:

HU, Hongping et al. The Improved Antlion Optimizer and Artificial Neural Network for Chinese Influenza Prediction. **Complexity**, Hindawi, v. 2019, 2019.

HUNTER, J. D. Matplotlib: A 2D Graphics Environment. **Computing in Science Engineering**, v. 9, 2007.

ISLAM, MM Faniqul et al. Likelihood prediction of diabetes at early stage using data mining techniques. In: COMPUTER Vision and Machine Intelligence in Medical Image Analysis. [S.l.]: Springer, 2020. P. 113–125.

J.NET, Swathi; J.N, Swathi. A SURVEY ON NATURE INSPIRED METAHEURISTIC TECHNIQUES FOR TRAINING FEEDFORWAD NEURAL NETWORKS. In:

JADDI, Najmeh Sadat; ABDULLAH, Salwani; HAMDAN, Abdul Razak. Multi-population cooperative bat algorithm-based optimization of artificial neural network model. **Information Sciences**, Elsevier, v. 294, p. 628–644, 2015.

JONES, E.; OLIPHANT, T. E.; PETERSON, P. SciPy: Open Source Scientific Tools for Python. In:

KARABOGA, Dervis; BASTURK, Bahriye. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. **Journal of global optimization**, Springer, v. 39, n. 3, p. 459–471, 2007.

KENNEDY, James; EBERHART, Russell. Particle swarm optimization. In: IEEE. PROCEEDINGS of ICNN'95-International Conference on Neural Networks. [S.l.: s.n.], 1995. v. 4, p. 1942–1948.

KHARE, Neelu et al. SMO-DNN: Spider Monkey Optimization and Deep Neural Network Hybrid Classifier Model for Intrusion Detection. **Electronics**, Multidisciplinary Digital Publishing Institute, v. 9, n. 4, p. 692, 2020.

KIM, GH; SEO, DS; KANG, Kyung In. Hybrid models of neural networks and genetic algorithms for predicting preliminary cost estimates. **Journal of Computing in Civil Engineering**, American Society of Civil Engineers, v. 19, n. 2, p. 208–211, 2005.

KINGMA, Diederik P; BA, Jimmy. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KOHAVI, Ron et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: MONTREAL, CANADA, 2. IJCAI. [S.l.: s.n.], 1995. v. 14, p. 1137–1145.

KOWALSKI, Piotr A; ŁUKASIK, Szymon. Training neural networks with krill herd algorithm. **Neural Processing Letters**, Springer, v. 44, n. 1, p. 5–17, 2016.

KRKOVÁ, Věra. Kolmogorov's theorem is relevant. **Neural computation**, MIT Press, v. 3, n. 4, p. 617–622, 1991.

KUMARAN, J; SASIKALA, J. Dragonfly optimization based ANN model for forecasting India's primary fuels' demand. **International Journal of Computer Applications**, Foundation of Computer Science, v. 164, n. 7, p. 18–22, 2017.

LAKSHMI, N. DHANA. NATURE INSPIRED OPTIMIZATION ALGORITHMS IN ARTIFICIAL NEURAL NETWORK FOR SPEAKER RECOGNITION. **International Journal of Electrical Engineering and Technology (IJEET)**, v. 9, n. 3, p. 114–120, 2018.

LECUN, Yann A et al. Efficient backprop. In: NEURAL networks: Tricks of the trade. [S.l.]: Springer, 2012. P. 9–48.

LEE, Anzy; GEEM, Zong Woo; SUH, Kyung-Duck. Determination of optimal initial weights of an artificial neural network by using the harmony search algorithm: application to breakwater armor stones. **Applied Sciences**, Multidisciplinary Digital Publishing Institute, v. 6, n. 6, p. 164, 2016.

LEEMA, N; NEHEMIAH, H Khanna; KANNAN, Arputharaj. Neural network classifier optimization using differential evolution with global information and back propagation algorithm for clinical datasets. **Applied Soft Computing**, Elsevier, v. 49, p. 834–844, 2016.

LIN, W. et al. An Artificial Neural Network Approach to Power Consumption Model Construction for Servers in Cloud Data Centers. **IEEE Transactions on Sustainable Computing**, v. 5, p. 329–340, 2020.

LITTLE, Max A et al. Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. **Biomedical engineering online**, Springer, v. 6, n. 1, p. 23, 2007.

MACHMUD, Rizanda; WIJAYA, Adi et al. Behavior determinant based cervical cancer early detection with machine learning algorithm. **Advanced Science Letters**, American Scientific Publishers, v. 22, n. 10, p. 3120–3123, 2016.

MAHAJAN, Aakanksha; KUMAR, Sushil; BANSAL, Rohit. Diagnosis of diabetes mellitus using PCA and genetically optimized neural network. In: IEEE. 2017 International Conference on Computing, Communication and Automation (ICCCA). [S.l.: s.n.], 2017. P. 334–338.

MALMGREN, Helge; BORGA, Magnus; NIKLASSON, Lars. **Artificial Neural Networks in Medicine and Biology: Proceedings of the ANNIMAB-1 Conference, Göteborg, Sweden, 13–16 May 2000**. [S.l.]: Springer Science & Business Media, 2012.

MANDAL, Sudip; SAHA, Goutam; PAL, Rajat K. Neural network training using firefly algorithm. **Global Journal on Advancement in Engineering and Science (GJAES)**, v. 1, n. 1, 2015.

MAVROVOUNIOTIS, Michalis; YANG, Shengxiang. Training neural networks with ant colony optimization algorithms for pattern classification. **Soft Computing**, Springer, v. 19, n. 6, p. 1511–1522, 2015.

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

MCDERMOTT, James; FORSYTH, Richard S. Diagnosing a disorder in a classification benchmark. **Pattern Recognition Letters**, Elsevier, v. 73, p. 41–43, 2016.

MEHTA, S.; PAUNWALA, Chirag; VAIDYA, Bhaumik. CNN based Traffic Sign Classification using Adam Optimizer. **2019 International Conference on Intelligent Computing and Control Systems (ICCS)**, p. 1293–1298, 2019.

MENEZES, Roberto. **Thermal Generator Scheduling using Adaptive Genetic Algorithm and Interior Point Method**. Jan. 2017. Tese (Doutorado).

MILLER, Geoffrey F; TODD, Peter M; HEGDE, Shailesh U. Designing Neural Networks using Genetic Algorithms. In: ICGA. [S.l.: s.n.], 1989. v. 89, p. 379–384.

MINSKY, Marvin; PAPERT, Seymour. An introduction to computational geometry. **Cambridge tiass., HIT**, 1969.

MIRJALILI, Seyedali. Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. **Neural Computing and Applications**, Springer, v. 27, n. 4, p. 1053–1073, 2016.

_____. How effective is the Grey Wolf optimizer in training multi-layer perceptrons. **Applied Intelligence**, Springer, v. 43, n. 1, p. 150–161, 2015.

MIRJALILI, Seyedali; LEWIS, Andrew. The whale optimization algorithm. **Advances in engineering software**, Elsevier, v. 95, p. 51–67, 2016.

MIRJALILI, Seyedali; MIRJALILI, Seyed Mohammad; HATAMLOU, Abdolreza. Multi-verse optimizer: a nature-inspired algorithm for global optimization. **Neural Computing and Applications**, Springer, v. 27, n. 2, p. 495–513, 2016.

MIRJALILI, Seyedali; MIRJALILI, Seyed Mohammad; LEWIS, Andrew. Grey wolf optimizer. **Advances in engineering software**, Elsevier, v. 69, p. 46–61, 2014.

_____. Let a biogeography-based optimizer train your multi-layer perceptron. **Information Sciences**, Elsevier, v. 269, p. 188–209, 2014.

MOAYEDI, Hossein et al. Novel nature-inspired hybrids of neural computing for estimating soil shear strength. **Applied Sciences**, Multidisciplinary Digital Publishing Institute, v. 9, n. 21, p. 4643, 2019.

MOLINA, Daniel et al. Comprehensive Taxonomies of Nature-and Bio-inspired Optimization: Inspiration versus Algorithmic Behavior, Critical Analysis and Recommendations. **arXiv preprint arXiv:2002.08136**, 2020.

MONTANA, David J; DAVIS, Lawrence. Training feedforward neural networks using genetic algorithms. In: IJCAI. [S.l.: s.n.], 1989. v. 89, p. 762–767.

NAGANNA, Sujay Raghavendra et al. Dew point temperature estimation: application of artificial intelligence model integrated with nature-inspired optimization algorithms. **Water**, Multidisciplinary Digital Publishing Institute, v. 11, n. 4, p. 742, 2019.

NAIR, Vinod; HINTON, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In: ICML. [S.l.: s.n.], 2010.

NAYAK, Janmenjoy; NAIK, Bighnaraj; BEHERA, HS. A novel nature inspired firefly algorithm with higher order neural network: performance analysis. **Engineering Science and Technology, an International Journal**, Elsevier, v. 19, n. 1, p. 197–211, 2016.

NAYAK, SC; MISRA, BB; BEHERA, HS. Artificial chemical reaction optimization of neural networks for efficient prediction of stock market indices. **Ain Shams Engineering Journal**, Elsevier, v. 8, n. 3, p. 371–390, 2017.

OJHA, Varun Kumar; ABRAHAM, Ajith; SNÁŠEL, Václav. Metaheuristic design of feedforward neural networks: A review of two decades of research. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 60, p. 97–116, 2017.

PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PHAN, Trong Huy; YAMAMOTO, K. Resolving Class Imbalance in Object Detection with Weighted Cross Entropy Losses. **ArXiv**, abs/2006.01413, 2020.

QIAO, Weibiao; MOAYEDI, Hossein; FOONG, Loke Kok. Nature-inspired hybrid techniques of IWO, DA, ES, GA, and ICA, validated through a k-fold validation process predicting monthly natural gas consumption. **Energy and Buildings**, Elsevier, p. 110023, 2020.

QIN, Jiaohua et al. A biological image classification method based on improved CNN. **Ecol. Informatics**, v. 58, p. 101093, 2020.

RANI R, Hannah Jessie; VICTOIRE T, Aruldoss Albert. Training radial basis function networks for wind speed prediction using PSO enhanced differential search optimizer. **PloS one**, Public Library of Science San Francisco, CA USA, v. 13, n. 5, e0196871, 2018.

RAO, Thiriveedhi Yellamanda Srinivasa; REDDY, Pakanati Chenna. Content and context based image retrieval classification based on firefly-neural network. **Multimedia Tools and Applications**, Springer, v. 77, n. 24, p. 32041–32062, 2018.

RASHEDI, Esmat; NEZAMABADI-POUR, Hossein; SARYAZDI, Saeid. GSA: a gravitational search algorithm. **Information sciences**, Elsevier, v. 179, n. 13, p. 2232–2248, 2009.

ROCHA NETO, Ajalmar R da et al. Diagnostic of pathology on the vertebral column with embedded reject option. In: SPRINGER. IBERIAN Conference on Pattern Recognition and Image Analysis. [S.l.: s.n.], 2011. P. 588–595.

RODAN, Ali; FARIS, Hossam; ALQATAWNA, Ja'far et al. Optimizing feedforward neural networks using biogeography based optimization for e-mail spam identification. **International Journal of Communications, Network and System Sciences**, Scientific Research Publishing, v. 9, n. 01, p. 19, 2016.

ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958.

RUMELHART, David E; HINTON, Geoffrey E; WILLIAMS, Ronald J. Learning representations by back-propagating errors. **nature**, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

SADOWSKI, Łukasz et al. The nature-inspired metaheuristic method for predicting the creep strain of green concrete containing ground granulated blast furnace slag. **Materials**, Multidisciplinary Digital Publishing Institute, v. 12, n. 2, p. 293, 2019.

SAHA, Sankhadip; CHAKRABORTY, Dwaipayan; DUTTA, Oindrilla. Guided convergence for training feed-forward neural network using novel gravitational search optimization. In: IEEE. 2014 International Conference on High Performance Computing and Applications (ICHPCA). [S.l.: s.n.], 2014. P. 1–6.

SCHAFFER, J David; WHITLEY, Darrell; ESHELMAN, Larry J. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In: IEEE. [PROCEEDINGS] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks. [S.l.: s.n.], 1992. P. 1–37.

SERIZAWA, Tatsuki; FUJITA, Hamido. Optimization of Convolutional Neural Network Using the Linearly Decreasing Weight Particle Swarm Optimization. **arXiv preprint arXiv:2001.05670**, 2020.

SHETA, Alaa; BRAIK, Malik; AL-HIARY, Heba. Modeling the Tennessee Eastman chemical process reactor using bio-inspired feedforward neural network (BI-FF-NN). **The International Journal of Advanced Manufacturing Technology**, Springer, v. 103, n. 1-4, p. 1359–1380, 2019.

SHI, Yuhui; EBERHART, Russell C. Empirical study of particle swarm optimization. In: IEEE. PROCEEDINGS of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406). [S.l.: s.n.], 1999. v. 3, p. 1945–1950.

SHIH, Po-Chou et al. Application of Metaheuristic Algorithms for Determining the Structure of a Convolutional Neural Network with a Small Dataset. In: PROCEEDINGS of the 3rd International Conference on Computer Science and Application Engineering. [S.l.: s.n.], 2019. P. 1–7.

SIMON, Dan. Biogeography-based optimization. **IEEE transactions on evolutionary computation**, IEEE, v. 12, n. 6, p. 702–713, 2008.

SIVAGAMINATHAN, Rahul Karthik; RAMAKRISHNAN, Sreeram. A hybrid approach for feature subset selection using neural networks and ant colony optimization. **Expert systems with applications**, Elsevier, v. 33, n. 1, p. 49–60, 2007.

SMITH, Jack W et al. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In: AMERICAN MEDICAL INFORMATICS ASSOCIATION. PROCEEDINGS of the Annual Symposium on Computer Application in Medical Care. [S.l.: s.n.], 1988. P. 261.

STORN, Rainer; PRICE, Kenneth. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. **Journal of global optimization**, Springer, v. 11, n. 4, p. 341–359, 1997.

STREET, W Nick; WOLBERG, William H; MANGASARIAN, Olvi L. Nuclear feature extraction for breast tumor diagnosis. In: INTERNATIONAL SOCIETY FOR OPTICS e PHOTONICS. BIOMEDICAL image processing and biomedical visualization. [S.l.: s.n.], 1993. v. 1905, p. 861–870.

TAKEDA, H.; YOSHIDA, S.; MUNEYASU, M. Learning from Noisy Labeled Data Using Symmetric Cross-Entropy Loss for Image Classification. **2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)**, p. 709–711, 2020.

TIAN, Zhonghuan; FONG, Simon. Survey of meta-heuristic algorithms for deep learning training. **Optimization algorithms—methods and applications**, 2016.

TING, Kai Ming; SAMMUT, C; WEBB, GI. Encyclopedia of machine learning. **KM Ting.–Boston, MA: Springer**, 2011.

VAN DEN BERGH, F. Particle swarm weight initialization in multi-layer perceptron artificial neural networks. **Development and practice of artificial intelligence techniques**, Durban South Africa, v. 41, 1999.

WANG, Bin et al. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In: IEEE. 2018 IEEE Congress on Evolutionary Computation (CEC). [S.l.: s.n.], 2018. P. 1–8.

WANG, Shuihua et al. Fruit classification by wavelet-entropy and feedforward neural network trained by fitness-scaled chaotic ABC and biogeography-based optimization. **Entropy**, Multidisciplinary Digital Publishing Institute, v. 17, n. 8, p. 5711–5728, 2015.

WDAA, Abdul Sttar Ismail; STTAR, A. **Differential evolution for neural networks learning enhancement**. 2008. Tese (Doutorado) – Universiti Teknologi Malaysia.

WHITLEY, Darrell. Applying genetic algorithms to neural network learning. In: PROCEEDINGS of the Seventh Conference (AISB89) on Artificial Intelligence and Simulation of Behaviour. [S.l.: s.n.], 1989. P. 137–144.

YAMANY, Waleed et al. Moth-flame optimization for training multi-layer perceptrons. In: IEEE. 2015 11th International computer engineering Conference (ICENCO). [S.l.: s.n.], 2015. P. 267–272.

YANG, C. et al. Robot Learning System Based on Adaptive Neural Control and Dynamic Movement Primitives. **IEEE Transactions on Neural Networks and Learning Systems**, v. 30, p. 777–787, 2019.

YANG, Jie; MA, Jun. Feed-forward neural network training using sparse representation. **Expert Systems with Applications**, Elsevier, v. 116, p. 255–264, 2019.

YANG, Xin-She. A new metaheuristic bat-inspired algorithm. In: NATURE inspired cooperative strategies for optimization (NICSO 2010). [S.l.]: Springer, 2010. P. 65–74.

_____. Firefly algorithms for multimodal optimization. In: SPRINGER. INTERNATIONAL symposium on stochastic algorithms. [S.l.: s.n.], 2009. P. 169–178.

YANG, Xin-She; DEB, Suash. Cuckoo search via Lévy flights. In: IEEE. 2009 World congress on nature & biologically inspired computing (NaBIC). [S.l.: s.n.], 2009. P. 210–214.

YEH, I-Cheng; YANG, King-Jang; TING, Tao-Ming. Knowledge discovery on RFM model using Bernoulli sequence. **Expert Systems with Applications**, Elsevier, v. 36, n. 3, p. 5866–5871, 2009.

ZHANG, Jing-Ru et al. A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training. **Applied mathematics and computation**, Elsevier, v. 185, n. 2, p. 1026–1037, 2007.

ZHANG, Le; SUGANTHAN, Ponnuthurai N. A survey of randomized algorithms for training neural networks. **Information Sciences**, Elsevier, v. 364, p. 146–155, 2016.

ZHANG, Qingyang et al. Collective decision optimization algorithm: A new heuristic optimization method. **Neurocomputing**, Elsevier, v. 221, p. 123–137, 2017.

ZHANG, Yudong et al. Fruit classification by biogeography-based optimization and feedforward neural network. **Expert Systems**, Wiley Online Library, v. 33, n. 3, p. 239–253, 2016.